MASTER IN COMPUTER SCIENCE

Curling Strategy Analysis

Comparing different methods to enhance decision-making and ranking calls in simulated curling

Master Thesis

Michael Brunner

University of Bern
Faculty of Science, Pattern Recognition Group
PD Dr. Kaspar Riesen

May 2024

UNIVERSITÄT BERN

UNIVERSITÉ DE NEUCHÂTEL

UNIVERSITÉ DE FRIBOURG
UNIVERSITÄT FREIBURG

Abstract

This thesis investigates the development of strategic decision-making in simulated curling using advanced computational methods. A simulated solution could revolutionize tactical discussions in curling and enhance the strategy of professional curlers.

By focusing on position analysis and the integration of supervised learning and search algorithms, the study aims to enhance the accuracy and effectiveness of strategic calls in curling simulations. The research explores the strengths and limitations of self-learned and supervised learning mechanisms, revealing that supervised neural networks and graph neural networks outperform heuristic approaches and exhibit high accuracy in complex game scenarios. Additionally, the thesis evaluates the performance of search algorithms, combining Monte Carlo Tree Search and minimax, to refine call rankings.

The study underscores the necessity of a substantial and well-curated dataset to improve the robustness and adaptability of these learning models. However, the current datasets are insufficient within the realm of machine learning to enable the bot to consistently outperform professional curlers, particularly in complex mid-end positions. The findings also indicate a hybrid approach, combining different methods to analyze a position, might better address the challenges of a continuous space.

Overall, this thesis provides a comprehensive framework for improving strategic decision-making in simulated curling through advanced computational techniques, offering valuable insights and practical methodologies for future developments in sports analytics and artificial intelligence.

Contents

1 Introduction

In recent years, advancements in artificial intelligence (AI) and machine learning (ML) have provided new opportunities to enhance strategic decision-making in sports. Curling, with its intricate blend of strategy and precision, presents a unique challenge and a fertile ground for applying these technologies. This thesis explores the development and application of advanced computational methods to improve strategic decision-making in curling, focusing on position analysis, supervised learning, and search algorithms. Significant contributions in related work [1–3] have explored similar computational approaches, yet there is still no tool that can be used in professional curling to analyze and optimize decision-making processes.

To develop a bot capable of making strategic decisions in curling, several challenges must be addressed. Firstly, curling operates in a continuous space where the position and trajectory of each stone can vary infinitely, making it difficult for a bot to predict and evaluate outcomes precisely. Secondly, categorizing possible calls is inherently complex due to the vast number of potential strategies and moves available in any given situation. This complexity is compounded by the difficulty a computer faces in understanding the nuanced impact of specific shots on the overall game strategy. Overcoming these obstacles is essential for developing a bot that can respond adequately to dynamic game conditions and make informed, strategic decisions.

The primary goal of this thesis is to establish a baseline for further research into the application of machine learning and search algorithms in curling strategy and performance. In curling, decision-making is pivotal, often determining the outcome of a game. Traditional approaches rely heavily on the experience and intuition of players and coaches. While effective, these methods have limitations, particularly in their ability to process and analyze vast amounts of game data in real-time.

This thesis will thereby focus on position analysis in the context of curling, specifically the task of determining possible calls from a given position. This task is inherently challenging due to the continuous nature of the game space, where an almost infinite number of possible positions and trajectories must be considered. This paper will develop multiple methods for ranking these calls, in an attempt to identify the ones that maximize strategic advantage.

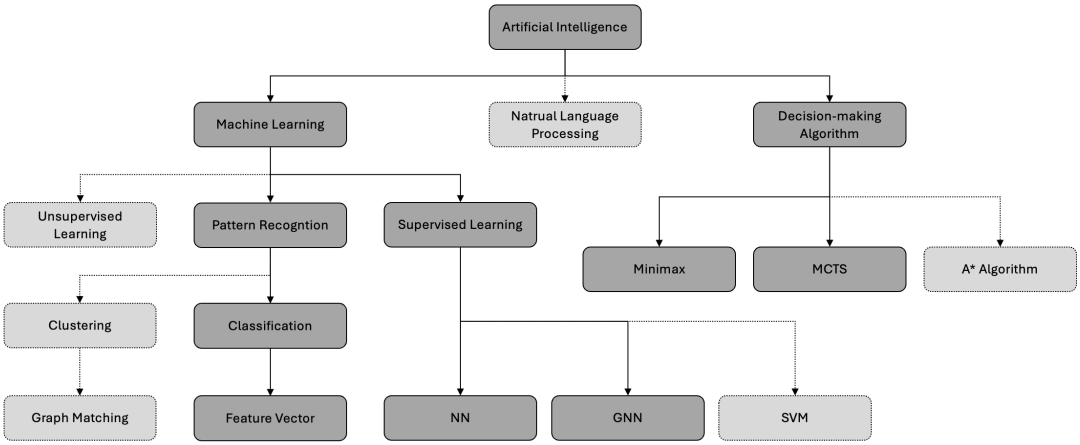

Figure 1.1: Key methods in a broader context.

Curling, often referred to as “chess on ice,” is a strategic sport that combines physical precision with complex tactical decision-making. Each game involves two teams throwing rocks on a sheet of ice towards a target area, with the objective of positioning their stones closer to the center than the opponent’s. The sport is not only physically demanding but also requires players to make strategic decisions under varying conditions, considering factors such as the state of the ice, the positioning of the stones and the score dynamics.

The motivation for this study stems from the potential of AI and ML to transform sports strategy and performance. In curling, decision-making is pivotal, often determining the outcome of a game. Traditional approaches rely heavily on the experience and intuition of players and coaches. While effective, these methods have limitations, particularly in their ability to process and analyze vast amounts of game data in real-time.

Figure 1.1 illustrates the diverse fields of AI explored and utilized in this thesis. The diagram is divided into three branches: Machine learning, natural language processing, and decision-making algorithms, each encompassing various techniques and methods relevant to AI. The specific approaches and their overarching categories used in this paper are shaded in dark grey; other concepts, although providing useful tools, were not employed in this study.

The main focus of this work is position analysis in the context of curling, specifically the task of determining possible calls from a given position. This task is inherently challenging due to the continuous nature of the game space, where an almost infinite number of possible positions and trajectories must be considered. The goal of this research is to develop multiple methods for ranking these calls, ultimately identifying the best call that maximizes strategic advantage.

The primary task is to analyze the current position on the ice and generate potential calls. Supervised learning methods, including neural networks and graph neural networks, are trained to evaluate game positions and propose strategic moves based on learned patterns from expert opinions and simulated gameplay. An investigation into which methods achieve the best results in extracting calls provides the core of this thesis. By focusing on generating and evaluating possible calls from any given position, this study aims to enhance decision-making in curling through precise and effective position analysis.

In addition to this primary focus, the research aims to evaluate the effectiveness and efficiency of different search algorithms, specifically Monte Carlo Tree Search and minimax, in various game situations. Both of these algorithms play a major role in evaluating candidate moves in chess bots, providing a foundation for evaluating different options in the context of curling due to the similar turn-based gameplay. These algorithms will be analyzed and compared based on their performance in identifying optimal calls under different conditions, providing insights into their strengths and limitations in the dynamic and strategic environment of curling.

Another key objective is to develop a position classification system capable of matching game positions to similar scenarios. This system could enable the direct calculation of win probabilities or the transfer of strategic calls from one position to another, thereby improving the accuracy and efficiency of strategic decision-making. By leveraging historical data and advanced classification techniques, the research seeks to create a robust framework for understanding and exploiting positional similarities in curling, ultimately enhancing the predictive and strategic capabilities of the simulation model.

The long-term goal is to develop a bot that not only competes against but potentially beats professional players. Additionally, this bot should be able to calculate win probabilities for specific calls in given positions, offering a strategic tool that professional curlers can utilize to refine their gameplay.

The thesis is structured into several key sections, each addressing different aspects of the research and development process. Chapter 2 lays the theoretical foundation for the thesis. It begins with an overview of curling rules and basic strategic principles, highlighting the importance of accurate decision-making in the sport. The chapter then delves into various AI and ML concepts, focusing on supervised learning and pattern recognition, neural networks, graph neural networks, and search algorithms such as minimax and Monte Carlo Tree Search. These theoretical insights form the basis for the methodologies developed in subsequent chapters.

Chapter 3 describes the methodologies and approaches developed for the curling bot. It covers the simulation environment, including the game logic, physics engine, and player accuracy. The chapter details the bot's decision-making process, from position analysis and curl calculation to free path checking and the application of search algorithms. Various mechanisms are explored and implemented, providing a comprehensive overview of the bot's capabilities.

Chapter 4 presents the findings from the evaluation of different components within the curling game simulation. It includes an analysis of the learning bots' accuracy, a comparison of search algorithm parameters and the impact of database expansion on position classification. The chapter also compares the performance of NN and GNN configurations, as well as examines the bots' effectiveness in gameplay scenarios.

The concluding chapter summarizes the key findings of the research, discussing the strengths and limitations of the developed methodologies. It outlines potential improvements and future research directions, emphasizing the importance of continuous data collection, feature representation, and algorithm optimization. The chapter also reflects on the broader implications of the study for the field of sports analytics and AI.

2
Theory

This chapter provides an overview of the theoretical foundations essential for this thesis. It begins with Section 2.1, which offers an introduction to the sport of curling, setting the stage for the subsequent technical discussions. Section 2.2 explores the realm of machine learning, with an emphasis on two key strategies: supervised learning and pattern recognition. Following this, Section 2.3 delves into search algorithms, specifically focusing on minimax and Monte Carlo Tree Search, which are pivotal to the methodologies employed in this research. Finally, Section 2.4 integrates these diverse theoretical components, highlighting their interconnections and relevance to the overall research objectives.

2.1 Curling

Curling is a strategic and highly competitive sport that originated in Scotland. This subsection briefly explains the rules and strategic basics of curling relevant to gameplay and tactics, as well as win probabilities from historical data. Technical aspects are omitted as they are not relevant to the thesis.

The game is played between two teams, each consisting of four players, with the objective of scoring more points than the opponent. A typical game consists of eight to ten ends, depending on the competition [4]. In each end, teams alternate turns until each has thrown eight rocks. Each player has a specific role during their turn: one player throws the rock, two sweep and the last, called the Skip, holds the broom and calls the shots from the opposite end of the sheet. According to the World Curling Federation, "*at the completion of an end (when all stones have been played), a team scores one point for each of its own stones located in or touching the house that are closer to the tee [also referred to as pin] than any stone of the opposition.* [4] R12.b)" The last stone advantage in curling is known as the "hammer". If a team scores in an end, they must start the next end without the advantage of the last stone. In the event of a blank end, where no team scores, the team with the hammer retains it for the next end. Understanding the rules of curling is essential for effective game analysis and strategy development. This foundational knowledge sets the stage for exploring advanced topics, such as search algorithms and machine learning strategies in the following sections.

The basic strategy in curling involves several critical factors. One key element is assessing the win probability after each end, considering the score and possession of the hammer. Win probability is

Starting Margin:		-3	-2	-1	0	+1	+2	+3
Starting WP:		5.9	20.5	41.1	65.2	87.0	95.1	98.5
End score	+3	33.2	58.9	80.2	94.8	98.8	99.7	100
	+2	11.0	33.2	58.9	80.2	94.8	98.8	99.7
	+1	3.7	11.0	33.2	58.9	80.2	94.8	98.8
	0	5.2	19.8	41.1	66.8	89.0	96.3	98.9
	-1	1.2	5.2	19.8	41.1	66.8	89.0	96.3
	-2	0.3	1.2	5.2	19.8	41.1	66.8	89.0
	-3	0	0.3	1.2	5.2	19.8	41.1	66.8

Figure 2.1: Win probability with hammer and six ends remaining [5].

crucial as it provides insights into which end scores are advantageous or disadvantageous. Data from *doubletakeout.com* [5], which includes matches from the World Curling Tour, illustrates the win probability between ends. This probability varies based on factors such as the number of ends remaining, the score, and possession of the hammer.

Figure 2.1 presents the win probability data for a team holding the hammer with six ends remaining, based on games played by the top 25 teams in the men's world curling ranking. The first row in each table displays the relative score to the opponent at the start of the end, while the second row shows the win probability at the beginning of that end in percentage. Subsequent rows detail the win probability after the next end based on the scores achieved during the current end. The data illustrates that any score for the opponent is never beneficial when holding the hammer. Generally, the higher the team's own score, the better the win probability, with the exception of scoring a single point. Indeed, this results in the loss of the hammer and can be strategically disadvantageous. These probabilities are crucial for the bot, as they provide a basis for making informed choices that enhance winning chances. Understanding and leveraging win probability is fundamental to strategic decision-making in curling. By analyzing end scores and possession of the hammer, players and bots can make informed choices that enhance their chances of winning.

Another important aspect is making strategic calls based on the specific situation on the ice. A "call" in curling refers to the strategic decision for the next shot. This decision-making process considers several critical factor [6]:

- **Score:** The current game score influences strategic decisions; teams behind in score may take greater risks.
- **End:** The stage of the game, or how many ends are left, slightly influences strategy but emphasizes scoring regardless.
- **Hammer:** Possession of the hammer significantly impacts strategy; teams with the hammer prefer an open field to maximize scoring potential.
- **Shot:** The specific shot being played; early shots set up positioning, while later shots aim to secure points or limit the opponent's score.

- **Position:** The arrangement of stones on the ice, which dictates immediate tactical choices.

Understanding the factors that influence specific calls in curling is crucial for optimizing strategy during a match. While the overall goal is to secure a win by the end of the game, each specific call made during play aims to improve the immediate tactical situation on the ice. To illustrate effective call evaluation to achieve these short-term goals, insights can be drawn from the expertise of Howard, Olympic gold medallist and two time world champion. His strategies provide a framework for understanding how to assess and execute specific calls based on various in-game factors. Both human players and the simulation bot utilize these strategic principles to optimize gameplay decisions. The strategy [6] adjusts based on whether a team is leading or trailing, with riskier calls becoming more common as the game progresses or as a team finds itself at a score disadvantage. The hammer's possession dictates tactical preferences; teams with the hammer aim to keep the playfield open to ensure scoring opportunities, while their opponents strive to congest the house, increasing the likelihood of a steal, a score for the team without the hammer.

In summary, the decision-making process for each call in curling is multifaceted, involving the assessment of various dynamic factors. By understanding and strategically evaluating elements such as the score, end, hammer possession, shot type, and stone positions, players as well as bots can make informed decisions that optimize their gameplay. These principles form the foundation for effective in-game strategy and are crucial for the decision-making process of the different bots developed for this thesis. Other factors such as player accuracy and ice conditions also play roles in strategic planning, but will be neglected as it is programmed for one specific ice and player strength.

2.2 Machine Learning

This section delves into the basics of machine learning [7], focusing on supervised learning implemented with neural networks and graph neural networks, as well as pattern recognition, specifically on matching with a feature vector. Understanding these concepts is essential for developing a bot that is able to analyze and respond to specific positions on the curling sheet with appropriate calls. By leveraging supervised learning, the bot can be trained to recognize patterns and make predictions, while pattern recognition techniques enable the bot to optimally match current game states to historical ones in a database.

2.2.1 Neural Networks

Neural networks have become pivotal in solving complex classification tasks such as image classification [8, 9]. This subsection introduces neural networks and supervised learning, setting the stage for a deeper exploration of their structure and function.

Neural networks consist of interconnected layers of artificial neurons that process data through dynamic network architectures. Typically, these include input, hidden, and output layers:

- **Input Layer:** Receives raw data or features in form of a vector.
- **Hidden Layers:** Perform computations using weighted connections and activation functions to transform input data.
- **Output Layer:** Produces predictions or classifications from the processed data.

Figure 2.2 illustrates a neural network with a fully connected hidden layer. In such networks, every neuron in one layer connects to every neuron in the next, facilitating complex data transformations.

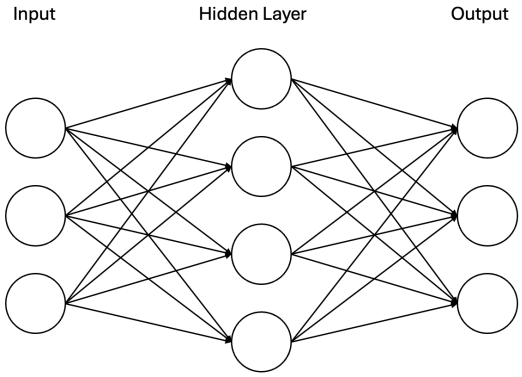

Figure 2.2: Structure of a neural network with a fully connected layer.

In a fully connected layer, each neuron's output h_i is computed as follows [8]:

$$h_i = f\left(b_i + \sum_{j=1}^{n}(x_j \cdot w_{ij})\right)$$

where:

- h_i: Output of the neuron after applying activation function f.
- f: Non-linear activation function.
- b_i: Bias of the neuron h_i.
- x_j: Outputs from the previous layer's neurons.
- w_{ij}: Weights connecting neuron h_i to neurons in the previous layer.
- n: Number of neurons in the previous layer.

Neural networks often utilize supervised learning, where a model is trained to map input data to known output labels using a labeled dataset [10]. The training process involves adjusting the network's parameters, which include weights and biases, to minimize prediction errors. To fine-tune a network, the following optimization techniques are used:

- **Cost Function:** Measures the discrepancy between predicted outputs and true labels, with common examples including mean squared error and cross-entropy loss.
- **Gradient Descent:** An optimization algorithm that iteratively adjusts parameters in the direction that minimally reduces the error, guiding the network towards optimal performance.

As parameters are fine-tuned, the network learns to perform tasks such as mapping images to class labels in image classification, a process detailed through the gradient descent mechanism that iteratively reduces the cost function.

Neural networks, with their robust architecture and learning capabilities, are a fundamental component of machine learning, particularly in classification tasks. Understanding their structure and training

methodologies is crucial for developing advanced algorithms. This foundational knowledge sets the stage for discussing specific applications within curling simulations in subsequent chapters, illustrating how these algorithms adapt and function within the complex environment of sports analytics. In this thesis, the goal is to train neural networks where each training sample consists of an input representing the current position of stones on the ice and an output indicating the optimal call. This approach leverages the neural network's ability to classify and predict outcomes, making them invaluable for creating effective curling bots.

2.2.2 Graph Neural Networks

Unlike traditional neural networks that operate on data in euclidean spaces, graph neural networks [11, 12] are designed for non-Euclidean domains. They excel in managing graph-structured data, enabling the capturing of complex relational information through an innovative process known as message passing. Graph neural networks represent a significant evolution in the realm of deep learning, particularly suited to processing data structured as graphs. These networks are crucial in domains where data exhibits intrinsic relational properties, such as social network analysis, molecular chemistry, and knowledge graphs. This subsection explores the fundamental aspects of GNNs, specifically message passing and pooling, and their relevance to this thesis.

Graphs are mathematical structures [13] defined as $G = (V, E)$, where V is a set of vertices (nodes) and E a set of edges connecting these nodes. Each vertex $v \in V$ possesses its own attributes X_v, and edges may also have their own features and weights. With this definition of a graph, this thesis now delves into the mechanisms of a GNN.

Message passing [11] is a fundamental mechanism in graph neural networks, enabling each node to receive and combine information from its neighbors. This iterative process involves several steps:

1. Each node sends a "message" based on its current state and edge features.
2. The node aggregates incoming messages, often using functions like sum, mean, or max, to capture different aspects of neighborhood information.
3. The aggregated message is then combined with the node's current state to update its features through a transformation, typically a neural network layer.
4. This updated state can be used for further processing or making predictions.

Through multiple rounds of message passing, nodes effectively gather and integrate information from larger neighborhoods, enhancing the GNN's capacity to model complex relationships within the graph. As illustrated in Figure 2.3, a node aggregates features from its neighbors, refining its representation based on the aggregated information. The resulting graph has the same structure, with updated attributes.

This procedure is mathematically defined by the following formula [11]:

$$H_v^{(l+1)} = f\left(H_v^{(l)}, \{H_u^{(l)} : u \in N(v)\}\right)$$

Here, $H_v^{(l)}$ denotes the representation of node v at layer l, $N(v)$ represents the set of neighboring nodes, and f is a differentiable function that aggregates and transforms these features.

Graph convolutional layers [14] extend the concept of message passing by incorporating trainable parameters that allow for more complex transformations of node features. These layers are a cornerstone in learning on graph-structured data, enabling the network to perform convolutions over graphs. Importantly,

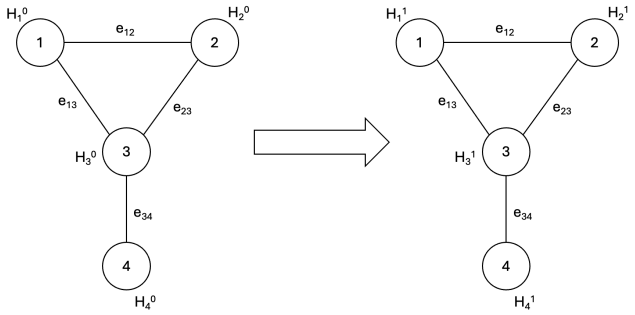

Figure 2.3: Illustration of the message passing mechanism in graph neural networks.

graph convolutional layers are not constrained by the dimensions of the input graph. The fundamental operation of a graph convolutional layer can be expressed by the following formula [14]:

$$x_i' = W_1 x_i + W_2 \sum_{j \in N(i)} e_{i,j} x_j$$

where:

- x_i': Updated feature of node i, with x_i being the original feature.
- $N(i)$: The set of neighbors of node i.
- W_1 and W_2: Trainable weight matrices that transform node features.
- $e_{i,j}$: Edge features between node i and its neighbor j.

This layer effectively captures both the features of the nodes themselves and the influence of their neighborhoods, making it powerful for tasks that require an understanding of local graph structures. By learning optimal values for W_1 and W_2, the graph convolutional network can adapt to the specific characteristics of the graph data it processes, enhancing its predictive performance on tasks such as node classification, link prediction, and graph classification.

Pooling [11] in GNNs is utilized to compress and summarize the features within a graph. Pooling operations enhance the GNN's ability to handle graphs of varying sizes and structures by providing a summarized representation that maintains essential topological and feature-based information. There are two main types of pooling [11]:

- **Hierarchical Pooling:** This progressively coarsens the graph by merging nodes in a hierarchical manner, typically based on clustering or other criteria. It reduces the graph's size while preserving its structural and feature information.
- **Flat Pooling:** In this approach, a fixed-size subset of nodes is selected or aggregated in a single step. Techniques such as global attention or ranking functions are used to identify the most representative nodes, which are then pooled to form a summarized representation.

Figure 2.4 shows the hierarchical pooling process in action, where the graph is pooled to a single node in two steps. In each step, neighborhoods of nodes are pooled into a single node, with the new node

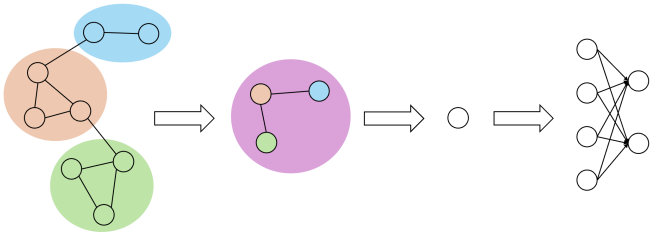

Figure 2.4: Demonstration of pooling in a GNN.

encapsulating the feature information from its constituent nodes. The final step is to transform this single node with many features into a vector that can be classified using different kinds of neural networks. These pooling techniques are critical for tasks requiring a condensed graph-level output and play a significant role in the performance and scalability of GNNs across various applications.

In summary, graph neural networks, with their advanced message passing, convolutional layers, and pooling techniques, provide a robust framework for modeling complex relationships within graphstructured data. These techniques enable GNNs to effectively model and analyze complex relationships within graph-structured data, making them indispensable for advanced machine learning applications in various domains. In this thesis, GNNs are employed to accurately represent curling positions, enabling the bot to make precise and strategic calls. This capability is essential for enhancing the bot's performance and achieving effective gameplay in curling simulations.

2.2.3 Pattern Recognition

Pattern recognition is a fundamental aspect of machine learning, particularly in the field of classification. This section delves into matching algorithms, with a special focus on matching using a feature vector. This method is similar to the k-NN algorithm with $k = 1$, but differs in that each object is treated as its own cluster rather than being part of a larger one. This approach is especially relevant for comparing different positions and finding the most similar ones in a database.

Matching with a feature vector involves identifying and matching objects based on their vectorized attributes [8, 15]. This technique is particularly effective in complex scenarios, such as analyzing curling positions where each rock has additional attributes depending on its location. The first step in the matching process is to extract features that capture the essential characteristics of the objects of interest. In the context of curling, relevant features might include the location, angles, and the visibility from the hack; attributes that are crucial for understanding rock placement and determining strategic calls. Once features are extracted, they are transformed into a standardized vector format. This standardization ensures that each vector is uniform in length, with each element accurately representing a predefined feature, thus facilitating straightforward comparisons between vectors.

To calculate the pairwise distance between objects, a distance metric is needed. This metric is crucial for matching new objects to similar ones in the database. A typical one is the Euclidean distance, which is defined as follows [16]:

$$d(X, Y) = \sqrt{\sum_{i=1}^{n} (x_i - y_i)^2}$$

This formula calculates the straight-line distance in an n-dimensional space between two vectors X and Y, with a smaller distance indicating greater similarity. For the matching process to be effective, the selected distance metric must satisfy several mathematical properties [16]:

- **Non-negativity**: $d(x, y) \geq 0$ for all x, y.
- **Identity of Indiscernibles**: $d(x, y) = 0$ if and only if $x = y$.
- **Symmetry**: $d(x, y) = d(y, x)$ for all x, y.
- **Triangle Inequality**: $d(x, z) \leq d(x, y) + d(y, z)$ for all x, y, z.

In this thesis, a weighted distance metric is used to account for the varying importance of different features. This approach ensures that more important features have a greater influence on the distance calculation, thereby improving the accuracy of the matching process.

In practical applications [15], each new object, represented as a feature vector, is compared pairwise with every vector in a database using the distance metric. The object is matched to the database object with the smallest computed distance, optimizing for the highest similarity. This process enables the comparison of win probabilities of new positions with historical data, facilitating strategic decisions. Attributes such as optimal or candidate calls can also be associated with each position in the database, and these recommendations can be transferred to new positions based on matching results.

In summary, pattern recognition using feature vectors is a powerful technique for classifying and analyzing positional data in curling. By extracting relevant features, representing them as standardized vectors, and using distance metrics for matching, we can effectively compare new game states with historical data. This method enables the bot to make accurate and strategic calls, significantly enhancing its decision-making capabilities in curling simulations.

2.3 Search Algorithms

Search algorithms are essential tools in evaluating and comparing strategic decisions in curling. This section focuses on two prominent algorithms: Minimax and Monte Carlo Tree Search. These algorithms are highly effective in exploring possible game states and determining optimal calls, thereby enhancing decision-making processes in curling simulations. By systematically analyzing potential moves, search algorithms provide a robust framework for strategic planning and optimization.

2.3.1 Minimax Algorithm

The minimax algorithm [17, 18] is a recursive decision-making tool essential in zero-sum games, such as chess [19], where two rational opponents aim to minimize their potential losses under worst-case scenarios. This makes it an excellent algorithm to assist in the decision-making process for the curling bot. The game is modeled as a tree, with nodes representing game states and edges representing possible moves. Additionally, the explanation will also delve into the application of alpha-beta pruning, an optimization technique that enhances the efficiency of the minimax algorithm by eliminating branches that do not influence the final decision.

Figure 2.5 serves as an illustrative example of a minimax decision tree, where each node represents a game state and edges represent possible moves, with the root node being the current state. The goal is to evaluate which of the first two possible moves is the best response. Starting from the terminal nodes, the

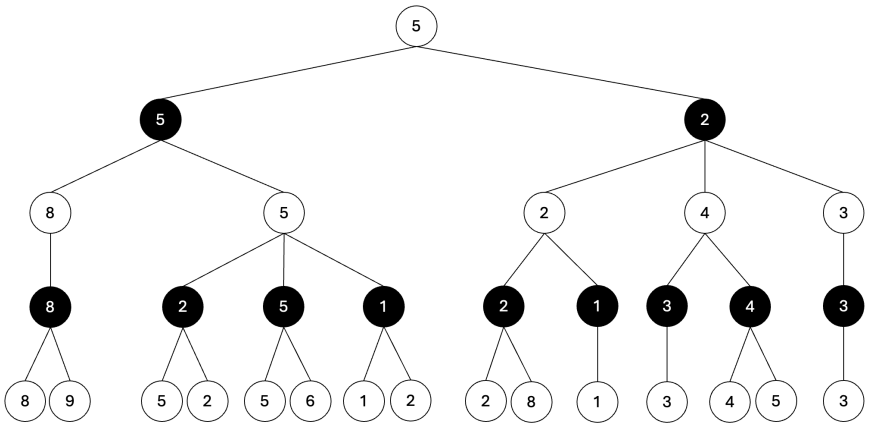

Figure 2.5: Example of a minimax decision tree.

algorithm backtracks, with Maximizer nodes (white) selecting the maximum value from their children and Minimizer nodes (black) picking the minimum value. For example, the leftmost Minimizer node selects 8 as the minimum between 8 and 9, while its Maximizer parent picks 5 as the maximum between 8 and 2. This process continues until the root node, representing the current player's optimal decision. In this example, the optimal move for the Maximizer leads to a final score of 5, by choosing the left move. The minimax algorithm thereby provides a systematic approach to decision-making in zero-sum games by evaluating all possible moves and counter-moves to identify the optimal strategy. By backtracking from the terminal nodes and selecting optimal values at each decision point, the algorithm ensures that the chosen move maximizes the player's advantage while minimizing potential losses. This robust method is particularly useful for the curling bot to make informed strategic decisions.

Alpha-beta pruning [18, 20] is an optimization technique for the minimax algorithm that significantly enhances its efficiency by pruning branches in the game tree that cannot influence the final decision. By eliminating these branches, the computational complexity is reduced, and the decision-making process is accelerated as unnecessary evaluations are avoided.

The alpha-beta pruning algorithm [20], detailed in Algorithm 1, maintains two key values: alpha and beta. Alpha represents the minimum score that the maximizing player can guarantee, while beta represents the maximum score that the minimizing player can ensure. These values are crucial for pruning branches that do not need to be explored because they cannot affect the outcome. The pseudocode has been adapted to fit the specific context of curling. It simulates gameplay until the end of the end is reached and then returns the win probability. This adaptation ensures that the algorithm is directly applicable to the curling scenarios addressed in this thesis, providing a practical and efficient method for strategic decision-making.

To visualize this process, Figure 2.6 depicts alpha-beta pruning using the same tree shown in Figure 2.5. In this depiction, edges in gray are not evaluated further. The algorithm starts by exploring the leftmost branch of the tree, assessing the terminal nodes and propagating the values up to their parent nodes. As the algorithm continues to evaluate the other branches, it uses two values, alpha and beta, to keep track of the minimum and maximum scores that the maximizer and minimizer are assured of, respectively. When the algorithm reaches the rightmost branch, it begins to prune nodes that cannot affect the final decision, focusing on the evaluation of the second choice after the root. The maximizer

Algorithm 1 Minimax with Alpha-Beta Pruning

```
1:  function MINIMAX(node, alpha, beta, isMaximizingPlayer)
2:      if end is over then
3:          return win probability given the new score
4:      end if
5:      if isMaximizingPlayer then
6:          maxEval ← −∞
7:          for each child of node do
8:              eval ← MINIMAX(child, alpha, beta, false)
9:              maxEval ← max(maxEval, eval)
10:             alpha ← max(alpha, eval)
11:             if beta ≤ alpha then
12:                 break
13:             end if
14:         end for
15:         return maxEval
16:     else
17:         minEval ← +∞
18:         for each child of node do
19:             eval ← MINIMAX(child, alpha, beta, true)
20:             minEval ← min(minEval, eval)
21:             beta ← min(beta, eval)
22:             if beta ≤ alpha then
23:                 break
24:             end if
25:         end for
26:         return minEval
27:     end if
28: end function
```

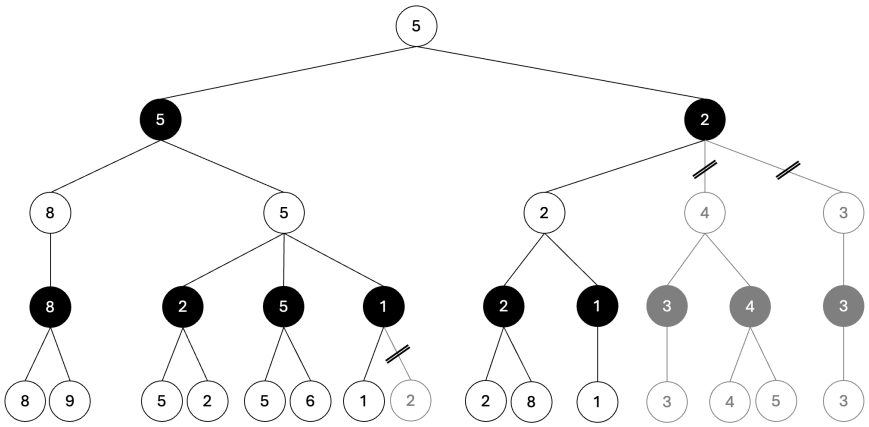

Figure 2.6: Effect of alpha-beta pruning on the minimax algorithm.

would not choose this option, as the previously evaluated move yields a score of 5. The second choice cannot yield a score higher than 2, regardless of the evaluation of the remaining moves by the minimizer. Therefore, by using alpha-beta pruning, the algorithm effectively reduces the number of nodes it needs to evaluate, thereby increasing efficiency. This optimization ensures that only the most promising branches are explored in depth, while less favourable ones are quickly discarded. This method significantly enhances the performance of the minimax algorithm, making it more suitable for complex decision-making scenarios such as those encountered in curling simulations.

The minimax algorithm, enhanced by alpha-beta pruning, is a powerful tool for decision-making in zero-sum games. By systematically evaluating potential moves and counter-moves, it helps in identifying the optimal strategy for each player. In the context of curling simulations, this algorithm is invaluable for evaluating and comparing possible calls, ensuring that the bot can make strategic decisions that maximize its chances of winning. The reduction in computational complexity provided by alpha-beta pruning further enhances the efficiency and effectiveness of the simulation.

2.3.2 Monte Carlo Tree Search

Monte Carlo Tree Search (MCTS) [8, 18, 21] is a powerful search algorithm used to make optimal decisions in uncertain and complex environments. This is the core search algorithm used in related works focusing on the decision-making in curling [1]. Due to the inherent uncertainty in curling, each call must be evaluated multiple times to increase the certainty of the decision. Monte Carlo Tree Search balances exploration and exploitation [21] by iteratively building a search tree based on random simulations of the game.

The Monte Carlo Tree Search algorithm consists of four main steps [22], which are repeated until a computational budget (e.g., time limit) is exhausted. The four steps along with the application in curling, are illustrated as follows:

1. **Selection:** Starting from the current game state, the algorithm selects child nodes, each representing a different possible call. The selection policy balances the exploration of less promising nodes, resulting from executing calls, and the exploitation of nodes with higher win rates.

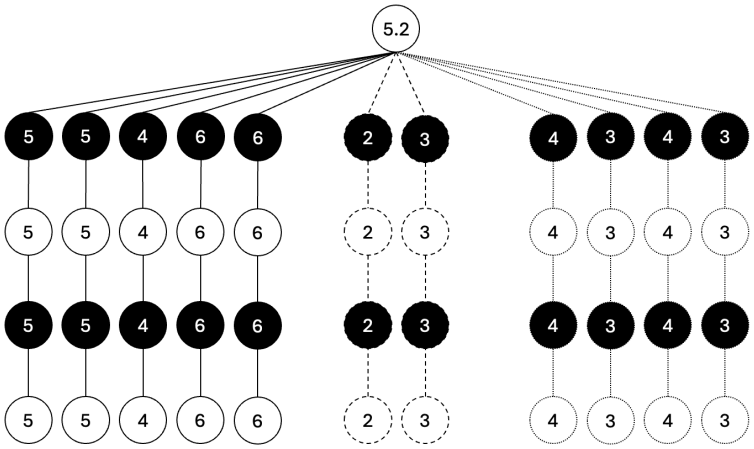

Figure 2.7: Call evaluation using Monte Carlo Tree Search.

2. **Expansion:** Upon reaching a node that has unvisited children, the algorithm expands the tree by adding more child nodes. These nodes represent additional possible moves. Due to the high computational time required for simulating a curling shot, the number of child nodes added is limited.

3. **Simulation:** For each newly added node, the algorithm performs a random simulation to the end of the game. In the context of curling, the simulation runs only until the end of an end, as the win probability for this stage of the game can be definitively determined.

4. **Backpropagation:** The win probabilities of terminal nodes are propagated back up the tree, updating the value estimates and visit counts of the nodes along the path to reflect the outcomes of the simulations. The updated values from the nodes are then used for further selection.

This iterative process allows the Monte Carlo Tree Search algorithm to build a comprehensive search tree that effectively evaluates the potential outcomes of different moves. Figure 2.7 illustrates the Monte Carlo Tree Search process, evaluating three different moves branching from the root node. Each move, depicted with different dashed lines, is evaluated multiple times. This example shows that more promising moves are explored more frequently, while less promising moves are neglected. Through repeated simulations and backpropagation of results, the algorithm refines its estimates, leading to more reliable decision-making. This approach allows the algorithm to compare more promising moves with higher certainty, without expending excessive computational time on less favourable options. In the context of curling, Monte Carlo Tree Search is particularly useful due to the high variability and uncertainty in the game's dynamics. By performing numerous simulations for each potential call, the algorithm can provide a more accurate evaluation of the expected outcome. This helps the curling bot to make informed strategic decisions by considering the most likely results of each possible move.

In summary, Monte Carlo Tree Search is an effective method for handling the complexities and uncertainties of curling. By iteratively simulating game outcomes and updating the search tree, Monte Carlo Tree Search provides valuable insights that enhance the strategic capabilities of the curling bot, ensuring more accurate and confident call evaluations.

2.4 Key Takeaways

This chapter explored the foundational concepts and strategies essential for developing an intelligent curling bot. It began with an overview of the rules and basic strategic ideas of curling, providing the necessary context for understanding the game's dynamics and the decision-making processes involved.

The discussion then delved into various artificial intelligence concepts crucial for the bot's development. Supervised learning with neural networks (NNs) and graph neural networks (GNNs) were highlighted as vital tools for solving complex classification tasks and handling graph-structured data. By training these networks with input data representing the current state of the game and output labels for the best calls, the bot can learn to make accurate and strategic decisions. In the context of curling, GNNs can represent the relational structure of stones on the ice, enabling the bot to understand and predict the best moves in more complex scenarios.

Pattern recognition with feature vectors was also covered, involving the matching of objects based on their vectorized attributes. By extracting relevant features from the game state and representing them as standardized vectors, the bot can compare new positions with historical data to make informed decisions.

Furthermore, the chapter discussed search algorithms, highlighting the minimax algorithm enhanced by alpha-beta pruning. This method provides a systematic approach to decision-making by evaluating all possible moves and counter-moves. Additionally, Monte Carlo Tree Search was presented as an effective method for handling the uncertainty and variability of curling by performing multiple simulations for each potential call, refining the evaluation of moves through iterative simulations and backpropagation.

These AI concepts collectively enhance the bot's ability to analyze game states, predict outcomes, and make strategic decisions. By integrating these methods, the bot can effectively navigate the complexities of curling, optimizing its performance and strategic planning. This theoretical basis covers the essential concepts, and Chapter 3 will explain how these techniques were applied in the context of this thesis.

3 Solution

This chapter provides a comprehensive exposition of the simulation environment and methodologies developed for this thesis. Section 3.1 details the implementation framework of the simulation, highlighting both the computational and infrastructural configurations used. Section 3.2 delves into the analytical techniques and strategic tools available to the computational players, referred to as ”bots”, enabling them to make informed decisions based on varied game scenarios. Section 3.3 explores the differences among the bots in the simulation and describes the methods by which these bots systematically derive strategic decisions, from observed positions and states during gameplay. Finally, Section 3.4 sets up the different methodologies how the bots are tested.

3.1 Simulation

This section details the framework used to implement the simulation environment for this thesis. It covers the core components that enable the simulation to function effectively and realistically. These components include the game logic, which models the rules and mechanics of curling; the physics engine, which simulates the physical interactions and movements of the stones; and player accuracy, which introduces variability to replicate human performance. Together, these elements create a robust and dynamic environment for testing and developing strategic decision-making in curling.

The game logic forms the core of the simulation environment, encapsulating the rules and mechanics of curling. It is responsible for accurately modeling the game's flow and ensuring that all actions adhere to the established rules. The game logic includes the following components:

- **Turn Management:** Each end begins with the initialization of the curling sheet, where the positions of the stones are set to out of play. Each shot in the simulation is executed sequentially, waiting for all stones to come to a complete stop before the next shot is initiated.
- **Strategy Implementation:** Between shots, the bot evaluates the current state of play, assessing the positions of all stones. It then uses predefined strategies to determine the optimal next call, translating this strategy into specific shot parameters such as weight (speed), handle (rotation), and line (launch angle).

- **Scoring System:** After all stones of an end have been played, the game logic calculates the score based on the location of the stones in relation to the pin. The game continues through eight ends, concluding with a tally of the scores to declare a winner. This process provides a complete emulation of a curling match, testing the bots' strategic capabilities.

The implementation of these components sets the foundation for the game logic, ensuring that all actions and outcomes adhere to the established rules of curling. This robust game logic is crucial for maintaining the integrity of the simulation, allowing for accurate rule enforcement and realistic gameplay.

With the basics of the game logic in place, the focus shifts on the physics engine, which is essential for realistically imitating the physical dynamics of curling. The physics engine is a critical component of the simulation, designed to realistically mimic the physical interactions and movements of the curling stones on the ice. This level of detail is essential for creating a realistic and immersive simulation, providing a solid foundation for testing and developing strategic decision-making in curling. The engine processes the game through iterative time steps, meticulously managing key physical interactions and behaviors:

- **Change of Position**: Positions of the rocks are updated each time step based on their velocities.
- **Curling Motion**: Curling motion is simulated by adjusting the velocity vector. As the rock slows, its velocity is scaled by a factor slightly less than 1, and the direction is incrementally rotated to simulate the curl. The degree of rotation increases as the rock decelerates, governed by the following relationship:

$$\text{Rotation} = \begin{cases} \text{sign} \times \frac{0.05}{|V|} & \text{if } v < 100, \\ \text{sign} \times \frac{0.025}{|V|} & \text{if } v < 260, \\ \text{sign} \times \frac{0.01}{|V|} & \text{otherwise.} \end{cases}$$

  Figure 3.1 visualizes the rotation of the velocity vector, showing that large velocities result in almost no curl, whereas smaller velocities produce a much larger one.

- **Velocity Reduction**: The velocity decreases multiplicatively and ceases when insufficient to significantly alter position:

$$V = \begin{cases} V \times \frac{|V|-0.3}{|V|} & \text{if } v < 10, \\ 0 & \text{otherwise.} \end{cases}$$

- **Collisions**: The physics engine accurately calculates the interactions between stones, incorporating momentum, kinetic energy, and the loss of energy due the static friction between the rock and the ice. Friction between the stones is also realistically modeled to enhance the authenticity of rock interactions.

The constants used in the simulation's physics engine were empirically derived through an iterative process of trial and error. Starting with a framework that reflects research on the friction and the curl of a stone [23], the constants were refined through expert evaluation of professional curlers to ensure that the simulation mirrors real-world curling dynamics accurately. To enhance user experience, shots are rendered at accelerated speeds to approximate real-time action, ensuring a continuous and realistic portrayal of rock movement. However, the underlying simulation processes shots much faster, typically completing a throw calculation in approximately 0.03 seconds. This is important for efficient algorithmic performance during strategic decision-making.

To replicate the inherent uncertainties and variability in player performance observed in actual curling games, Gaussian noise is introduced to each shot in the simulation. This method effectively limits the

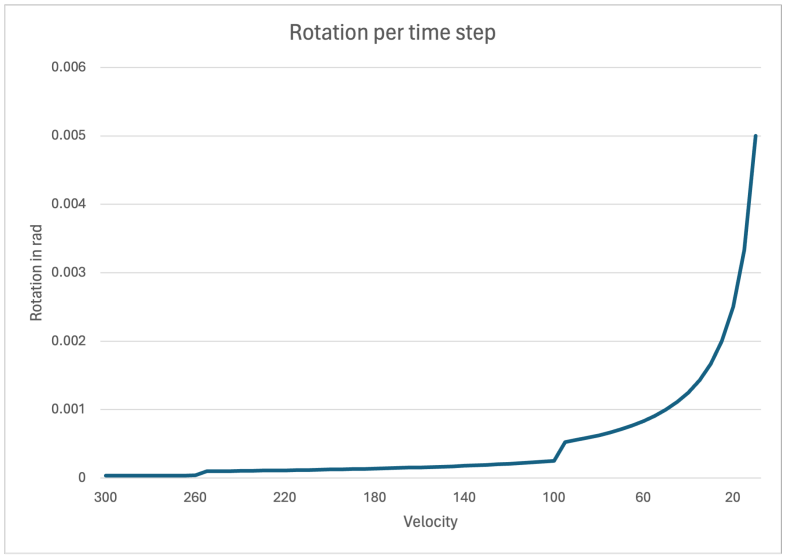

Figure 3.1: Visualisation of the rotation relative to the rocks velocity.

player's ability to consistently deliver perfect shots, thereby creating a more realistic and challenging simulation environment.

The variability in the weight of a shot is modeled using Gaussian distributions, with different standard deviations reflecting the accuracy levels observed among top-level players. This approach captures the natural variability in the strength of each throw.

- For 30% of all throws, the noise level has a standard deviation (σ) of 10 cm, representing high accuracy.
- For 63% of all throws, σ is 30 cm, indicating moderate accuracy.
- For 7% of all throws, σ is 1 m, accounting for occasional significant deviations or mistakes.

The actual weight of the throw is determined by:

$$\text{weight}_{\text{new}} = \mathcal{N}(\mu, \sigma^2)$$

where μ is the intended weight of the throw and σ is the standard deviation based on the above percentages. Figure 3.2 visualizes the distribution curve, centering around the weight the player aims to achieve. This distribution allows players to be accurate most of the time, with a small chance for significant outliers that represent mistakes typical in top-level play. This distribution ensures that most throws are close to the intended weight, with a small chance of larger deviations, mirroring real-world player performance.

The accuracy of the line, or the trajectory, of the shot is also influenced by Gaussian noise. The standard deviation of this noise varies depending on the type of shot, reflecting the difficulty in controlling different types of throws.

- **Take-outs:** These heavier and faster shots have a smaller standard deviation ($\sigma = 5cm$), reflecting their higher accuracy.

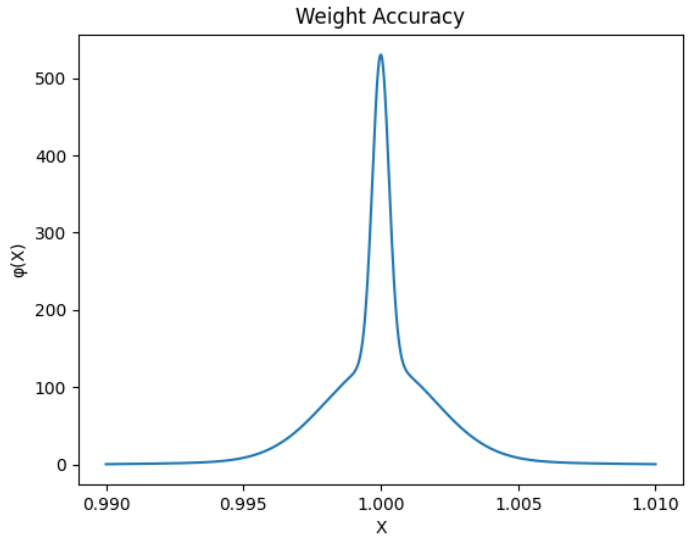

Figure 3.2: Visualization of Gaussian noise on the weight.

- **Draws and Guards:** These lighter and slower shots have a larger standard deviation ($\sigma = 10cm$), representing the greater difficulty in controlling their line due to the higher amount of curl.

This relationship is formalized as:

$$\sigma_{\text{line}} = \begin{cases} 5cm & \text{if } v > 280 \text{ (take-outs)}, \\ 10cm & \text{if } v \leq 280 \text{ (draws and guards)}. \end{cases}$$

where v is the velocity of the throw.

By modeling the line accuracy in this way, the simulation accurately reflects the increased difficulty of executing lighter, more precise shots compared to heavier, more direct shots.

The Gaussian noise models for weight and line accuracy are seamlessly integrated into the simulation. Each shot's intended line and weight are adjusted based on these noise models, ensuring that even the most skilled virtual players exhibit performance variations similar to those of top human players. This integration enhances the authenticity of the simulation, providing a realistic and challenging environment for strategic analysis and training. By introducing controlled variability, the simulation not only mirrors the unpredictability of real-life curling but also tests the robustness and adaptability of the strategies employed by the bots.

3.2 Bot

This section details the components and methodologies used to develop the bot, which is responsible for making strategic decisions during the curling simulations. The bot operates by analyzing the current game position, determining the best possible calls, calculating the required curl for specific shots, checking for

free paths, and using search algorithms to enhance decision-making.

The core mechanism involves extracting a ranking of possible candidates calls from a given position, considering five key factors: score, end, hammer, shot, and rock locations. This mechanism plays a crucial role and will be further discussed in Section 3.3.

To execute a call accurately, the bot calculates the precise line, weight, and handle required. The bot has several options for executing a shot:

- **Draw**: Calculating the necessary line and weight to place the stone at a specific location on the ice.
- **Hit and Stay**: Determining the required amount of curl and initial speed to hit a rock and maintain its position.
- **Freeze**: Computing the line and weight needed to freeze to another rock, including options for corner freezes.
- **Tap**: Assessing the line and weight needed for a straight or angle tap on a rock.
- **Hit and Roll**: Analyzing the angle and speed required to hit a rock and roll to a designated location.
- **Double Take-out**: Calculating the angle and the trajectories necessary to remove two rocks from play.
- **Run Back**: Similar to a double take-out, involving hitting a front rock to travel back and remove a secondary rock.

This comprehensive set of shot options allows the bot to execute any possible shot, at any location, and relative to any rock on the ice. This versatility is crucial for adapting to various strategic scenarios during gameplay.

With the line and weight determined, it is crucial to ensure that the path is unobstructed before executing the chosen shot. This calculation is not only a practical necessity but also fundamental to the strategic integrity of the game. By verifying a clear path, the bot ensures that the intended purpose of each call is fully realized, maintaining the effectiveness and precision of strategic plays.

The procedure involves a two-step process: Trajectory computation and obstacle detection to ensure no other rocks block the path.

- **Trajectory Computation**: The trajectory of the stone is computed based on the initial weight and line settings of the shot. This involves calculating all points along the expected path and using these points to construct a line that represents the projected trajectory of the rock's center. If the intention is to hit another rock, the trajectory line terminates before reaching this rock. This method provides a visual and precise prediction of the stone's path, facilitating strategic planning and enabling crucial adjustments before the shot is executed.
- **Obstacle Detection**: After establishing the trajectory line, the bot calculates the distance from the line to each rock on the ice. If any rock is found to be closer to the line than the diameter of a stone, the path is deemed obstructed; otherwise, it is clear. This step is critical to ensure that the planned trajectory is free of interference, allowing the stone to reach its target without deviation.

Figure 3.3 visualizes this process, where the starting point on the lower left represents the hack, and the curved line depicts the targeted path of the thrown rock's center. The other points represent the centers of all other rocks on the ice. The line is diagonal as the computations are made with predefined points to facilitate faster calculation. These points do not reflect the actual positions on the ice but are relative to other rocks, which is sufficient for accurately determining the free path.

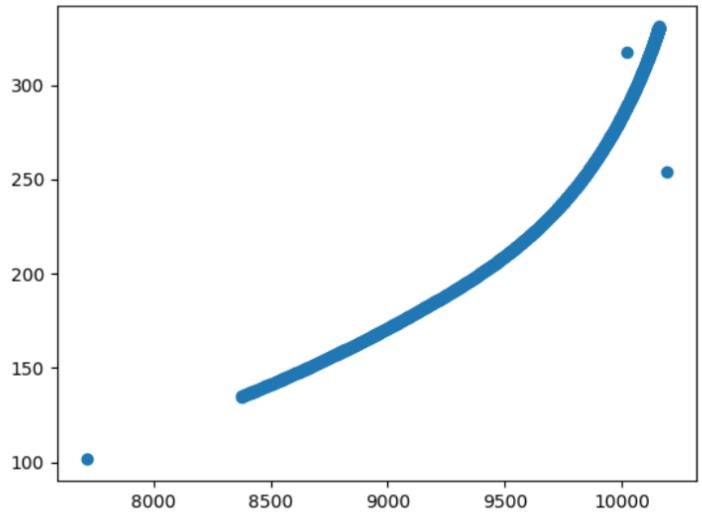

Figure 3.3: Illustration of free path calculation.

Ensuring an unobstructed path allows the bot to execute strategic decisions with higher confidence and precision. This process is vital for aligning the execution of the shot with its strategic objective, maximizing the likelihood of achieving the desired outcome on the ice.

Finally, to enhance a call, search algorithms play a pivotal role in evaluating the potential effectiveness of candidate calls generated during position analysis. To optimize the decision-making process, the two search algorithms are combined. In the first stage, Monte Carlo Tree Search balances how many times each candidate call gets evaluated, ensuring a broad exploration of potential moves. Once a promising subset of calls is identified, the minimax algorithm is employed for a depth search after each candidate call, considering potential counter-moves by the opponent and aiming to minimize potential losses under worst-case scenarios. This hybrid approach leverages the strengths of both algorithms, ensuring a thorough and efficient analysis that enhances the bot's strategic decision-making capabilities.

The search depth extends until an end is completed, as it then becomes possible to use the win probability of intermediate ends. This means that earlier shots in an end have more depth than later shots. Consequently, the choice of search algorithm and the setting of parameters depend on which shot of the end is being played.

The search parameters are adjusted such that every call takes roughly the same amount of computational time. There are three different search parameters that define the search:

- c, representing the number of calls analyzed, usually in a range between three and 20.

- w, indicating the width of each call, or how many times a call gets analyzed, usually in a range between five and 30.
- f, standing for the number of follow-up calls in later arising positions. Due to the time complexity, this is between one and four.

In gameplay, the bot is designed to respond within five seconds, ensuring timely decision-making. By adjusting these parameters, the bot can effectively balance computational efficiency and strategic depth. This adaptability ensures optimal performance across different scenarios, maintaining the integrity and effectiveness of the strategic decision-making process in curling simulations.

These tools, the curl calculation, the free path calculation, and the search algorithms, ensure effective decision-making for each bot. Each bot differs in its mechanism to rank the calls from a given position, which is the most challenging task.

3.3 Position Analysis

This section examines the diverse methodologies utilized by six different bots to generate candidate calls. Each bot adopts a unique approach, ranging from straightforward heuristic methods, implemented with given rules, to intricate neural network-driven strategies. These techniques are engineered to assess the current position on the ice, incorporating factors such as the score and hammer advantage, to propose potential strategic moves. These methods are designed to produce a ranked list of calls; based on the search algorithms, only the top k calls are considered. Calls that are not possible, such as attempting to hit an opponent's rock when no such rocks are in play or if the projected path of the call is not free, are automatically excluded.

Initially, the first two bots rely on heuristic-based approaches, characterized by their rigidity and predefined rule sets, which limit flexibility but ensure quick decision-making. In contrast, the subsequent four bots function within a unified strategic framework that utilizes a list of 305 predefined calls. This extensive repertoire covers a broad spectrum of calls, to have an adequate response for each position, significantly enhancing their capability to analyze and adapt to complex scenarios. Each call within this framework is assigned a unique identifier, streamlining network configuration and analysis.

3.3.1 Naive Bot

The Naive Bot is designed to establish a basic performance baseline within the simulation, facilitating comparisons with more sophisticated competitors. Its strategy is straightforward: the primary choice is to hit the opponent's rock closest to the pin. The secondary choice is to draw to the pin. While these rules do not reflect a deep strategic understanding of the game, they serve as an effective baseline. By focusing on eliminating the opponent's closest rocks, this strategy provides a fundamental challenge that more advanced strategies must surpass to enhance gameplay and increase scoring opportunities.

3.3.2 Bot with Heuristics

The Heuristic Bot employs a predefined set of rules designed to facilitate strategic decisions based on the current score and the stage of the end. This bot serves as a benchmark to assess the effectiveness of static strategy frameworks against more dynamic, learning-based approaches. Its decision-making framework is divided into three main phases: opening calls dictating the initial four rocks of the end, subsequent

Score	Hammer	No Hammer
Leading by two or more points	1. Hit on opponent rock in house 2. Come around behind corner guard 3. Come around behind center guard 4. Draw to the wings of the house	1. Hit on opponent rock in house 2. Come around behind center guard 3. Draw to center top four
Trailing by two or more points	1. Corner guard 2. Come around behind corner guard 3. Come around behind center guard 4. Freeze rock behind center guard	1. Center guard 2. Come around behind center guard 3. Freeze rock behind center guard
Level score	1. Hit on opponent rock in house 2. Come around behind corner guard 3. Corner guard 4. Come around behind center guard 5. Freeze rock behind center guard	1. Hit on opponent rock in house 2. Come around behind center guard 3. Center guard 4. Freeze rock behind center guard

Table 3.1: Strategic plays based on the relative score and possession of the hammer.

strategic calls for the later stages and finally the last rock, which focuses on maximizing the score.

Opening calls are essential for setting the initial strategy for the end. They are methodically categorized based on the current score and whether the bot has the hammer, which determines the strategic focus of the play. The predefined rules for these scenarios are detailed in Table 3.1, demonstrating a systematic approach to initiating play under diverse conditions. One key distinction in strategy is based on whether the bot is leading or trailing in the game. When leading, the bot tends to prefer take-out shots to simplify the end and maintain its lead by reducing the number of opponent rocks in play. Conversely, if trailing, the bot adopts more aggressive strategies, utilizing calls that increase complexity and potential scoring opportunities to change the game's momentum. Another notable difference lies in the bot's use of the hammer. With it, the bot often has more options to play on the sides of the rink, leveraging the last-stone advantage to secure scoring positions away from the center. Without the hammer, the strategy focuses more on central play, aiming to control the key scoring area and limit the opponent's opportunities.

As the end progresses, the Heuristic Bot's strategy adapts to the evolving game dynamics, showcasing its sophisticated decision-making capabilities. The rules become more complex and include strategies such as doubles, clearing, and run backs. These rules are divided into different subsets depending on the current score, the hammer and the hypothetical score of the current end. They are set up similarly to the opening calls but with more cases and up to 15 different calls per case.

Finally, the last call focuses on maximizing the score. Similar to the middle end stage, there are three factors that determine which rules are implemented: the current score, the hammer, and the hypothetical score of the current end. In this case, several calls, such as playing a guard, are neglected as they are never beneficial to the situation. The focus is more on calls like drawing to the pin to secure the end.

3.3.3 Self-Learned Neural Network Bot

The wide range of predefined rules makes the heuristic bot flexible enough to respond accurately in given scenarios, but it is still limited by the rigidity of its predefined strategies. As improvement, the development of the Self-Learned Neural Network Bot employs a systematic and data-driven approach, leveraging extensive simulated gameplay to train a series of neural network models. Each model is specifically tailored for strategic decision-making corresponding to individual shots within an end. This subsection covers the structure of the neural network, which is used for the position analysis within the bot, along with an explanation of the input vector and the data collection process.

The architecture of the neural network is intricately designed to process complex inputs derived from the game's position. The input layer comprises a feature vector with 3,888 entries, where each entry captures a specific aspect of the current game situation. This extensive input is essential for a detailed analysis of the game state. The network architecture includes two linear layers, each containing 100 neurons. These layers are interconnected by a ReLU (Rectified Linear Unit) activation function to introduce non-linearity, enhancing the model's capability to learn complex patterns.

The output layer utilizes a softmax function to transform the linear layer outputs into a probability distribution across 305 potential calls. Each output neuron is associated with a specific call, allowing for a precise mapping of input features to game strategies. This setup enables the bot to evaluate and select the most probabilistically advantageous moves based on the current game conditions. Additionally, alternative configurations of the network are explored to optimize performance and accuracy, the details of which will be discussed in Chapter 4.

The input vector for the neural network is meticulously designed to capture a wide array of features from the game's position, amounting to 3,888 distinct inputs. These inputs provide a rich, detailed view of the game's state, facilitating sophisticated analysis and decision-making. Key features encoded within the input vector include:

1. **Score and Score Range:** This includes a detailed listing of each team's rocks in the house, organized from the closest to the furthest from the pin, accompanied by additional information about the positional structure.
2. **Path Information:** Indicates whether paths are blocked or playable, providing strategic insights into possible moves.
3. **Rock Location:** The precise location of each rock on the ice is recorded, offering critical spatial data for strategy formulation.
4. **Area Indicators:** Identifies whether a rock is situated in key areas such as the house, the four-foot circle, or on the centerline, influencing tactical decisions.
5. **Guarded Status:** Details how effectively each rock is protected by guards, with measures varying by the height and position of the guards.
6. **Frozen Status:** Notes how many rocks are directly located behind each rock, assessing the difficulty of removing the rock from play.

Figure 3.4 visually illustrates how positions on the ice are transformed into this comprehensive feature vector. Each rock, in play or not, is represented by 142 neurons. For rocks not currently in play, all corresponding neurons are set to zero to preserve the accuracy of the data representation. The arrangement of these rocks in the vector is determined by their proximity to the pin. This and other configurations of

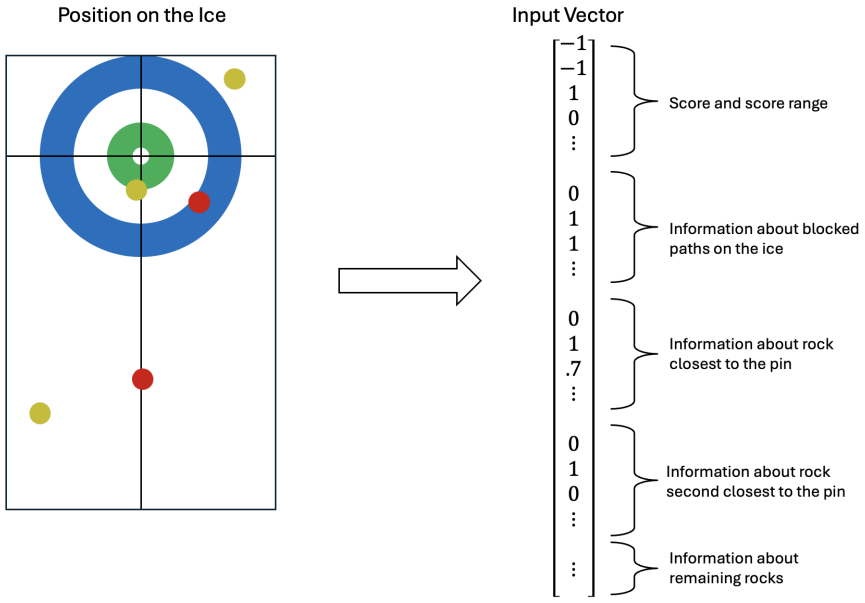

Figure 3.4: Conversion of a position on the ice to its corresponding feature vector.

the feature vector will be further explored in Section 4.1.

During the data collection phase, over 20,000 unique game positions were simulated to encompass a broad spectrum of potential game scenarios. Each position was thoroughly analyzed by playing out each potential call until the best statistical outcome was determined, resulting in a comprehensive dataset of position-call pairs. This dataset forms the foundational training material, enhancing the neural network's ability to make informed decisions across diverse game conditions.

To ensure adaptability by both teams, the dataset focuses exclusively on the team with the red rocks. To simulate a play for the team with the yellow rocks, the bot is programmed to invert the colors of all rocks on the ice and recalculate the feature vector accordingly. This color inversion allows for training a single model that is capable of making optimal decisions regardless of the team color.

Furthermore, to augment the training data without extending the data collection period, each game position was mirrored along the centerline and paired with its corresponding strategic call. This data augmentation technique effectively doubles the dataset size, providing a more varied set of scenarios for model training.

To streamline the training process, each shot in the training phase is evaluated based on a standard end scenario. This means that learning is not tailored to each specific end and score but is rather based on an average win probability that takes into account all possible ends and relative scores. While this approach may limit the initial quality of the bot's call ranking, it is compensated by adjusting the search parameters during actual gameplay to reflect the real-time win probabilities specific to the current end and score.

Models were developed sequentially for each shot, from the last to the first, to precisely address the strategic requirements specific to each rock's context. Figure 3.5 visualizes the process to develope the first three models.

- **Last Rock Model:** Trained on the final shot of the end, this model concentrates on decision-making

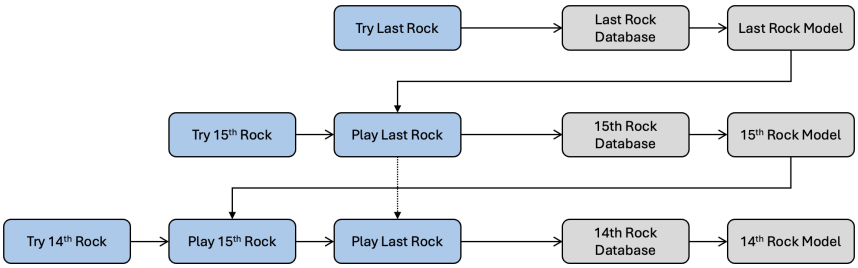

Figure 3.5: Visualization of the self-learning process.

that directly impacts the immediate score outcome, which makes it easy to score each shot.

- **Second Last Rock Model:** Following the optimization of the last rock model, training progresses to the second-to-last rock. In this stage the all possible options for the fifteenth rock are explored, and subsequently playing the last rock by the opposing team as learned before. This way it is possible to assign a value, dependant on the score outcome of the end, to each shot.
- **Earlier Rocks:** Training for progressively earlier shots is handled equivalently as the second-to-last rock. By exploring all possible shots and finishing the end with previously learned models.

Each model utilizes subsets of the initial 20,000 simulated positions, specifically tailored to reflect the number of rocks already played in each scenario. This approach allows the bot to feature 16 distinct models, each finely tuned for optimal performance for specific shots during an end.

This structured, sequential training approach ensures that decisions for earlier shots in the end are influenced by strategies learned for later shots. While this methodology provides an efficient way to train the bot, it also introduces a vulnerability to errors. For instance, an incorrectly classified position in the last shot could lead to an incorrect example in the dataset for the second-to-last rock, potentially propagating errors backward through the training sequence.

In summary, the Self-Learned Neural Network Bot leverages a comprehensive and systematic approach to strategic decision-making. By training neural networks on extensive simulated data and using a sequential, rock-specific method, the bot develops sophisticated strategies tailored to each game scenario. Despite potential vulnerabilities, this methodology allows for nuanced and highly adaptive gameplay.

3.3.4 Supervised Neural Network Bot

The Supervised Neural Network Bot shares the same architectural framework as the Self-Learned Neural Network Bot but is distinguished by its training methodology and data utilization. It focuses on data labelled by experts to refine strategic decision-making capabilities, harnessing the insights of experienced curling professionals. This bot utilizes an identical feature vector and neural network structure to that outlined in Subsection 3.3.3. It also employs the same color-switching and mirroring technique, minimizing the effort required for dataset preparation and ensuring consistency across various game scenarios, but differs in the method of the data collection.

Unlike the Self-Learned Bot which autonomously generates strategic calls through extensive simulation, however, the Supervised Neural Network Bot is trained on a dataset which includes over 12,000 positions, each rigorously evaluated by seasoned curlers who have identified the best call based on their

knowledge and strategic insights. This expert-guided methodology ensures that the training data reflect advanced strategic thinking and adhere to the competitive standards of curling. The training process for this neural network involves fine-tuning the model to align closely with expert-labeled outputs, effectively learning to replicate expert strategic decisions. Like its self-learned counterpart, the Supervised NN Bot is equipped with 16 individual models, each specifically fine-tuned for distinct shots within the game. This targeted approach ensures that the bot's strategies are both precise and tailored to the various stages of the game, in the same way a human would strategize.

By leveraging expert knowledge for its training, the Supervised NN Bot not only mimics high-level human play but also integrates these expert insights into its algorithmic decisions. This approach provides a robust framework for the bot, ensuring that each decision it makes is grounded in well-established strategic principles and expert validation.

3.3.5 Graph Neural Network Bot

The Graph Neural Network Bot builds upon the foundational data and training strategies employed by the Supervised Neural Network Bot but incorporates a graph-based model architecture. This advanced approach is specifically designed to more effectively analyze the relational dynamics between rocks on the curling sheet. By representing rocks as nodes and their connections as weighted edges, the GNN bot processes the game state through multiple layers of graph convolutions, capturing complex dependencies and spatial relationships to inform strategic decisions.

By using the same extensively annotated dataset as described in Subsection 3.3.4, this bot ensures consistency in training quality while exploring new dimensions of data representation. Similar to the Supervised NN Bot, the GNN Bot is equipped with 16 distinct models, with each dedicated to optimizing decision-making for a specific shot in the end. This differentiation ensures that strategies are finely tuned to the unique tactical requirements of each shot, from the opening to the final positions of an end. The key difference lies in the representation of the position: here, the position is converted into a graph, and a graph neural network is trained to accurately respond with the optimal call.

In the graph neural network architecture of the bot, each node within the input graph represents a rock on the curling sheet. These nodes are intricately defined by several attributes that capture the essence and strategic significance of each rock's position. Key attributes for each rock include:

- **Location:** The coordinates of the rock on the ice, relative to the pin.
- **Area Indicator:** A categorical attribute indicating whether the rock is positioned in strategic areas such as the house, the four-foot circle, or on the centerline.
- **Guarded Status:** Two numerical values ranging from 0 to 1, reflecting how well the rock is guarded, with separate measurements for the in-turn and out-turn approaches.
- **Frozen Status:** A numerical value between 0 and 1 indicating the extent to which the rock is frozen to another, affecting its mobility and strategic removal options.

Weighted edges are established between nodes to depict the relational dynamics among rocks. The weight of an edge is determined by the strength of the relationship between rocks. That is, heavier weights are assigned to edges where rocks exhibit a stronger connection, such as being in similar lines which may indicate a guarding relationship or when rocks are in close proximity to each other. This graph structure allows the bot to analyze the position and interpret complex rock relationships. Figure 3.6 provides an

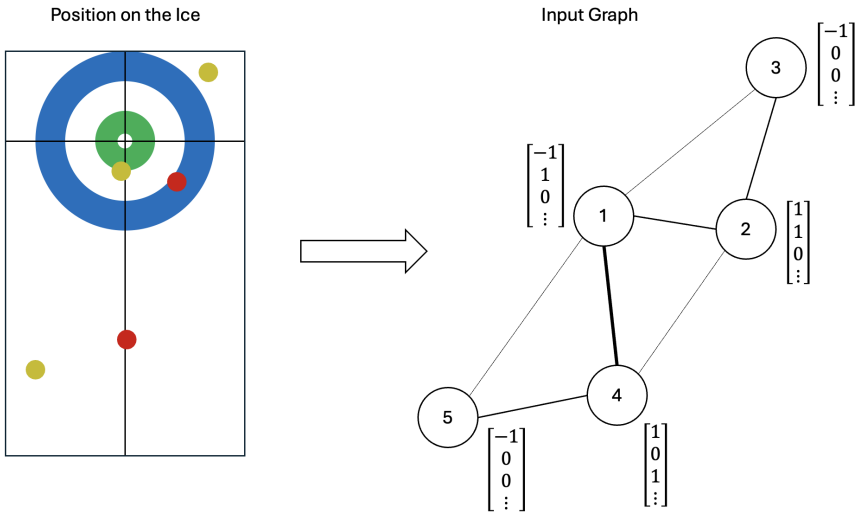

Figure 3.6: Conversion of an example position to its input graph.

example of the conversion from the current position into a graph.

With a clear understanding of the input for the GNN, the next step is to explain the architecture of the GNN. The GNN Bot utilizes a specialized architecture designed to process graph-structured data, enabling sophisticated analysis of the relationships between rocks in curling:

- **Graph Convolution Layers:** The core of the model consists of two graph convolution layers that transform the input features (attributes of rocks and their positions) into higher-level abstract representations. These layers leverage the message passing paradigm to facilitate the flow of information between nodes, thereby capturing the intricate relationships among the rocks.
- **Pooling and Output Layer:** After processing through the graph convolution layers, node features are aggregated using global mean pooling, which condenses the information from all nodes into a single vector. This vector is then processed through a linear layer to project the features into an output space suitable for classification. The final output is processed by a softmax function, which converts the linear layer outputs into a probability distribution over possible strategic calls, indicating the most advantageous moves given the current game state.

The forward pass of the GNN orchestrates data through these components: graph convolution layers, pooling, and fully connected layers. This structured data flow ensures that the GNN Bot can quickly assess game dynamics and make informed decisions, reflecting a deep understanding of the ongoing strategic interactions.

The training process for each of these models involves iterative optimization, where the model is exposed to all positions from the training dataset specifically tailored for that shot. This approach allows the models to learn from a wide range of scenarios, enhancing their ability to generalize and apply strategic knowledge effectively under varying game conditions. During training, each model cycles through the dataset, adjusting its parameters to minimize prediction errors and improve its accuracy in forecasting the most effective call.

By leveraging graph neural networks, the GNN Bot enhances its ability to understand and analyze the complex relational dynamics between rocks on the curling sheet. This advanced approach, combined with the meticulous training of 16 distinct models for each shot, ensures that the bot can make highly informed and strategic decisions. The integration of graph-structured data processing allows the GNN Bot to offer a sophisticated and robust solution for strategic decision-making in curling simulations.

3.3.6 Database Bot

The Database Bot represents the final approach in this thesis, leveraging a comprehensive database of positions, each linked with a call ranking chosen by experts. The mechanism involves three key steps: feature extraction, finding the closest match using a weighted metric, and transferring the call to the new position. Additionally, this thesis investigates the number of distinct positions that exist according to the given metric.

Feature extraction is the foundational step in preparing game data for effective comparison. The process involves distilling complex game situations into a standardized form, or feature vector, that quantitatively describes key aspects of a position:

- **Score** and **Score Range**: Current and potential future score based on the ongoing end.
- **Rock Count**: The number of red and yellow rocks overall, as well as those within the house.
- **Rock Proximity**: Distances of each rock from the pin, providing a spatial analysis of stone placement.
- **Tee Line**: Indicators of whether or not rocks are in front or behind the tee line.
- **Guard and Freeze Metrics**: Quantitative measures from 0 to 1 indicating how well a rock is guarded or frozen, reflecting defensive and offensive potentials.
- **Guard Count**: Total number of guards in play, and in certain ares, influencing the strategic approach to rock protection.

These features are designed to encapsulate all necessary information to allow for a comprehensive assessment of the game state, ensuring that every crucial element is considered in the decision-making process.

With a uniformly defined feature vector, it is possible to compare two vectors and compute their distance relative to each other. Vector comparison in the curling simulation is executed using an adapted Euclidean distance method that prioritizes more influential game aspects by adjusting the importance of less crucial features. This specialized approach ensures that strategic decisions focus on the most impactful elements of the current position:

- **Weighted Importance:** Weights are assigned to various features based on their relative importance in influencing game outcomes. For instance, rocks positioned closer to the center of the house are deemed more critical and thus assigned higher weights compared to rocks that are farther away or outside the house. Specifically, the guarded status of the shot rock (most important rock) is given a full weight (factor of 1), while lesser strategic rocks, for example the fifth rock, receive a reduced weight (factor of 0.5).
- **Distance Calculation:** The modified distance calculation integrates these weights, enhancing the focus on key strategic elements. This method is not only more relevant but also ensures a more accurate match to historical data, as it amplifies the influence of critical game factors over peripheral ones.

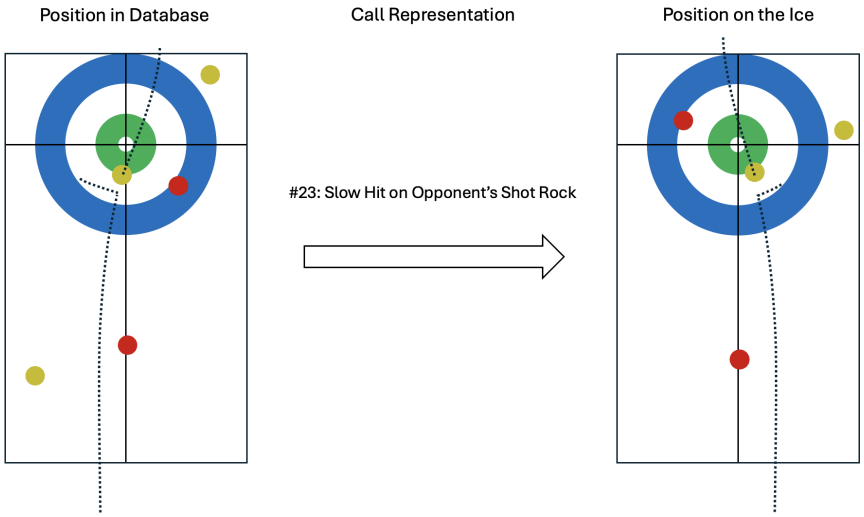

Figure 3.7: Transferring a call from the database to the current position.

The formula used for this adapted Euclidean distance is expressed as:

$$D(X,Y) = \sqrt{\sum_{i=1}^{n} w_i \cdot (x_i - y_i)^2}$$

where $D(X,Y)$ is the distance between vectors X and Y, w_i denotes the weight assigned to the i^{th} feature, and x_i and y_i are the feature values for vectors X and Y respectively.

This weighting and calculation method allows the bot to swiftly locate the most analogous past position within the database. Upon identifying the position most similar to the current one, the bot adopts the associated ranked calls. It then fine-tunes these strategies to fit the specifics of the new situation, illustrated in Figure 3.7.

To make future improvements, this thesis also investigates how many possible arrangements with the previously defined metric exist. To enhance the robustness and comprehensiveness of the database, a continuous expansion strategy is employed. This involves observing and analyzing new positions during actual games. Each new position is compared to existing entries in the database:

- **Threshold for Similarity:** A predetermined threshold determines whether a new position is similar enough to those already in the database. If no existing entry matches the new position closely enough (as defined by the similarity threshold), the position is added to the database.
- **Continuous Learning:** This ongoing process not only expands the database but also refines the bot's ability to handle a wider variety of strategic scenarios, enhancing its adaptability and accuracy over time.

By integrating new positions into the database, the bot progressively becomes more adept at recognizing and responding to diverse and complex game situations. This dynamic approach to database management ensures that the bot remains current with evolving strategies and conditions in competitive curling. This

strategy of expanding the database through real-time game observation underscores a commitment to continuous improvement and adaptability, ensuring that the bot remains a valuable tool for strategic analysis and decision-making in the sport of curling.

3.4 Test Setup

This section details the methodologies and configurations used to evaluate the various components of the developed bot within the curling simulation. The setup includes the evaluation of the positions analysis, search algorithms, position classification accuracy, along with an evaluation of each bot in gameplay to test its adaptability in game scenarios.

The evaluation of learning bots involves testing three different bots: the Self-Learned NN bot, the Supervised NN bot, and the GNN bot. Each bot's performance is assessed by how well it handles decision-making across various stages of a curling game. Each test is conducted on 100 positions, selected to focus on different stages of an end, such as early, middle, and late stages. Positions were selected by top curlers who defined the ”correct” call for each position, and additional ”acceptable” calls were noted, reflecting the subjective nature of strategy in curling. This ensures that the bots are evaluated based on high-level strategic thinking.

The impact of different configurations on the performance of neural network models is also explored. This includes varying the size of the neural networks and adjusting the input vectors. Different structures, such as the number of layers and neurons per layer, are tested to identify the optimal network size that balances complexity with the risk of overfitting. The three different models that were tested contain different number and sizes of hidden layers, all equiped with a ReLU activation function and a softmax at the output layer.

- Final Model: Two hidden layers of size 100 each.
- Extended Model: Two hidden layers of size 500 each.
- Reduced Model: One hidden layer of size 20.

Two different configurations of input vectors are assessed, specifically, one with random rock order and one with a fixed method to order the rocks in the input vector. The order is straightforward, where the rock closest to the pin gets assigned as the first rock in the feature vector, the second-closest gets assign the second spot and so forth.

To compare the accuracy of NNs and GNNs in handling different game scenarios, a test setup was created that categorized positions by their complexity. This involved defining positions with varying numbers of rocks and strategic intricacies. The test set included simpler scenarios with fewer rocks and straightforward strategies, as well as more complex configurations with multiple rocks and intricate strategic considerations. Each bot was tested on these categorized positions to evaluate their ability to accurately predict optimal moves. The comparison aimed to determine how the structural differences between NN and GNN affect their performance across a spectrum of position complexities. This setup provided a basis for analyzing which type of neural network is more effective in different strategic contexts, where both networks are trained on the same dataset.

The evaluation of search algorithms focuses on comparing the effectiveness of Monte Carlo Tree Search and minimax in enhancing strategic decision-making. Different setups of parameters are tested to compare their effectiveness, including the number of analyzed calls (c), the maximum number of times a

call gets analyzed (w), and the number of follow-up calls in later positions (f). The success rates of the search algorithms are compared against decisions recommended by the Supervised Neural Network Bot to understand how search algorithms help the bot align with expert strategies.

Timing analysis measures the computational time required for each parameter setup under various game conditions. Execution times are recorded for each algorithm during different stages of the game, on a standardized set of positions with predetermined conditions to ensure consistency. The analysis aims to determine how the stage of an end (how many rocks are remaining), and the chosen parameters c, w and f affect the performance and thoroughness of strategic analysis. The goal is to balance thorough analysis with efficient processing to ensure quick decision-making during gameplay.

The accuracy of the bots' ability to classify positions during gameplay is examined by assessing the accuracy with different database sizes and the number of rocks in the position. The database is expanded through games played between two bots, with each position classified before every shot. If the dissimilarity to the most similar position in the database is greater than a threshold (σ), the position is added to the database to enhance future classification accuracy. The threshold σ is is set to one, aligning with expert opinion that such a call transfer still yields the optimal call.

To comprehensively evaluate the adaptive capabilities of the bots, a detailed tournament setup was designed. Each bot played multiple matches against every other bot, including self-matches, to assess their strategic flexibility and overall performance in diverse gameplay scenarios. These matches are summarized in two tournaments:

1. **With Search Algorithms:** In this setting, bots utilize their full potential by employing search algorithms to determine the best strategic moves, showcasing their maximum strategic capabilities.
2. **Without Search Algorithms:** Here, bots execute shots based on their immediate position analysis, providing critical insights into the strengths and weaknesses of each bot's decision-making process without the aid of advanced search techniques.

This setup allowed for an in-depth examination of how each bot adapts to different opponents and strategies. Additionally, to understand how well the bots' strategies align with human gameplay, they strongest bot was tested against human players of varying expertise, from professional curlers to complete novices. The human opponents were categorized as follows:

- **Professional Curler:** Competes international levels.
- **Experienced Amateur:** Possesses significant experience but does not compete professionally.
- **Casual Player:** Started curling within the last three to five years.
- **Non-Curler:** Lacks any prior experience with curling.

These matches aimed to provide insights into the effectiveness of the bots' decision-making algorithms when faced with real-world strategic challenges.

This comprehensive test setup ensures a thorough evaluation of the curling game simulation's various components, providing valuable insights into the effectiveness and efficiency of the implemented strategies and algorithms.

3.5 Analysis

In this chapter, the development and implementation of various bots for the curling game simulation were detailed. The Naive Bot and Heuristic Bot serve as baselines, using simple strategies to make decisions. The Self-Learned Neural Network Bot and Supervised Neural Network Bot leverage neural networks to enhance strategic decision-making, with the former relying on extensive simulated gameplay and the latter on expert-annotated data. The Graph Neural Network Bot advances this approach by employing a graph-based architecture to capture the relational dynamics between rocks more effectively. Finally, the Database Bot utilizes a comprehensive database of pre-ranked positions to inform its strategic choices, incorporating a continuous expansion strategy to improve over time.

Each bot's methodology, from feature extraction to strategic call ranking, highlights the diverse approaches used to tackle the complexity of curling gameplay. By integrating advanced machine learning techniques and continuous learning mechanisms, these bots collectively demonstrate the potential for AI to make informed and strategic decisions in dynamic, real-world scenarios. The next chapter will present the results of evaluating these bots, providing insights into their performance and effectiveness in various aspects of the game.

4 Results

This chapter presents the findings from the evaluation of various components of the curling game simulation, including position analysis, search algorithms, and position classification. The accuracy of the learning bots is detailed in Section 4.1. Next, Section 4.2 compares the performance of different search algorithm parameters. The accuracy of position classification, as explained in the database bot, and the impact of database expansion are examined in Section 4.3. Finally, Section 4.4 offers a comprehensive comparison of the bots in gameplay and includes tests against curlers with varying levels of expertise.

4.1 Position Analysis of Learning Bots

This section evaluates the performance of three different bots: the Self-Learned NN bot, the Supervised NN bot, and the GNN bot. The analysis compares how each bot handles decision-making across various stages of a curling game, with specific attention to the quality of their strategic calls. Additionally, different structures of NNs are compared.

The performance of each bot in correctly identifying the optimal or acceptable calls was quantified and is presented in Table 4.1. The table shows the percentage of times each bot made a call that matched the curlers' chosen strategies.

The data reveals several trends regarding the performance of the bots during different stages of the end. The accuracy generally increases at later stages of the end due to two primary reasons. Firstly, at these

Test Condition	**Self-Learned NN**		**Supervised NN**		**GNN**	
	Correct	Acceptable	Correct	Acceptable	Correct	Acceptable
Fifth Rock	23%	41%	41%	80%	32%	79%
Ninth Rock	12%	51%	33%	69%	35%	73%
Second Last Shot	64%	87%	82%	90%	83%	88%
Last Shot	95%	98%	93%	96%	89%	96%

Table 4.1: Percentage of correct and acceptable calls made by each bot purely by the position analysis.

Configuration	Ordering Input	Accuracy Training Set	Accuracy Test Set
Final Model	Yes	97%	95%
Extended Model	Yes	99%	82%
Reduced Model	Yes	82%	80%
Final Model	No	62%	49%
Extended Model	No	88%	38%
Reduced Model	No	53%	41%

Table 4.2: Performance of various neural network configurations on the training set and the test set.

stages, a bot can evaluate more calls using a search algorithm, which allows for more thorough analysis and better decision-making. Secondly, the strategy in the later stages is usually more straightforward, focusing mainly on either scoring more points or limiting the opponent's ability to score points. In contrast, the performance tends to decline during the middle stages of the end. This is attributed to the more complex strategies involved, making it more challenging for the bots to make optimal decisions. The complexity and variability of potential outcomes in these stages require more sophisticated analysis, which the current systems might struggle to handle effectively. The accuracy for earlier shots often shows a significant gap between choosing the optimal call and an acceptable call. This is primarily due to the fewer rocks involved, making it easier to identify an acceptable call.

This analysis highlights the capabilities and limitations of each bot. While the Self-Learned NN bot performs well in the final shots, it struggles in the earlier stages of the game. Both the Supervised NN and the GNN bots show similar performance compared to each other during all stages of the end. The drop in performance of the Self-Learned NN bot is explainable as the learning of earlier stages of the game is dependent on the performance of later shots. In contrast, the Supervised NN bot and the GNN bot benefit from expert-provided training data, making them less dependent on the sequence of learning.

When examining individual positions, it is clear that bots have more difficulty recognizing good calls in more complex scenarios. This is because there are more possible options to consider, and some calls are not detectable by the bot due to the limited number of predefined calls. Overall, the results underscore the complexity of strategic decision-making in curling, where later stages of the game allow for more precise and effective calls. The findings also affirm the significant impact of search algorithms in enhancing decision-making quality, suggesting avenues for future improvements in bot design and strategy development.

Next, the thesis explores the impact of different configurations of the input vector and neural network architectures on the performance of models used in curling bot simulations. As the input is deemed more important, two different input vector are tested across various structures of neural network, summarized in Table 4.2. The results indicate that while the complexity of the neural network structure can influence performance, careful consideration must be given to the risk of overfitting, especially when training data is limited. The chosen network configuration, with two hidden layers of 100 neurons each, provides a good balance between model complexity and the need to prevent overfitting, ensuring robust and reliable performance across different gameplay scenarios. Additional data might allow for a larger network that can incorporate more nuances in complex positions.

The choice of the input vector is crucial in effectively capturing all relevant aspects of a curling game's state. Simple image-based inputs are inadequate for representing the strategic complexity of the sport, necessitating a more detailed feature-based approach:

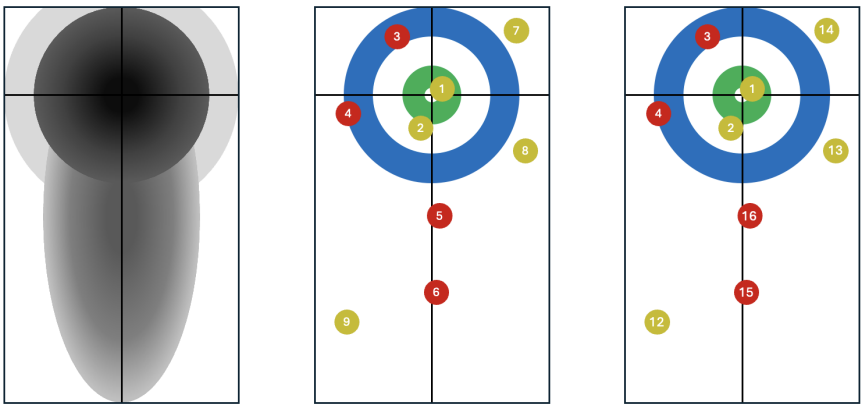

Figure 4.1: Different attempts of ordering rocks.

Number of rocks	Shots remaining	Accuracy NN	Accuracy GNN
$r < 4$	1	95%	90%
$4 \leq r < 8$	1	92%	88%
$r \geq 8$	1	87%	88%
$r < 4$	2	87%	86%
$4 \leq r < 8$	2	80%	80%
$r \geq 8$	2	71%	79%

Table 4.3: Comparison of NN and GNN position classification accuracy.

- **Ordering of Rocks:** Performance improves when rocks are ordered according to their importance to the current game situation. This ordering was determined by the distance to the pin, allowing the network to prioritize more significant rocks during processing.
- **Detailed Features:** Including features from all rocks on the ice significantly enhances neural network performance. This comprehensive input ensures that the network has access to all game-critical information.
- **Indicator Vectors:** Using indicator vectors, which consist only of values 0 and 1, to represent qualitative aspects such as whether a rock is guarded, enhances performance. This restructured data presentation, with more entries used to represent one feature, rather than using a single float input, optimizes how the neural network processes the information.

A further enhancement could involve redefining the order of the rocks in the input vector, as visualized in Figure 4.1. Prioritizing rocks within the house by their distance to the pin and then including guards based on their distance from the center line could improve the neural network's ability to evaluate and respond to game situations more effectively.

Specifically comparing the differences between the NN and the GNN, the findings show similar numbers in overall positions but differ when the positions are categorized into different strategies. Table 4.3 illustrates the accuracy of each bot during model training on the chosen test set, specifically analyzing the accuracy for only the last rock and second-to-last rock in various configurations.

Search Parameters	Rock remaining			
	5	**4**	**3**	**2**
No Search	42%	63%	69%	82%
$f = 1$	55%	61%	71%	88%
$f = 2$	58%	70%	71%	93%
$f = 4$	64%	73%	80%	95%

Table 4.4: Success rate with different search parameters.

The results indicate similar performance levels for both NN and GNN, with NN slightly outperforming GNN in simpler scenarios. However, as the complexity of the positions increases with more rocks, the performance of NNs declines more sharply compared to GNNs. This trend suggests that GNNs may be better suited for handling complex positions, whereas NNs could be more effective in simpler scenarios with fewer rocks.

- **Ordering of Rocks:** GNNs do not require a specific ordering of input features, providing flexibility in dynamic game situations where the importance and interrelations of rocks can change throughout play.
- **Complexity in Data Representation:** Graph neural networks leverage the graph structure to model complex relationships between rocks, allowing for a more nuanced understanding of the game state. However, this complexity necessitates advanced data processing techniques.
- **Lack of Position Data:** While GNNs excel at analyzing relationships between individual rocks, they may not capture the holistic strategic dynamics dependent on the overall positioning of all rocks as effectively as NNs.

Future improvements could involve using GNNs for more complex positions and NNs for simpler ones, or potentially integrating both networks into a hybrid model. This hybrid approach would combine the strengths of both architectures: GNN's adeptness at handling relational data and NN's efficiency in processing positional features. Integrating these architectures could enable the model to capture both detailed interactions among rocks and the overall strategic layout, potentially leading to richer and more precise game analysis.

4.2 Search Algorithm Parameters

This section evaluates the effectiveness and efficiency of the developed search algorithm, combining Monte Carlo Tree Search and minimax, with alpha-beta pruning, used within the curling simulation. The focus is on both the quality of decisions made and their computational performance, under different circumstances and parameters, providing a dual perspective on their utility in game scenarios.

Firstly, this section assesses the quality of strategic decisions made by the search algorithm within the simulation. This comparison aimed to understand how search algorithms help the bot to align with expert strategies, which is indicative of their ability to mimic high-level human strategic thinking.

Table 4.4 presents the success rates with different parameters. The focus on this analysis is on the parameter f, which indicates the maximum number of follow-up calls in later positions. In each scenario the number of analyzed calls c is set to 10 and the maximum width w is set to 5. The column shows how many shots are remaining in the end.

The data in Table 4.4 shows that incorporating a search algorithm generally improves the success rate of strategic calls compared to the baseline (No Search). The success rate increases with the number of

Search Parameters	Rock remaining					
	13	**5**	**4**	**3**	**2**	**1**
$c = 4, w = 4, f = 1$	9.41	3.35	2.72	2.07	1.37	0.47
$c = 4, w = 20, f = 1$	50.91	13.25	10.84	8.53	4.52	1.46
$c = 20, w = 20, f = 1$	157.44	40.16	31.17	25.89	15.74	6.72
$c = 4, w = 4, f = 2$	-	29.67	13.09	7.01	2.70	0.48
$c = 4, w = 4, f = 4$	-	182.54	63.34	18.70	3.24	0.47

Table 4.5: Time used for evaluating $c \times w$ calls, with f follow-up calls in seconds.

follow-up calls (f), indicating that deeper evaluations lead to better decision-making. For example, with 5 rocks remaining, the success rate increases from 55% with $f = 1$ to 64% with $f = 4$. This trend is consistent across different stages of the end, demonstrating the benefit of deeper search depths in enhancing call quality.

To ensure a quick decision making the computation demand of each setup has to be evaluated alongside the quality. Timing analysis focused on measuring the computational time of each parameter setup under various game conditions. The number of calls analyzed in each setup was recorded, alongside the time taken to process these calls. The timing results for the different setups were documented in Table 4.5. The top row indicates the number of rocks remaining in the end.

The results highlight a near-linear increase in computation time with the number of calls (c) and evaluations per call (w). However, the efficiency of Monte Carlo Tree Search allows for some time savings even with higher values of c and w, demonstrating MCTS's ability to balance exploration and exploitation.

In contrast, increasing the depth of follow-up calls (f) has a much greater impact on computation time. This significant increase underscores the computational intensity of deeper evaluations.

The findings suggest that early end evaluations should be conducted with $f = 1$ to manage computation time effectively, ensuring timely decision-making. This approach allows the bot to make quick decisions in the earlier stages of an end while leveraging the strategic depth provided by search algorithms. As the game progresses and fewer rocks remain, increasing the depth of follow-up calls becomes more feasible and beneficial for achieving higher accuracy in strategic calls.

Overall, the combination of Monte Carlo Tree Search and minimax enhances the bot's decision-making quality. By adjusting search parameters based on the stage of the game, the bot can balance thorough analysis and efficient processing, leading to optimal performance in curling simulations. These insights pave the way for further improvements in bot design and strategy development, emphasizing the importance of dynamic parameter adjustments to maximize both accuracy and efficiency.

4.3 Position Classification

This section evaluates the accuracy of the bots' ability to classify positions during gameplay and examines the impact of varying database sizes on this accuracy.

Table 4.6 illustrates the accuracy with various database sizes and different numbers of rocks in the position. The first trend is that a larger database enhances the accuracy. The second trend is that the bot has more difficulty accurately classifying positions with more rocks.

The database was expanded through games played between two bots, with each position being classified before every shot. If $\sigma \geq 1$, indicating significant dissimilarity and uncertainty, the position is added to the database to enhance future classification accuracy. After analyzing 300,000 positions from around 2,500

Number of Rocks	200 positions	1,000 positions	10,000 positions	52,726 positions
$r < 4$	80%	91%	98%	99%
$4 \leq r < 8$	12%	25%	41%	53%
$8 \leq r < 12$	1%	8%	12%	15%
$r \geq 12$	0%	1%	2%	3%

Table 4.6: Comparison of accuracy depending on the number of positions in the database.

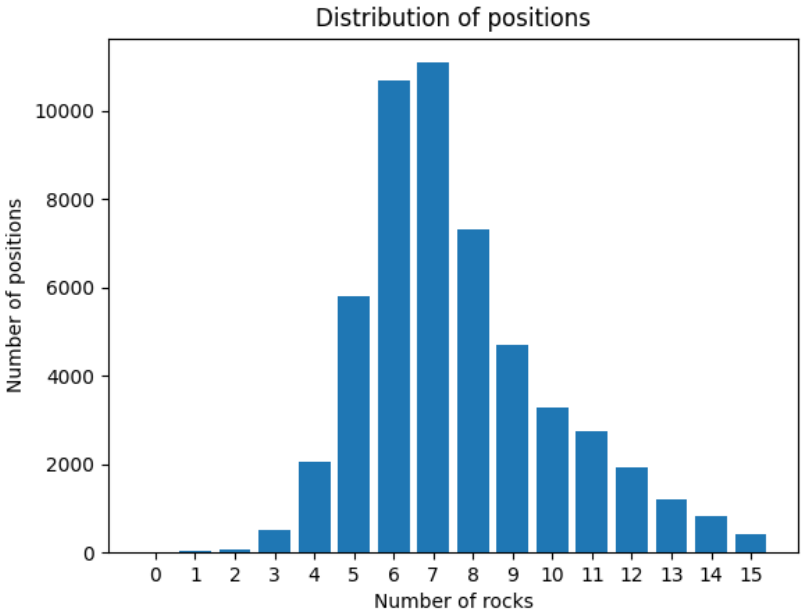

Figure 4.2: Distribution of positions with 52,726 entries in the database.

games, the database expanded to include 52,726 entries. Figure 4.2 illustrates the distribution of these positions, which predominantly feature five to nine rocks. There are fewer positions with less than five rocks, because there are fewer configurations that have not already been encountered. Conversely, positions with more than nine rocks occur less frequently in gameplay, leading to their underrepresentation in the database.

Figure 4.3 illustrates the probability p of adding a new position based on the current database size and the number of rocks in the position, with $1 - p$ representing the accuracy. The likelihood of adding new positions generally decreases as the database expands, particularly for positions with fewer rocks. However, the results indicate that additional positions can still be found, as $1 - p$ is not yet zero for more complex configurations. This approach establishes only a lower bound for the number of possible configurations since the search, especially for positions with more than four rocks, is not exhaustive. While a more extensive search would likely uncover additional positions, the significant computational demands necessitate limiting the scope of this study. Nonetheless, the observed trends and tendencies are sufficiently clear to support the conclusions drawn.

Overall, the position classification results demonstrate a clear pattern: classifying positions with fewer rocks is inherently easier and requires fewer entries in the database. To enhance classification accuracy,

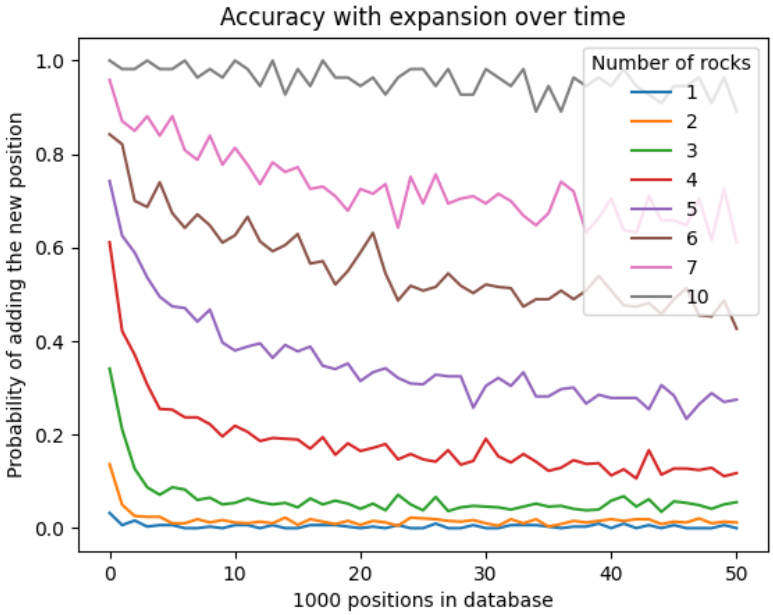

Figure 4.3: Probability of expansion over time.

especially for configurations with many rocks, further database expansion could be pursued. Alternatively, relying on position classification primarily for configurations with fewer than eight rocks could ensure high accuracy, with additional mechanisms needed to handle more complex configurations effectively.

4.4 Bot Comparison

This section presents a comprehensive evaluation of each bot's performance in gameplay scenarios, comparing them not only against each other but also against human players of varying skill levels. This analysis demonstrates each bot's adaptability in dynamic game environments and highlights their strategic strengths and weaknesses. By including games against human players, the evaluation puts the bots' capabilities into perspective, offering insights into how well they can compete against experienced and novice curlers alike.

The results of the tournament matches are succinctly summarized in the win table, depicted in Table 4.7. Each bot participated in a total of 120 games across both settings, featuring search algorithms and without them.

Each bot is guaranteed a minimum of ten wins, reflecting the games played against themselves, and can achieve up to 110 wins due to these self-matches. The Supervised Neural Network Bot and the Graph Neural Network Bot exhibit the strongest overall performance, particularly without the use of search algorithms. Conversely, the Naive and Database bots show limited success, indicating their strategies may not be as robust or adaptable under competitive conditions.

The tournament results, depicted in Table 4.7, and the scoreboard of each match highlight several key trends:

Bot	With Search	Without Search
Supervised NN	94	104
Supervised GNN	92	94
Heuristic	69	71
Self-Learned NN	61	50
Database	31	31
Naive	13	10

Table 4.7: Number of wins made by each bot in the tournament.

Opponent Type	Number of Games	Number of Wins	Win Rate of Bot
Professional Curler	19	0	0%
Experienced Amateur	12	2	17%
Casual Player	9	4	44%
Non-Curler	10	10	100%

Table 4.8: Performance of the top bot against human players of varying expertise.

- The Naive bot primarily achieves victory in games against itself, with minimal success against other bots, indicating its limited strategic variety.
- The Heuristic bot demonstrates better performance when leading but struggles when behind, suggesting that its strategy may not be robust with an offensive setup, where the opponent has more rocks in the house.
- Both the Supervised NN and GNN bots exhibit superior performance, winning the most games.
- Especially the Supervised NN Bot excels in the tournament without search.

Furthermore, the inclusion of search algorithms generally improves decision quality, allowing bots to reconsider their initial choices and potentially select better strategies. However, this flexibility also introduces greater variance in outcomes, as enhanced strategic options can lead to both superior and inferior decisions, depending on the effectiveness of the bot's algorithm and the complexity of the game situation. This variability is also highlighted in the greater number of "upsets", where a worse bot beats a stronger bot due to the increase in randomness.

The Supervised Neural Network Bot, deemed the most successful in previous tests, was further evaluated through matches against human players of diverse skill levels. These matches were conducted to evaluate the bot's strategic decision-making capabilities and to gauge the effectiveness of these decisions in comparison to human players with varying levels of curling expertise. The results of these encounters are summarized in Table 4.10, which details the bot's performance across different categories of player expertise.

The outcomes against human players highlight the Supervised NN Bot's varying success across different skill levels. While the bot was unable to secure wins against professional curlers, indicating a gap in handling complex strategic scenarios, its performance improved against less experienced players. The bot achieved a 17% win rate against experienced amateurs and 44% against casual players, suggesting it competes effectively in mid-level strategic contexts. Notably, its 100% success against non-curlers demonstrates the bot's capability to outperform complete novices.

These results suggest that while the Supervised NN Bot is competent against novice and intermediate players, enhancements are needed for it to challenge expert-level strategies. This points to opportunities for

further development, particularly in advanced strategic modeling and learning to better mimic professional gameplay nuances.

5
Conclusion and Future Work

This thesis explored the development of strategic decision-making in curling using advanced computational methods. The primary focus was on position analysis, leveraging supervised learning with various data collection techniques to evaluate and rank possible calls. Additionally, the utility of search algorithms in refining call rankings was investigated. The insights gained from this study highlight both the potential and the challenges of using these techniques in complex game scenarios.

The central questions posed at the beginning revolved around the effectiveness of advanced computational methods in enhancing strategic decision-making in curling. Specifically, how can position analysis be improved to provide better insights into optimal calls? How do search algorithms perform in various game situations? And what methods can be developed to classify and evaluate game positions effectively? A multi-faceted approach was employed, implementing and comparing several mechanisms through a series of tests.

Different methods of position analysis were tested on an expert-guided test set and in game scenarios. The results from these comparisons highlight the strengths and weaknesses of various approaches in position analysis. Firstly, all bots would benefit significantly from an increase in the dataset size. A larger dataset would enable the development of a more robust network capable of handling complex scenarios.

The learning mechanisms demonstrated promising results, often outperforming the heuristic approach in gameplay scenarios. The self-learned NN bot showed strong performance on the last shot but exhibited weaknesses in earlier stages. This suggests that middle and early end stages have to be evaluated more thoroughly. A more robust evaluation of later shots could also benefit earlier shots, as mistakes in the final shots of an end get transferred down. Another approach is to combine the self-learned training with expert-supervision, by discarding wrong examples from the dataset, gathered by the self-learned method. This way a more robust training set, with less outliers, could increase the accuracy.

The supervised NN bot and the GNN bot achieved the highest win rates and overall accuracy among all tested bots. A combination of these two methods could potentially yield even better results. Specifically, training simpler positions with the NN and more complex positions with more rocks using the GNN could

optimize performance across different scenarios.

The database bot, with only 200 positions in its database, performed poorly, underscoring the necessity for a larger database. For positions with fewer than six rocks, a database size of at least 25,000 well-chosen or iteratively evaluated position-call pairs is recommended to achieve significantly better performance.

An analysis of the best inputs and structures for both NNs and GNNs, using a supervised learning approach, revealed that the accuracy of these models is more dependent on the input vector than on the specific structure of the NN or GNN. Ordering the rocks and incorporating several different attributes for each rock proved crucial for achieving good results.

A promising approach would combine these methods in the following manner:

- Use a database for positions with four rocks or fewer.
- Use a supervised NN for positions with five to eight rocks.
- Use a supervised GNN for positions with more than eight rocks.
- To efficiently increase the dataset size, a self-learned approach is recommended.

By adopting this strategy, the system can leverage the strengths of each method, ensuring robust and accurate position analysis across a wide range of scenarios.

The learning bots were trained to respond to a set of 305 calls. While this setup allows for significant flexibility and the ability to handle a wide range of positions, it is not sufficient when compared to the continuous decision-making capability of human players. Humans can adjust their calls in a fluid and dynamic manner, something the fixed set of 305 calls cannot fully replicate. One way to improve this is to train different networks with varying output sizes. More complex positions often require more sophisticated calls, necessitating a larger list of potential responses. Conversely, simpler positions do not benefit from an extensive list, which could complicate classification and decision-making. By tailoring the size of the output set to the complexity of the position, the system can better mimic human decision-making.

Along with the position analysis search algorithms were observed, regarding their improvement to the call ranking. The combination of the Monte Carlo Tree Search and the minimax algorithm was evaluated, focusing on its accuracy and efficiency in making strategic calls with various parameters, namely c (the number of analyzed calls), w (the maximum number of times a call gets analyzed) and f (the number of follow-up calls).

Increasing the number of follow-up calls (f) enhances call accuracy but significantly increases computation time, making it infeasible to use $f > 1$ in earlier stages of an end. Limiting the depth of follow-up calls while allowing $1 < f < 10$ at all stages, with a maximum depth of 4, could provide a balance between accuracy and efficiency. This would mean the bot has to evaluate positions regarding their win probability in the middle of an end. A naive approach could be to evaluate the position purely on the hypothetical score, but this would miss other significant details. Further evaluations are needed to validate this approach.

The efficiency limitation poses a serious challenge for the broader application of the minimax algorithm. Although a more efficient implementation of the simulation could potentially enhance the thoroughness of minimax's analysis, the required improvement in speed would need to be substantial to have a meaningful impact.

The main focus of the position classification was to improve the database bot, but a robust classification system could also enhance the search algorithm, if every position in the database gets coupled not only with a call ranking but also with a win probability.

Different methods and feature vector representations, along with distance metrics, were analyzed to determine the most effective approach for classifying new positions. The construction of the feature vector is of paramount importance. It must capture features of the entire sheet independently of the rocks' relationships to each other, while also assigning a specific position to each rock to assess various aspects such as how guarded and how frozen a rock is, or whether it is in front of the tee. The order of these rocks within the feature vector is critical and raises several questions. Using a defined order based on the distance from the pin is a solid starting point, but further improvements can be made with more sophisticated methods. For instance, incorporating additional criteria for ordering the rocks could enhance the accuracy of the classification.

A weighted distance metric was found to improve classification accuracy. By reducing the importance of less significant rocks, the weighted metric helps to focus on the most important features, minimizing interference from less relevant rocks. This approach ensures that the classification process emphasizes the critical aspects of each position, leading to more accurate and meaningful comparisons. With the results from the database expansion presented, it is possible to demonstrate a lower bound of 52,726 possible positions using the chosen setup for comparison. A more extensive search would likely reveal more positions, particularly those with 12 or more rocks. Even with the current database, the probability of recognizing such positions is below 3%.

Overall, the position classification system developed in this research provides a robust framework for evaluating and comparing game positions, leveraging the strengths of different feature vector representations and distance metrics to optimize performance. Improvements can be made in the ordering of the rocks and in a further expansion of the database.

5.1 Future Work

Games against human players have established a solid baseline, demonstrating that the bot can be a valuable tool for improving strategic thinking in curling. For beginners, the bot offers an excellent platform to try out different strategies and learn the fundamentals of the game. However, the bot is not yet advanced enough for professionals curlers to reliably test and refine new strategies.

Going forward, there are numerous avenues for improving the bot's performance and capabilities. One of the primary challenges with reinforcement learning is that the bot utilizes different models, making it difficult to identify and rectify mistakes within specific models. Enhancing the bot's ability to recognize and correct errors across its models is essential for its development.

Another crucial aspect is the ordering of rocks as input, which has been identified as a significant factor in the bot's decision-making accuracy. Exploring more sophisticated methods for dynamically ordering rocks based on their importance to the position could lead to better performance.

Additionally, the results show that splitting the bot into different mechanisms tailored to specific aspects of the game would enhance its overall effectiveness. For instance, distinct modules could be developed to handle the early, middle, and late stages of the end, each optimized for the unique challenges and strategies relevant to those phases. One possible approach is to use a database bot for the early stages of the end, where fewer rocks are involved, and machine learning models for later stages, where different

inputs are used depending on the complexity of the positions.

Increasing the size of the dataset with accurate position-call pairs is important for both the database approach and machine learning. One way to achieve a larger dataset is to use a self-learned approach with expert supervision to filter out incorrect calls.

Redefining the bot's training approach using reinforcement learning could also provide significant improvements. By continuously learning and adapting from both simulated and real games, the bot could develop more advanced strategies and improve its performance over time. A challenging aspect of this approach is managing the bot's different models for various stages of the game. Currently, the bot uses distinct models tailored to specific stages of the end, which complicates the training process. Implementing a single, unified model could streamline the training process but may reduce the bot's adaptability to the specific context of each shot. Balancing the need for specialized models with the simplicity of a unified approach will be crucial for optimizing the bot's strategic capabilities.

In summary, while the current bot offers a robust foundation for strategic training and competition, there is substantial potential for further enhancement. By addressing the identified challenges and exploring innovative solutions, future research can continue to advance the capabilities of curling bots, making them even more valuable tools for players at all levels.

Bibliography

[1] Han Y, Zhou Q, Duan F. A game strategy model in the digital curling system based on NFSP. Complex and Intelligent Systems. 2021:1-7.

[2] Masui, F., Otani, H., Yanagi, H., Ptaszynski, M. (2019). Study on Game Information Analysis for Support to Tactics and Strategies in Curling. In: Cabri, J., Pezarat-Correia, P., Vilas-Boas, J. (eds) Sport Science Research and Technology Support. icSPORTS icSPORTS 2016 2017. Communications in Computer and Information Science, vol 975. Springer, Cham.

[3] K Lee, SA Kim, J Choi, SW Lee. Deep Reinforcement Learning in Continuous Action Spaces: a Case Study in the Game of Simulated Curling, PMLR 80:2937-2946, 2018.

[4] World Curling Federation. Rules of Curling 2023. Available from: https://worldcurling.org/wp-content/uploads/2023/07/2023-The-Rules-of-Curling.pdf

[5] The Forecast Factory LLC. True win probability charts. [online]. 2020-2023 [Accessed 14 April 2024]. Available from: https://doubletakeout.com/winprob

[6] Howard R. Curl to win: Expert advice to improve your game. HarperCollins Publishers Ltd; 2009

[7] Zhou ZH. Machine learning. Springer nature; 2021 Aug 20.

[8] Goodfellow I, Bengio Y, Courville A. Deep learning. MIT press; 2016 Nov 10.

[9] Mohri M, Rostamizadeh A, Talwalkar A. Foundations of machine learning. MIT press; 2018 Dec 25.

[10] Nasteski V. An overview of the supervised machine learning methods. Horizons. b. 2017 Dec 1;4(51-62):56.

[11] Hamilton WL. Graph representation learning. Morgan and Claypool Publishers; 2020 Sep 16.

[12] Wu Z, Pan S, Chen F, Long G, Zhang C, Philip SY. A comprehensive survey on graph neural networks. IEEE transactions on neural networks and learning systems. 2020 Mar 24;32(1):4-24.

[13] Balakrishnan R, Ranganathan K. A textbook of graph theory. Springer Science and Business Media; 2012 Sep 20.

[14] Morris C, Ritzert M, Fey M, Hamilton WL, Lenssen JE, Rattan G, Grohe M. Weisfeiler and leman go neural: Higher-order graph neural networks. InProceedings of the AAAI conference on artificial intelligence 2019 Jul 17 (Vol. 33, No. 01, pp. 4602-4609).

[15] Murphy KP. Machine learning: a probabilistic perspective. MIT press; 2012 Sep 7.

[16] Deza E, Deza MM, Deza MM, Deza E. Encyclopedia of distances. Springer Berlin Heidelberg; 2009.

[17] Knuth DE. The art of computer programming. Pearson Education; 1997.

[18] Sutton RS, Barto AG. Reinforcement learning: An introduction. MIT press; 2018 Nov 13.

[19] Shannon CE. XXII. Programming a computer for playing chess. The London, Edinburgh, and Dublin Philosophical Magazine and Journal of Science. 1950 Mar 1;41(314):256-75.

[20] Knuth DE, Moore RW. An analysis of alpha-beta pruning. Artificial intelligence. 1975 Dec 1;6(4):293-326.

[21] Browne CB, Powley E, Whitehouse D, Lucas SM, Cowling PI, Rohlfshagen P, Tavener S, Perez D, Samothrakis S, Colton S. A survey of monte carlo tree search methods. IEEE Transactions on Computational Intelligence and AI in games. 2012 Feb 3;4(1):1-43.

[22] Chaslot G, Bakkes S, Szita I, Spronck P. Monte-carlo tree search: A new framework for game ai. InProceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment 2008 (Vol. 4, No. 1, pp. 216-217).

[23] Maeno N. Dynamics and curl ratio of a curling stone. Sports Engineering. 2014 Mar;17:33-41.