# Idle time driven decision making by Always-On Agents

Thesis submitted in partial fulfillment
of the requirements for the degree of

*(Master of Science in **Computer Science and Engineering** by Research)*

by

Sravya Sri Garapati
201102076
garapati.sravyasri@research.iiit.ac.in

International Institute of Information Technology, Hyderabad
(Deemed to be University)
Hyderabad - 500 032, INDIA
April 2022

International Institute of Information Technology
Hyderabad, India

# CERTIFICATE

It is certified that the work contained in this thesis, titled "Idle time driven decision making by Always-On Agents" by Sravya Sri Garapati, has been carried out under my supervision and is not submitted elsewhere for a degree.

Date

Adviser: Dr. Kamalakar Karlapalem

To Almighty God, Grateful for all he has given me.

# Acknowledgements

I would like to express my deep gratitude towards my thesis guide, Dr. Kamal Karlapalem. His knowledge, discipline, patience, and out-of-the-box thinking are nothing less but an immense inspiration to me. Thank you for being prompt and always open to discussions.

I must thank my mother for supporting me mentally and physically through thick and thin. She helped me overcome challenging times, and this thesis is only possible because of her. I also want to thank my father, sister, and extended household for supporting me throughout my decisions. I should thank Dr. Chandra Sheker Rao and his staff for giving me the best treatment, helping me recover, and for always reassuring me.

I am very grateful to be part of the prestigious institute IIIT Hyderabad. The institute and people here helped me grow personally and academically. I want to thank my friends for spending their valuable time and helping me whenever I was blocked. Thank you, folks, for teaching me to be a sport and uplifting me always come what may.

# Abstract

Always-on agents are those agents which are like daemon programs that are always on and can do tasks as and when they arrive. These agents are idle when there is no task assigned to them. Further, those agents that work on a task also wait for some event or task completion, and hence, are also idle for short durations while executing the tasks. The question arises what should the agent be doing when it is idle.

In this thesis, we conduct an empirical analysis to show improved decision-making capabilities by agents when they exploit their idle time. We analyze scenarios where (i) the agents analyze their past tasks regarding decisions taken and their impact and where (ii) the agents cooperate with other agents to improve their decision-making. Performance improvement can be measured by considering an increase in success rate, avoiding strategies that may not work, quicker decision making, etc. While executing a task, our always-on agent stores pertinent details of the task done in a database, such as decisions taken, paths of execution of the task, goodness of them, etc. The agent uses different strategies to use this stored knowledge. We present and evaluate three strategies that always-on agents can use (i) Frequent Decision Strategy (FDS) - the agent stores the prior executions and their frequency of success and failure, repeats the most frequent successful decision taken during prior executions of the task, (ii) Analyzed Decision Strategy (ADS) - the agent analyses prior executions that were successful or not, stores in the database the goodness of various alternatives and chooses the best alternative and (iii) Online Analysis Decision Strategy (OADS) - the always-on agent while executing its task, during its idle time analyses the possible future situations and prepares the list of best possible decisions that can be taken in future. The FDS and ADS are used when the agent is not doing any task and is off-line. In contrast, OADS generates new mock tasks to consider possible alternative task execution situations to expand its decision-making scenarios while having a task at hand. We conduct our empirical study on always-on agents playing Connect-4 games to check the viability and improved decision-making.

We also present idle time analysis in always-on cooperative agents. We propose a Multi-agent Framework that always-on cooperative agents can use in their idle time to improve decision-making. We formulate scenarios in which the agent has a rationale to cooperate with other agents (i) Exception Handling - the agent cooperates to handle situations in which it does not know what to do, (ii) Confidence Boosting - the agent cooperates to boost confidence in actions it wants to take, (iii) To obtain Different Experiences - the agent cooperates to obtain different experiences. We conduct an empirical study on always-on cooperative agents using the Multi-agent Framework and show improved decision-making.

# Contents

# List of Figures

# List of Tables



-

*Chapter 1*

# Introduction

An agent is a software that lies in an environment, and that is capable of taking autonomous action in order to meet its design objectives [45]. The key traits of an intelligent agent are to be social, reactive, and proactive. Being social, the agent interacts with other agents, being proactive, the agent works towards a goal, being reactive, the agent perceives the environment and reacts to it.

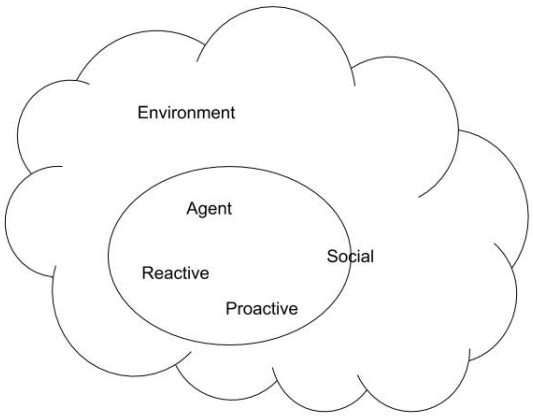


**Figure 1.1** Key traits of an agent

The field of agents is vast, and there are numerous applications of agents in different domains [13]. Some applications of agents are in personal assistants, meeting schedulers, email assistants filtering emails, intelligent user interfaces, retrieving information from the Internet, manufacturing operations, electronic business, online shopping, telecommunication, and decision-making. For example, Maxims [20] is one of the oldest agents that helps users manage their email. Maxims learns to act (forward,

delete, group and filter) on mail messages on behalf of the user. CMRadar [22] is an example of personal assistant agent for calendar management. The agent facilitates end-to-end meeting scheduling process like initiation, confirmation and rescheduling. OASIS [19] is an agent oriented system developed to manage air traffic. It helps the aircraft controller by finding the optimal sequence in which to land aircraft thereby maximizing runway utilization. An agent based approach for mobile e-Health monitoring [4] presents a networked team of intelligent agents that monitors and recommends actions to home-based chronic illnesses patients and medical staff in a mobile setting.

There are also always-on agents are agents that are like daemon programs that are always on and can do tasks as and when they arrive. These agents are idle when there is no task assigned to them. Further, those agents that work on a task, also wait for some event or task completion, and hence, are also idle during execution of the tasks.

As detailed above, always-on agents have idle time, and they could initiate new tasks and execute them during this time. There can be different kinds of analysis happening in the idle time of agents. This idle time in always-on agents posed us to think of how an always-on agent can leverage its idle time. Always-on agents when using their idle time can do much better than other agents. An always-on companion for isolated older adults developed by Candace Sidner and others[34] is an example of an always-on agent that helps the older adults to reduce isolation not just by always being around but also by providing specific activities that connect the user with friends, family, and the local community. This always-on agent can use its idle time to understand and gather information about the adult or look back on the decisions it took in its idle time and improve the quality of interactions, also the companion agent can use its idle time to cooperate with other always-on agents and improve its performance.

## 1.1 Idle time in Humans

As humans, we tend to analyze or contemplate in our idle time. Idle time analysis or contemplation consists of a random or directed exploration of choices to be made and store the best choices for future retrieval and use. There are many studies that support contemplation in humans [12, 26, 17]. We often think of the past and what we could learn from it, and what should be changed in the future when we get idle time. Let us look at a simple example of human contemplation. Say a student Ram has to go from university block A to university block B (Figure 1.2). He chooses path 1 on his first day of college. In his idle time, he explores the university to find that paths 2, 3 also lead to B and that path 2 is the shortest between block A and block B. He remembers it and uses path 2 to reach B from A the next day.

## 1.2 Idle time in Always-on Agents

Like humans, always-on agents can use their idle time productively. In our work, we introduce the concept of idle time analysis/contemplation in always-on agents. There are multiple things an agent can do in its idle time. For example, the agent can look at its past in idle time and do better in its

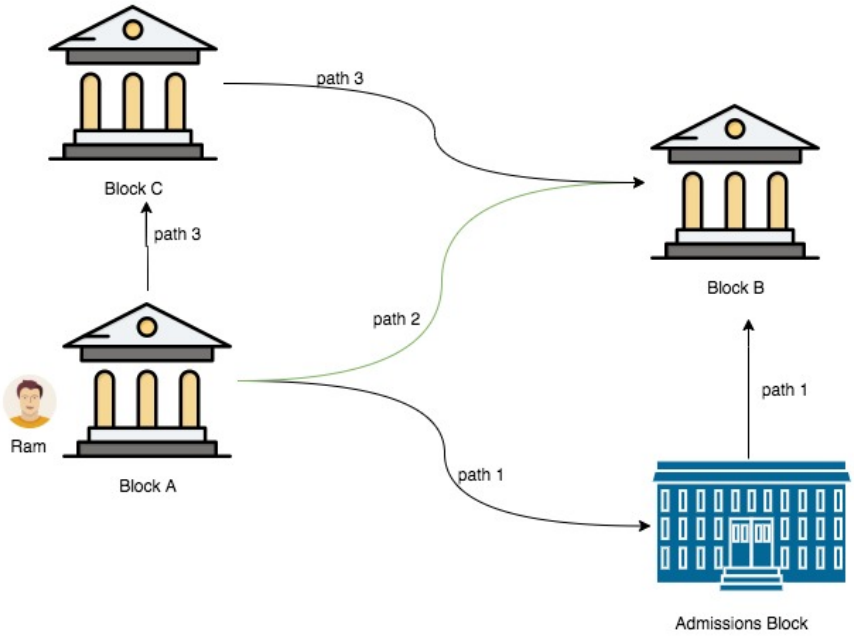


**Figure 1.2** Human Contemplation

assigned tasks. The agent can cooperatively work with peer agents, get their suggestions, store them, and use them. There is not much work to utilize the idle time in always-on agents to the best of our knowledge. To study the idle time in always-on agents, we need to propose methods to utilize idle time on top of the existing methods that agents use in their active time, and when there are multiple agents, we need a framework for agents to cooperate in idle time together. As part of this thesis, we introduce the concept of idle time in agents. We present and evaluate three different strategies that an always-on agent can use in idle time to analyze the tasks it has done (i) Frequent Decision Strategy- FDS, (ii) Analyzed Decision Strategy- ADS and (iii) Online Analysis Decision Strategy- OADS. FDS and ADS are used when the agent is not doing any task. In contrast, OADS generates new mock tasks to consider possible alternative task execution situations during breaks while actively executing a task. We conduct our empirical study on always-on agents playing Connect-4 games to check the viability and to show improvement in decision-making.

We present a Multi-agent Framework for always-on agents to cooperate with other agents in idle time. Using the proposed framework, always-on agents can cooperate by taking inputs from each other and analysing in a group. We evaluate always-on agents playing Connect-4 games, and we show improved decision-making in always-on agents when they analyze alone and in groups during idle time.

### 1.2.1 Contributions and Thesis Organisation

To sum up, following are the contributions,

- We present two types of idle times in agents (i) when an agent is not assigned any task (ii) when an agent is assigned a task but has some intervals of time during which it is idle.
- We present and evaluate Frequent Decision Strategy (FDS), Analyzed Decision Strategy (ADS), and Online Analysis Decision Strategy (OADS) that always-on agents can use in idle time.
- We propose a Multi-agent Framework and its components Subscription Layer, Communication Layer, and Voting Layer for always-on cooperative agents to cooperate and analyze with each other in idle time.
- We evaluate different scenarios in always-on cooperative agents and show improved decision making in always-on agents when they analyze in groups in idle time.
- Our results validate that idle-time driven decision making improves always-on agents performance.

The organization of the thesis is as follows.

Chapter 2 discusses the related work for always-on agents, idle time analysis, and efforts.

Chapter 3 discusses idle time-driven improved decision-making in always-on agents. We discuss and evaluate strategies employed by the agents 1. Frequent Decision Strategy 2. Analyzed Decision Strategy 3. Online Analysis Decision Strategy.

Chapter 4 discusses always-on cooperative agent's idle time analysis for improved decision making and evaluate them.

Chapter 5 discusses application of idle time analysis in agents of different domains.

Finally, Chapter 6 summarizes this thesis's conclusions, contributions and discusses a few directions for future research.

*Chapter 2*

# Related Work

## 2.1 Agents

An agent is an entity that is situated in an environment. It senses and acts on it in pursuit of its agenda and to effect what it senses in the future [9]. The key traits of agents such as social, reactive, and proactive are elaborated in the previous section (Section 1). There are several types of agents, and Nwana [24] attempts to place them into seven categories.

- Collaborative agents - Collaborative agents exhibit autonomy and cooperation with other agents to perform tasks assigned to them.
- Interface agents - Interface agents exhibit autonomy and learn to perform assigned tasks.
- Mobile agents - Mobile agents exhibit the capability to roam and gather information and come back to the home station when required.
- Information agents - Information agents perform the role of managing, manipulating, or collating information from distributed sources.
- Reactive agents - Reactive agents act or respond in a stimulus-response manner to the environment's state they are present in.
- Hybrid agents - Hybrid agents refer to those who combine two or more agent philosophies within a singular agent.
- Smart agents - Smart agents exhibit autonomy, cooperation, and learning to perform tasks assigned to them.

## 2.2 Always-On Agents

There is little mention of the always-on agents in the past literature. Some of the always-on agents are an always-on companion for isolated adults developed by Sidner and others [34] and a wearable

just-in-time information system that continually reminds the wearer of potentially relevant information based on the wearer's current physical and virtual context [27]. However, to the best of our knowledge, always-on agents in game playing are still not explicitly explained.

Computer systems that solve new problems by analogy with previous ones are called Case-Based Reasoning systems [46]. Case-Based Reasoning's basic idea is to remember past experiences and adapt them to problem-solving. CBR stores the past knowledge in memory, and when a new problem arises, it takes inference from the past solutions to solve similar problems.

Reinforcement learning is a process of discovering actions with the goal of maximizing a numerical reward signal [39]. Our contribution focuses on agent using idle time to reuse the skills already developed to improve the performance on top of any existing strategy agent uses in active time.

Lifelong Machine Learning (or Lifelong Learning) is a machine learning paradigm that learns continuously, accumulates the knowledge learned in previous tasks, and uses it to help effective future learning [5]. Continual learning is the constant development of increasingly complex behaviors. It is the process of building more complicated skills on top of those already developed [28]. A continual-learning agent should, therefore, learn incrementally and hierarchically. In our work, the learning does not happen continuously but in the idle time agent gets between tasks or while performing tasks. Our agent tries to reuse the skills already developed and can turn the learning on and off. Our goal is to show that the agent improves its decision making capabilities by utilizing idle time. The focus of our work is to determine strategies for task eventually through idle-time analysis. Thus, it is found learning. In other cases, what is learned may not always be useful.

Deepmind's AlphaGo Zero [35] achieved superhuman performance by using deep reinforcement learning [30]. It uses a trained neural network to predict its moves and the winner of AlphaGo's games. This neural network improves the strength of gameplay and better move selection in the next iterations. AlphaGo Zero is an agent which learns by playing several games with itself before starting to play with others. Our work presents using idle time after deployment and also has aspects of opponent modeled. Unlike neural networks [16] our work allows us to state what we have learned, which is essential for many problems. AlphaGo kind of systems can also use idle-time analysis for their applications.

Different AI search algorithms [29] explore future possibilities and find the choices taken in the present depending on the type of algorithm. They do not explore the past experiences, and they also do not exploit the idle time of agents.

GINA, an Othello playing agent using experience-based learning, is proposed and evaluated by De Jong and Schultz [14]. Katz [15] explains the experience-based learning techniques applied to Go-Moku. They do not indicate about always-on agents and when the learning happens. In this work, we present when the learning happens. We use the experience based technique proposed by De Jong and Schultz [14] for the evaluation of analyzed decision strategy in Connect-4 (Section 3.5.2) .

None of the previous works explicitly discusses always-on agents and the idle time exploitation in them.

## 2.3 Multi-Agent Systems

A multi-agent environment is one in which there is more than one agent, where they interact with one another and model each other's goals and actions [38]. There may be direct or indirect communication between agents in a general multi-agent system (discussed in Section 2.4.1). There are different taxonomies present for multi-agent system applications. For example, Dudek and others [7] presented a detailed taxonomy of multi-agent systems along different dimensions like team size, range, communication topology and throughput, team composition, and reconfigurability, and the processing ability of individual agents.

## 2.4 Cooperative Agent Learning

Cooperative Agent Learning is where we apply the concept of learning to cooperative agents. Mataric [21] describes how agents can be social and improve themselves collectively. He proposes ways in which agent's learning can be improved by being social with other agents. One such is, using observation, the agent observes their peers and imitates their behaviors, which improves the overall team's overall behavior. Panait and Luke [25] defines below two categories of cooperative multi-agent learning

- Team Learning- A single learner works iteratively to improve the team behavior [25].
- Concurrent Learning- Each agent works on improving its own behavior thereby improving overall behavior [25].

Our solution falls into the concurrent learning space where individual agent improves their own behaviour thereby improving overall behaviour. So, we looked at related work surveying concurrent learning.

### 2.4.1 Direct and Indirect Communication in Cooperative Multi-Agent Systems

Ming Tan [41] shows that cooperation, when done intelligently, each agent can benefit from other agents. They present cooperation in three ways. First, agents can communicate instantaneous information. Second, agents can communicate episodes. Third, agents can communicate learned decision policies. The work concludes that cooperative reinforcement-learning agents can learn faster and converge sooner than independent agents via sharing learned policies. Essentially Tan [41], and Berenji [1] suggest that in a cooperative learning environment, an agent must use communication to improve performance.

The above mentioned are explicit communication between agents. There are indirect communication methods as well in cooperative multi-agent learning. Indirect communication involves communicating implicitly, for example, by changing the environment. This is inspired from social insects use of pheromones to mark trails etc. Phe-Q: A pheromone-based q-learning [23] proposes an algorithm that is inspired by the ants depositing pheromones in the environment to communicate.

Our work presents always-on cooperative agents that use direct communication (Section 4.2.2.4). Our solution suggests that the agents share the learned policies in their idle time.

### 2.4.2 Commitment and Decommitment in Cooperative Multi-Agent Systems

Agents that decide to communicate may decide whether to cooperate or not. Even if they decide to cooperate for some time, they might decide to opt-out [38]. Agreeing to a commitment helps agents trust each other and allows smooth cooperation. Castelfranchi [2] proposes that commitment is a crucial notion both to support cooperative work. They propose three types of commitment.

- Internal commitment - Agent commits to itself
- Social commitment - Agent commits to other peer agents
- Collective or Group commitment - Agent commits to fill a role in a group

Our work discussed in Section 4.2.2.3 allows the agent to commit or de-commit using a subscription platform. The subscription layer keeps track of agents that are subscribing and unsubscribing, making the cooperation between them smoother, explicit and having a audit trail on commitments and decommitments.

### 2.4.3 Agents Learning from Other Agents

Reciprocity is a foundational principle for promoting cooperative behavior among self-interested agents. Sandip Sen [32] introduces a probabilistic reciprocity mechanism to show how to incorporate cooperative behavior among a group of self-interested agents. The resultant system exhibits close to optimal throughput with a fair distribution of the workload among the participating agents. Mahendra Sekaran and Sandip Sen [31], [33] propose and investigate the concept that agents can automatically understand and find if others agents are cooperating or non-cooperating and learn from them. They identify that a stochastic decision-making scheme helps agents develop reciprocity among themselves and is suitable for agents in multiple applications. They prove that non-cooperating agents have poor performance in the long run than the cooperating ones.

Bayesian learning method to update and understand models of other agents is proposed by Chalkiadakis and Boutilier [3]. Using these models agents can estimate the behaviour of other agents. This helps them cooperate with other agents in a better fashion. Milind Tambe [40] discusses the benefits of understanding peer agents briefly. They also discuss recursive agent tracking and agent-group tracking. DaSilva and others [6] deal with portion of problems for inter agent teaching. They propose two methods in which agents can learn from one another learner-driven or teacher-driven. Learner driven is where learner is responsible for initiating the interaction between agents and Teacher-driven is where the teacher initiates the interaction between agents.

None of the aforementioned work presents agents learning from others in idle time. In our work, we do not model peer agents and follow the best one but rather get the suggestions from all the peer cooperative agents and weigh them to choose the best decision in a given situation.

*Chapter 3*

# Idle time analysis of Always-On Agents

## 3.1 Introduction

Always-on agents are those agents which are like daemon programs that are always "on" and can do tasks as and when they arrive. These agents are idle when there is no task assigned to them. Further, those agents that work on a task, also wait for some event or task completion, and hence, are also idle for a short duration during execution of the tasks. The question arises what should the agent be doing when it is idle.

In this chapter,

1. We present two types of idle times in agents (i) when an agent is not assigned any task (ii) when an agent is assigned a task but has some intervals of time during which it is idle.

2. We propose Frequent Decision Strategy (FDS), Analyzed Decision Strategy (ADS), and Online Analysis Decision Strategy (OADS) that always-on agents can employ in idle time.

3. We evaluate the strategies for a Connect-4 playing always-on agent to analyze the tasks it has done.

## 3.2 Preliminaries

### 3.2.1 Connect-4 Game

We use Connect-4 for our evaluations in subsequent sections. Connect-4, also called Plot Four, is a two-player game played on a board with seven columns by six rows (Figure 3.1). Player Red (1st) and Player Blue (2nd) place one piece per turn in one available column. Each piece falls into the lowest free position of the column under the force of gravity. The game's objective is the first player to create lines of its four pieces, either vertically, horizontally, or diagonally wins. Figure 3.2 shows example board positions where Red and Blue win, respectively.

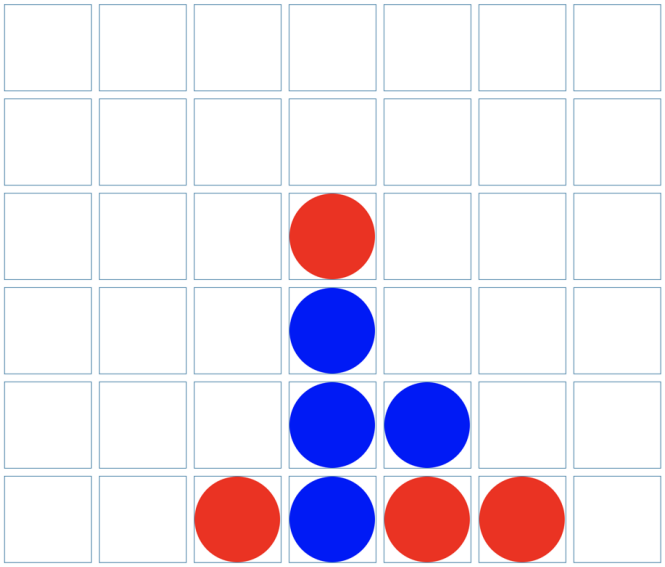

**Figure 3.1** Board in a Connect-4 game

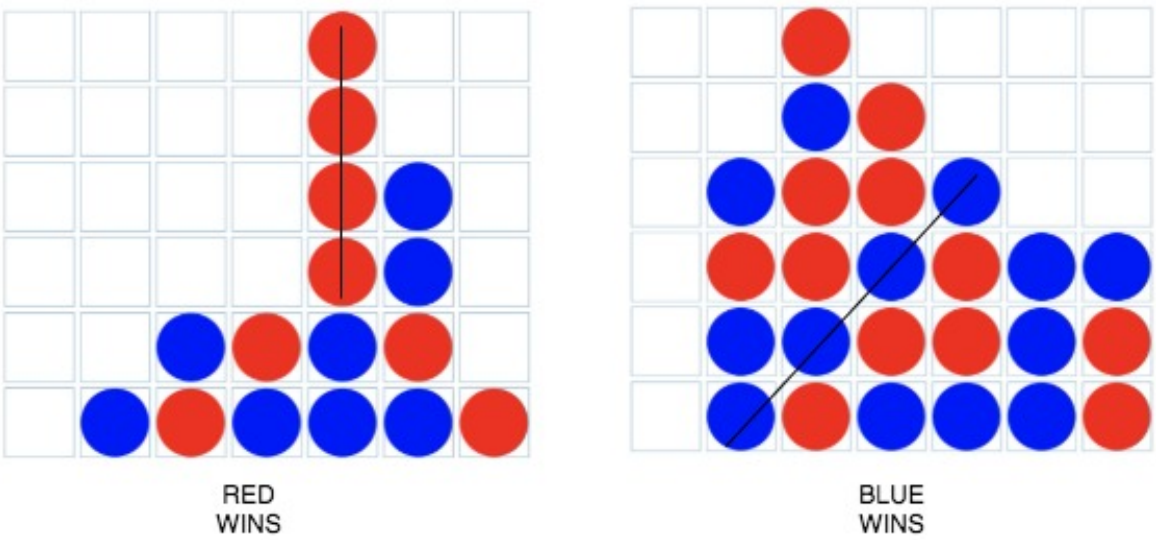


**Figure 3.2** Example winning positions in Connect-4 Game

**Connect-4 Board Representation:** To represent a board position in Connect-4, we have to represent its contents first. We chose character "1" to represent pieces of Player 1 (Red), character "2" to represent pieces of Player 2 (Blue), and "." to represent an open position. Each row in the board is represented by a concatenation of its contents from left to right. A board position is a string concatenation of each row from top to bottom. For example, a Connect-4 board in Figure 3.1 is represented as ".....1.....2.....22....1211." where 1 represents the Red player, and 2 represents the Blue player.

## 3.3 Idle time of Always-On Agents

Suppose, let A be an analyzing always-on Connect-4 agent who plays against another agent B, the goal is to win the game (first player to achieve a line of four connected pieces wins the game Figure 3.2), and the task is to put one of the pieces on the board when it is its turn. When both A and B are active, they play against each other. In idle time, A analyzes the games it played, gains some knowledge and stores in the analysis database or explores on future scenarios and stores different situations and possibilities. The agent becomes self-learning through empirical experiences in course of time.

We define two types of idle times of agents: (i) when the agent is not assigned any task, (ii) when the agent is assigned a task but is idle during short intervals while executing a task. For example, a Connect-4 agent can be treated as idle when it is not playing any game or playing a game but waiting for the opponent's response. The analysis is done in these idle times.

When an always-on agent finishes a task or game and is idle (Figure 3.3), it can use

- Frequent Decision Strategy (FDS)
- Analyzed Decision Strategy (ADS)

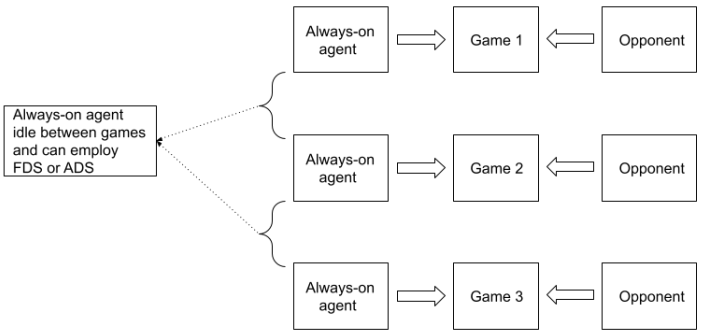


**Figure 3.3** Always-on agent employing ADS/FDS in two-player games

An always-on agent might be idle when it finishes a sub task and when it is in the middle of a task and is found idle for small amounts of time (Figure 3.4). An always-on agent who is in the middle of a task and is idle can employ

- Online Analysis Decision Strategy (OADS)

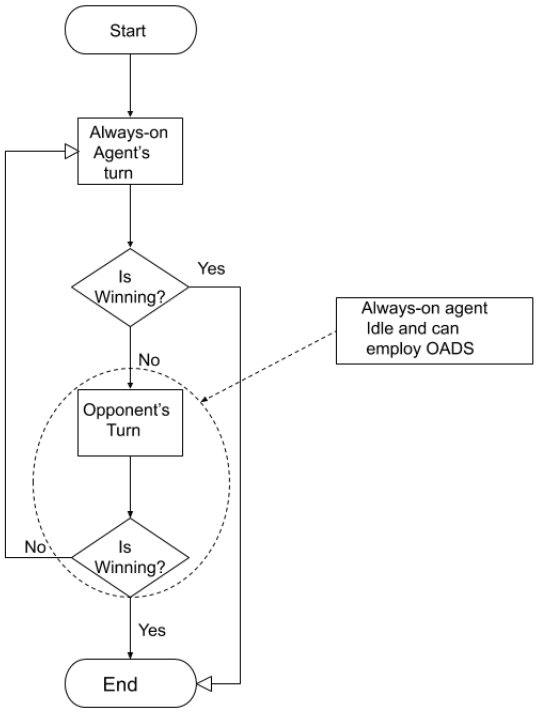


**Figure 3.4** Always-on agent employing OADS in a two-player game

### 3.3.1 Analysis Database

Analysis Database stores the information an always-on agent chooses to keep after analyzing in idle time. It can be different for different domains. In a Connect-4 playing agent the analysis database consists of a table with board position (state), goodness, number of wins and losses associated with the board position, and children of the board position (children are the board positions that occur immediate next to a boardposition). The agent, in idle time, analyzes the games it played and stores all or some of the above information in the analysis database.

Goodness for a board position or state is an estimate measured by different factors. For Connect-4, we use number of 2, 3 and 4 connected segments that player and opponent has at a board position to calculate the goodness. We take weighted sum of 2, 3 and 4 connected segments as shown in Equation 3.1 where $n_2$ is the number of 2 connected segments that player has and $n_3$ is the number of 3 connected segments and $n_4$ is the number of 4 connected segments to calculate the score of the player in that board position. Note that players win with at least one four connected segments. The score is calculated for both players, and the board's goodness is measured by subtracting the player's score from the opponent's score in that board position (Equation 3.2). For example, in the board position given in Figure 3.5, red player has two $n_2$ and one $n_3$ whereas blue player has one $n_2$ and one $n_3$. So, the score of the red player is 6, the score of the blue player is 5, and the goodness of board position for the red player is 6-5=1.

$$\text{Score of player} = n_2 * 1 + n_3 * 4 + n_4 * 40 \tag{3.1}$$

$$\text{Goodness of board position} = Score_{player} - Score_{opponent} \tag{3.2}$$

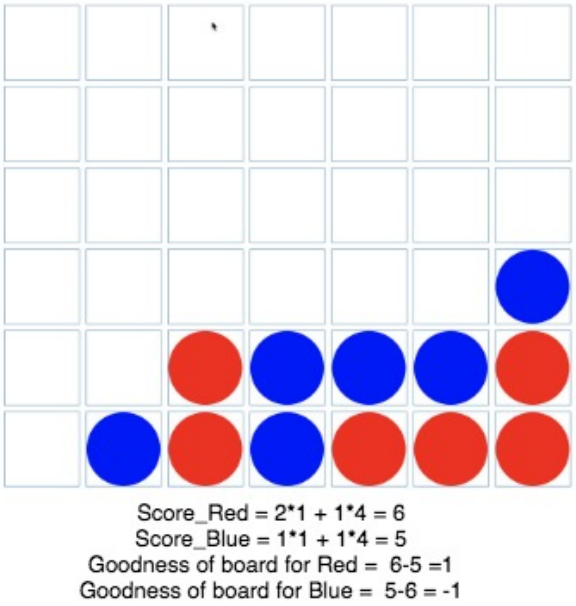


**Figure 3.5** Goodness of a board position in Connect-4 game

Wins and Losses are the numbers of wins and losses the board position contributes. Children is an array of strings storing the children of that board position. Our initial experiments have shown us that the number of reasonable board positions encountered would be manageable.

Table 3.1 shows a sample table in the analysis database of the always-on Connect-4 agent using idle time analysis where the board position, goodness, wins, losses, and children of that board position are the columns and $bp_1$, $bp_2$, and $bp_3$ are sample board positions.

## 3.4 Strategies for Idle time Analysis

We define and evaluate three strategies *Frequent Decision Strategy (FDS)*, *Analyzed Decision Strategy (ADS)*, *Online Analysis Decision Strategy (OADS)*, which an agent can use in its idle time.

| Board Position | Goodness | Wins | Losses | Children |
|---|---|---|---|---|
| ..... ...11..2221112 | 30 | 5 | 3 | [$bp_1$,$bp_2$] |
| ..... ...11..222111. | 12 | 7 | 2 | [$bp_3$,$bp_1$] |
| ..... ...11..122111. | 12 | 17 | 3 | [$bp_1$] |

**Table 3.1** Sample Nodes in Analysis Database for Connect-4

### 3.4.1 Frequent Decision Strategy

In *Frequent Decision Strategy*, the agent repeats the most frequent successful decision taken during prior occurrences of the given state. During the active time, the agent does the tasks assigned to achieve its goals. These tasks and their outcomes are logged and in idle time the agent goes through these logs and the states, frequencies of successes and failures corresponding to these states are stored in the analysis database. During the actual execution of the task, if the current state appears in the database, the decision to go to the frequent successful child state is selected. In case of a Connect-4 playing agent, the task is to play the game, state is the board position, the method is to put the piece on the board, and the goal is to win the game. The games played and results are stored in a log file. In idle time the Connect-4 agent goes through these logs of games and stores the states (board positions) and the frequency of wins and losses corresponding to these states. Table 3.2 shows a sample FDS analysis database. During the game, if the encountered board position appears in the analysis database, the agent chooses the move by which it can go to its most frequent winning child state. *Idle time of agent using FDS* (Algorithm 1) gives the algorithm for idle time of agent using FDS. *Active time of agent using FDS* (Algorithm 2) gives the algorithm for active time of agent using FDS. Figure 3.6 shows an example state board position (bp1) which has children board position 2 (bp2), board position 3 (bp3) and board position 4 (bp4). If using FDS in idle time at state bp1 would make a move to go to the most frequent winning child, in this case, bp2.

| Board Position | Wins | Losses | Children |
|---|---|---|---|
| ..... ...11..2221112 | 5 | 3 | [$bp_1$,$bp_2$] |
| ..... ...11..222111. | 7 | 2 | [$bp_3$,$bp_1$] |

**Table 3.2** Sample Nodes in Analysis Database for Connect-4 in FDS

**Algorithm 1** Idle time of agent using FDS

```
1: AD = Analysis Database
2: for game in gamesplayedbyagent do
3:     for board in game do
4:         if board in AD then
5:             AD.update(board,board.children,game.result)
6:         else
7:             AD.insert(board,board.children,game.result)
8:         end if
9:     end for
10: end for
```

**Algorithm 2** Active time of agent using FDS

```
1: AD = Analysis Database
2: if Agent's turn to decide then
3:     S = []
4:     S = AD.findChildrenWith
5: WinFrequency(boardposition)
6:     if S is not None then
7:         return move to reach the most frequent winning child state in S
8:     else
9:         return agent's choice of move
10:    end if
11: end if
```

### 3.4.2 Analyzed Decision Strategy

Frequent Decision Strategy is good, but it might not always lead to the best results. For example, in a game-like scenario where there is an opponent, it might suddenly change its complete strategy, in which case choosing the most frequent past decision might not work. In *Analyzed Decision Strategy* (ADS), the agent analyzes the prior executions using its choice of analysis technique depending on the domain and stores the goodness of various alternatives in the database. For example, in Connect-4 game, the analysis technique can be using the experience-based learning method where the agent goes through all the games played and assigns value to each board position it encounters (elaborated in Section 3.5.2). The agent does the tasks assigned to achieve its goals in active time. These tasks and their results are stored in a log file. In idle time, the agent analyzes the states, assigns goodness to each state, and stores it in the analysis database. If the current state is present in the database in active time, the decision to go to the best child state is chosen.

In the case of Connect-4, the games played and their results are stored in a log file. In its idle time, the agent goes through the games, analyzes, assigns each board position a goodness estimate using its choice of technique, and stores them in a database. An example ADS analysis database is shown in

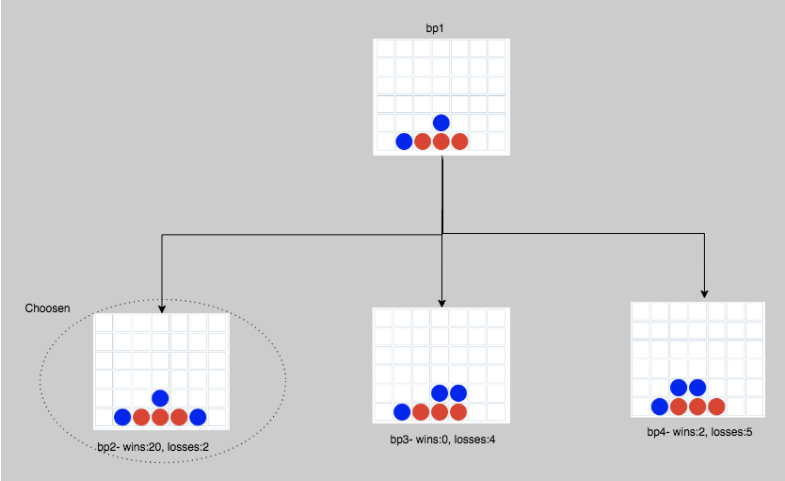


**Figure 3.6** Frequent Decision Strategy

Table 3.1. If the same board position appears in the analysis database during the game, it chooses the best-analyzed child corresponding to that board position.

*Idle time of agent using ADS* (Algorithm 3) gives the complete algorithm for how analysis is done in idle time and *Active time of agent using ADS* (Algorithm 4) gives algorithm for how analyzed knowledge is used in active time of agent. Figure 3.7 shows an example state board position (bp1) which has children board position 2 (bp2), board position 3 (bp3) and board position 4 (bp4). If using ADS in idle time at state bp1 would make a move to go to the best child with best goodness, in this case, bp2.

**Algorithm 3** Idle time of agent using ADS

```
AD = Analysis Database
for game in gamesplayedbyagent do
    technique = Agents choice of technique to analyze
    analyzed = set of states, goodness assigned to each state, children, wins and losses
analyzed = technique.analyze(game)
    for entry in analyzed do
        if entry not in AD then
            AD.insert(entry)
        else
            AD.update(entry)
        end if
    end for
end for
```

### 3.4.3 Online Analysis Decision Strategy

*Online Analysis Decision Strategy* (OADS) is when the always-on agent, while executing a task, during its idle time, analyzes the immediate possible future situations and be prepared with a list of best

**Algorithm 4** Active time of agent using ADS

```
AD = Analysis Database
technique = Agents choice of technique to analyze
if Agent's turn to decide then
    S = []
    S = AD.findChildren(boardposition)
    selectedChildByTechnique= technique.findBestChild(S)
    if selectedChildByTechnique is not None then
        return move to reach the selectedChildByTechnique
    else
        return agent's choice of move
    end if
end if
```

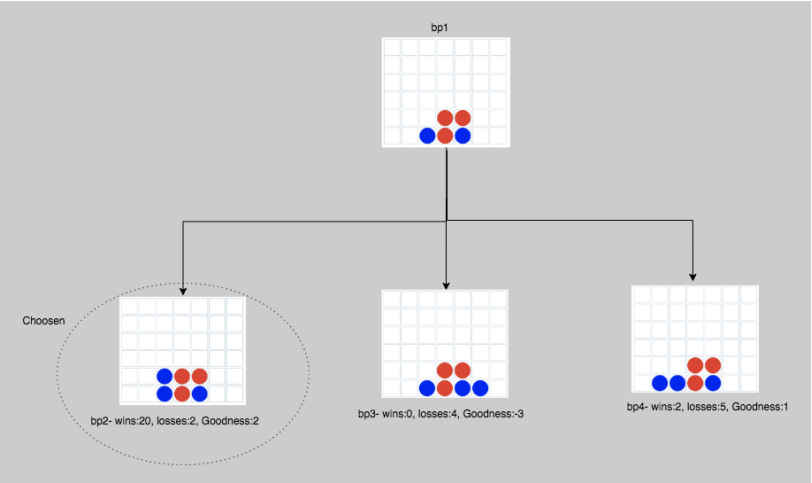


**Figure 3.7** Analyzed Decision Strategy

possible responses for these. The analysis here happens in idle times available while executing the tasks and between tasks when an agent is idle. If the future state is one of the expected ones in active time, the agent has the response ready, thereby making quick decisions.

In the case of the Connect-4 agent, the idle time of the always-on agent while its opponent is thinking is exploited to analyze possible future situations. When it is the opponents turn to move, instead of being idle, the agent analyzes by guessing possible opponent's next moves, ranks them in order of their goodness using its strategies, and stores them in a list. For each choice in the stored list, the agent prepares its analyzed set. So once the opponent responds, if the opponent's move is present in the analyzed set, the agent has a response ready, thereby reducing the response time and making quicker decisions.

For example, in the middle of a Connect-4 game, our agent has put a piece in a column on the board. While the opponent is thinking, the agent creates an anticipatedset which is a set of potential columns that the opponent can put its piece. It creates an analyzedset with its response to each move in the

anticipatedset. If the opponent responds by choosing one of the columns in the analyzedset, our agent has a response ready for it, thereby decreasing the response time. The decrease in response time helps agent react faster and *Idle time of agent using OADS* (Algorithm 5) gives the complete algorithm for how online analysis is done in the idle times when an agent is playing a game. *Active time of agent using OADS* (Algorithm 6) gives the complete algorithm how online analysis is used. Note that in OADS, when the agent is not playing any game, it can use FDS or OADS in idle time.

**Algorithm 5** Idle time of agent using OADS

```
1: boardposition = Current board position
2: anticipatedmoveset =
   getPotentialOpponentMoves(boardposition)
3: for move in anticipatedmoveset do
4:     newboardposition= boardposition.update(move)
5:     newmove= agent(newboardposition)
6:     analyzedset.append(newboardposition,newmove)
7:     if opponent responds then
8:         break
9:     end if
10: end for
```

**Algorithm 6** Active time of agent using OADS

```
1: opponentresponse= opponent's response
2: if opponentresponse in analyzedset then
3:     return move=analyzedset[opponentresponse]
4: else
5:     boardposition= current position of the board
6:     return agent's-choice(boardposition)
7: end if
```

### 3.4.4 Uncertain Idle Time

In uncertain idle time, i.e., when an always-on agent gets interrupted while analyzing, it stores its state, acts submissive by stopping the analysis, and attends to the task at hand. When the agent gets idle again, it resumes the analysis from the state it has left.

## 3.5 Experiments and Results

For evaluation of the idle time analysis strategies, we use traditional minimax Connect-4 engines. Our always-on agent uses the Connect-4 engine's choice of next move if it is not possible to use the analysis database for that board position. The agent stores the games as consecutive board positions and

end outcomes. We do not store the opening board positions in the database to avoid trivial repetition in the games. Note that while FDS and ADS are evaluated by the number of games won, OADS is evaluated by decreasing average response time. All the experiments are repeated fifty times that is about 25,000 games for each strategy against an opponent that does not use any idle time.

### 3.5.1 Evaluation of Frequent Decision Strategy

For evaluation of *Frequent Decision Strategy*, our always-on Connect-4 agent uses a traditional minimax game engine with depth-2 to play against the opponent with depth-4. We consider a case where the agent gets idle time after every 100 games. As by default opponent engine has greater depth than the agent, there is significant competition. Each row in Table 3.3 has an analysis database with all the previous row games appended except in the case of the first one for which the analysis database is empty. The win/loss data distribution metrics like mean, standard deviation and ranges across runs are tabulated in Table 3.3. As the number of games played by the agent increases, the win/loss ratio of our Connect-4 agent using Frequent Decision Strategy increases (Table 3.3). Figure 3.8 illustrates the win/loss ratio of depth-2 Connect-4 agent using Frequent Decision Strategy in idle time (orange line) versus Connect-4 agent not using idle time (green line). The results are shown against an opponent Connect-4 agent of depth-4 without idle time analysis.

| No of games | Mean Win/Loss Ratio ($\mu$) | Standard Deviation ($\sigma$) | Min | Max |
|---|---|---|---|---|
| 1-100 | 0.46 | 0.10 | 0.37 | 0.63 |
| 100-200 | 0.55 | 0.07 | 0.44 | 0.72 |
| 200-300 | 0.62 | 0.11 | 0.45 | 0.77 |
| 300-400 | 0.68 | 0.10 | 0.51 | 0.80 |
| 400-500 | 0.71 | 0.09 | 0.55 | 0.93 |

**Table 3.3** Results of Frequent Decision Strategy

### 3.5.2 Evaluation of Analyzed Decision Strategy

For evaluation of *Analyzed Decision Strategy*, our always-on Connect-4 agent uses a traditional minimax game engine with depth-2 to play against the opponent with depth-4. In idle time, our agent uses the experience-based learning [14] technique for analyzing and assigning goodness to each board position. As in De Jong and Schultz's [14], the agent stores all recorded moves in an analysis database in its idle time. Once the analysis database is formed with all the recorded games, our agent starts with each leaf node and, using the parent pointers, propagates values to its parents in a traditional min-max fashion. This process assigns eventual outcomes to each node capturing the strength of it from experience. An example for the same is shown in Figure 3.9, each board encountered is given a value based on the games in the experience.

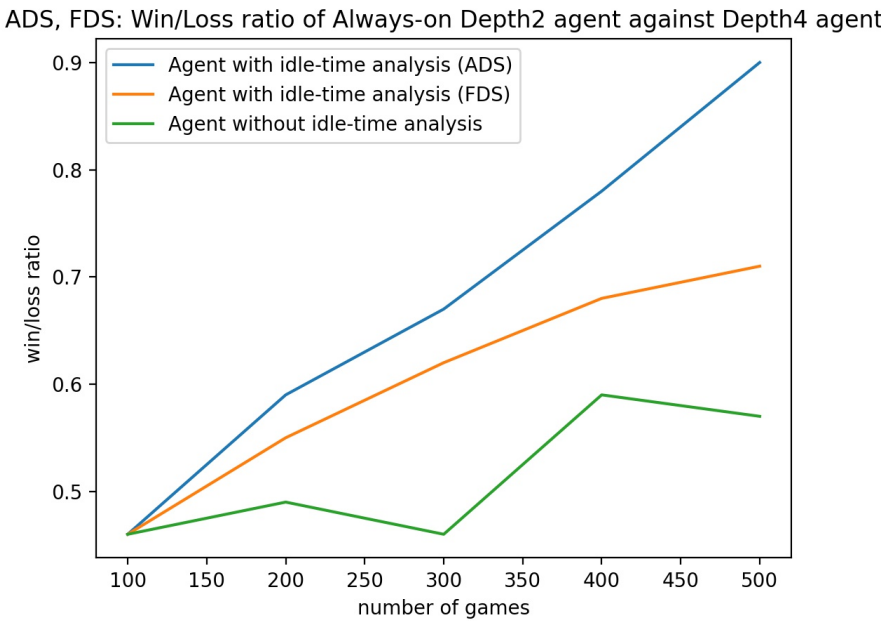


**Figure 3.8** Frequent Decision Strategy (FDS), Analyzed Decision Strategy (ADS) Result Comparison

During the active time, when our agent has to make a decision, it checks the analysis database for its board position. If the node exists, it chooses the child of the node with a positive win frequency and the highest eventual outcome. Suppose no child with positive win frequency exists; our agent checks if all the possible children are explored. If yes, it selects the one with the highest eventual outcome among them; if all the possible children are not explored, our agent uses its game engine to select a move. As the number of games being analyzed increases, the win/loss ratio of our Connect-4 agent using Analyzed Decision Strategy increases (Table 3.4). Each row in Table 3.4 has an analysis database with all the previous rows games appended except in the case of the first one for which the analysis database is empty, win/loss data distribution metrics like mean, standard deviation and ranges across runs are tabulated.

Figure 3.8 illustrates the win/loss ratio of depth-2 Connect-4 agent using Analyzed Decision Strategy in idle time (blue line) versus Connect-4 agent not using idle time (green line). The results are shown against an opponent Connect-4 agent of depth-4 without idle time analysis.

| No of games | Mean Win/Loss Ratio ($\mu$) | Standard Deviation ($\sigma$) | Min | Max |
|---|---|---|---|---|
| 1-100 | 0.46 | 0.10 | 0.37 | 0.63 |
| 100-200 | 0.59 | 0.12 | 0.39 | 0.95 |
| 200-300 | 0.67 | 0.18 | 0.40 | 1.06 |
| 300-400 | 0.78 | 0.17 | 0.47 | 1.09 |
| 400-500 | 0.9 | 0.2 | 0.5 | 1.37 |

**Table 3.4** Results of Analyzed Decision Strategy

### 3.5.3 Evaluation of Online Analysis Decision Strategy

For evaluation of the *Online Analysis Decision Strategy*, we show the quick decision-making of our Connect-4 agent. Our always-on Connect-4 agent uses a traditional minimax game engine with depth-2. While the opponent thinks the agent comes up with as many as possible opponents' decisions and prepares its responses to them. The first column of Table 3.5 represents the time given to the opponent per move in milliseconds. The second column in Table 3.5 shows the percentage decrease in average response time(ART) using Online Analysis Decision Strategy. As the opponent move time increases, the decrease in the average response time of our agent with using the OADS increases (Table 3.5). The Win/Loss ratio of the agent using OADS will be the same as Table 3.3 if it uses FDS and Table 3.4 if it uses ADS when not playing any game. The decrease in average response time helps agent react faster.

| Opponent Move time (millisec) | Decrease in Average Response Time for agent using OADS |
|---|---|
| 30 | 14% |
| 40 | 35% |
| 50 | 56% |
| 60 | 64% |
| 70 | 65% |
| 120 | 69% |

**Table 3.5** Results of Online Analysis Decision Strategy

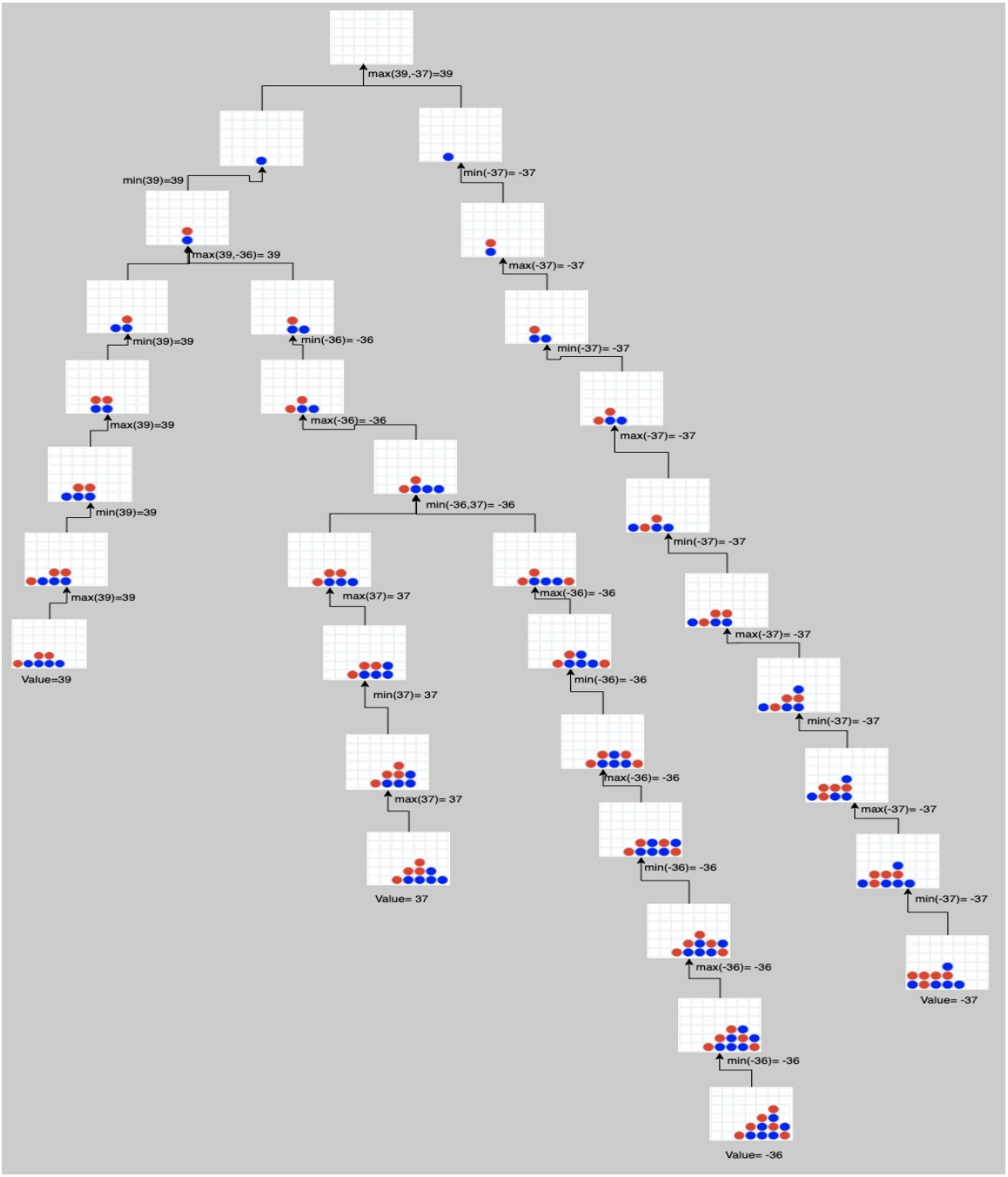


**Figure 3.9** Value propagation from bottom to top in ADS

*Chapter 4*

# Idle time analysis by multiple Always-On Cooperative Agents

## 4.1 Introduction

Always-on agents are like daemon programs that are always on and can do tasks as and when they arrive. Further, those agents that work on a task also wait for some event or task completion and hence, are also idle for a short duration between the execution of the tasks. The idle time of always-on agent can be used for analyzing its decisions with itself [10] and can also be used to cooperate, help others as well as get help from others. Always-on cooperative agents are those agents that are always-on and use idle time to cooperate and help each other (Figure 4.1).

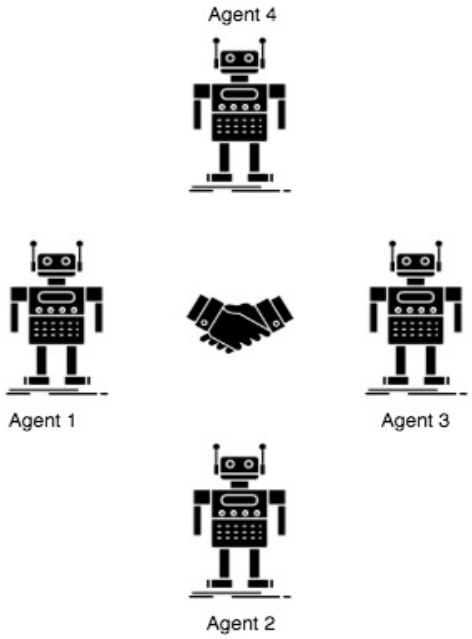


**Figure 4.1** Cooperative Agents

In the previous chapter, we discussed always-on agents using idle time to improve their performance. This chapter discusses always-on agents cooperating in idle time to contemplate individually and in groups to improve their decision-making. The phenomenon of cooperation is very commonly found in humans. They are known to exchange information and gain knowledge from peers or teachers. It is common in non-humans as well [8]. Here we introduce cooperation in the idle time of always-on agents. Let agent A and agent B be two always-on agents playing Connect-4, in active time, both agents are busy and play their respective games. In idle time agents analyze their games and can help each other in situations they face. We propose a framework and methods which enable the cooperation in idle time of always-on cooperative agents. Figure 4.2 illustrates a setting where two game-playing always-on cooperative agents A and B use idle time.

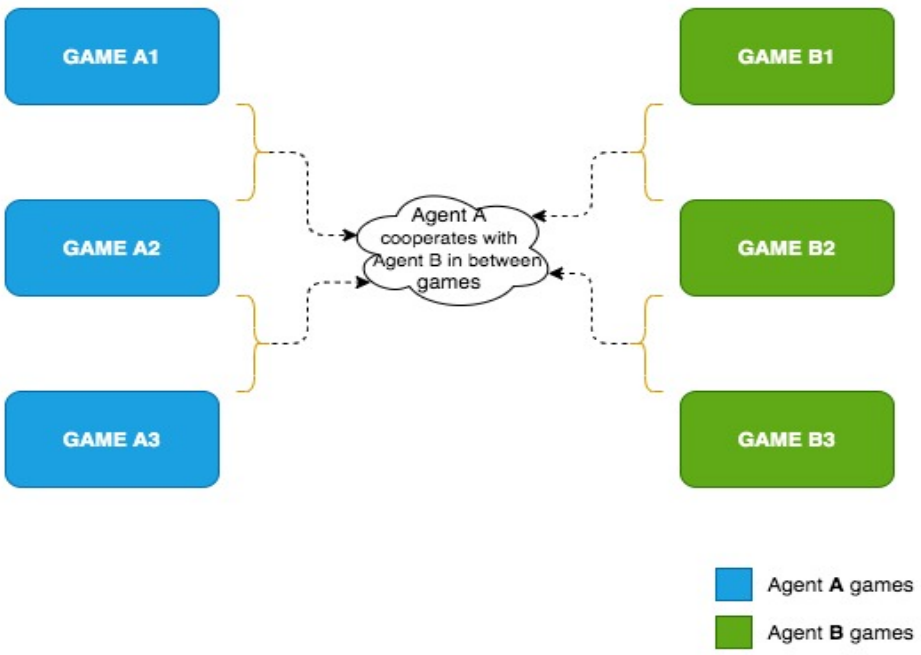


**Figure 4.2** Cooperative Always-on agents using idle time

There are multiple scenarios where agents working on separate tasks would want to cooperate and help each other. We formulate the below scenarios

1. *Exception Handling*- Always-on agent does not know what to do in a situation.

2. *Confidence Boosting*- Always-on agent when not confident of what to do in a situation and wants to boost its confidence.

3. *Different Experiences*- Always-on agent wants to gather different experiences than it has currently.

In this chapter, we make the below contributions

1. We propose a Multi-agent Framework that always-on cooperative agents can use in idle time to analyze in groups. We elaborate on different components in the Multi-agent Framework like the Subscription Layer, Communication Layer, and Voting Layer.

2. We evaluate always-on agents using cooperation in each formulated scenario, exception handling, confidence-boosting, and different experiences.

3. We show improved decision making in always-on cooperative agents when they analyze in groups using the Multi-agent Framework in idle time.

## 4.2 Multi-agent Framework for Cooperative Always-On Agents

We propose a Multi-agent Framework that facilitates always-on agents to use their idle time and cooperate. Agents can subscribe to the framework and analyze alone or in a group. Figure 4.3 illustrates the different components of the proposed Multi-agent Framework.

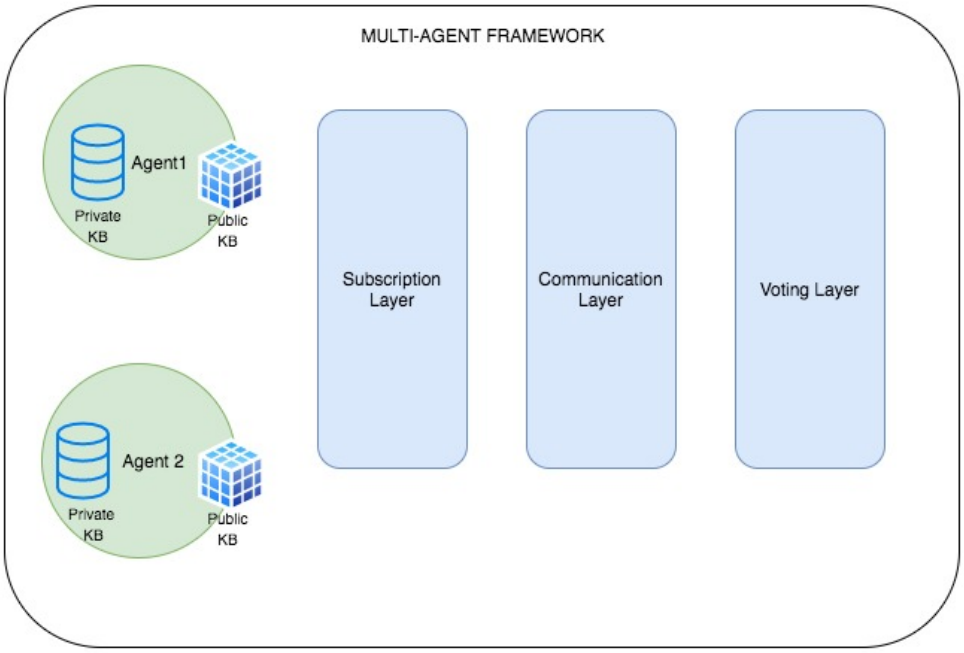


**Figure 4.3** Multi-agent Framework for cooperative always-on agents

### 4.2.1 Components of the Multi-agent Framework

As illustrated in Figure 4.3, different components of the Multi-agent Framework are

1. Always-On Agents

2. Subscription Layer

3. Communication Layer

4. Voting Layer

Each of the proposed framework components is explained in detail in the subsequent subsections.

### 4.2.2 Always-On Agent in the Multi-agent Framework

Each always-on cooperative agent has a Private Knowledge Base and Public Knowledge Base for analysis with other agents.

#### 4.2.2.1 Private Knowledge Base

The Private Knowledge Base is private and is confined to the agent itself. The always-on agent stores the knowledge it owns in this. It can also be defined as a pure database with only the agent's knowledge. Figure 4.4 illustrates a private database accessible only to the agent. Table 4.1 shows a sample Private Knowledge Base of an always-on cooperative agent in the domain of Connect-4. The notations of each column in the database are explained in Section 3.3.1. The Private Knowledge Base is initialized with the schema presented in the Table 4.1 when the agent is initializing.

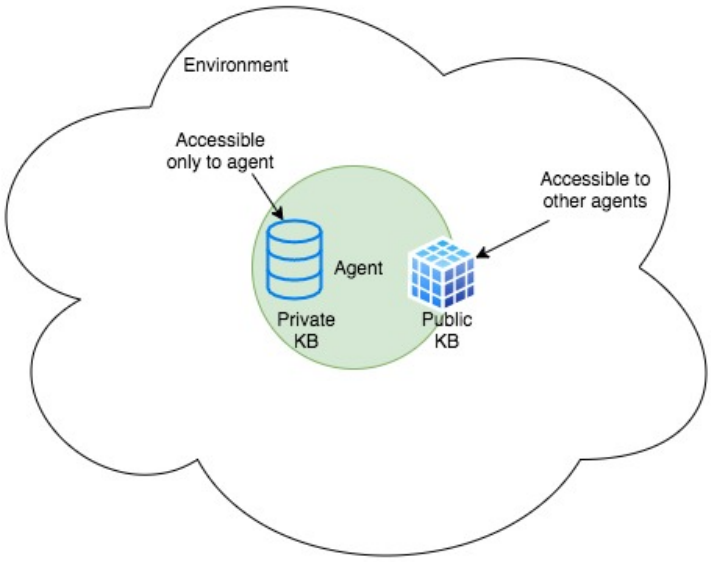


**Figure 4.4** Private and Public Knowledge Base

| Board Position | Goodness | Wins | Losses | Children |
|---|---|---|---|---|
| .....<br>...11..2221112 | 30 | 5 | 3 | [$bp_1$,$bp_2$] |
| .....<br>...11..222111. | 12 | 7 | 2 | [$bp_3$,$bp_1$] |
| .....<br>...11..122111. | 12 | 17 | 3 | [$bp_1$] |

**Table 4.1** Sample Nodes in Private Knowledge Base of Connect-4 agent

#### 4.2.2.2 Public Knowledge Base

The Public Knowledge Base has knowledge the agent owns and acquires from others. The Public Knowledge Base is public to all the agents agreeing to cooperate. When an agent wants to cooperate with others in idle time, it has to give components of Multi-agent Framework access to its Public Knowledge Base. Security is needed for Public Knowledge Base to stop non-cooperative agents from accessing the Public Knowledge Base. Figure 4.4 illustrates a Public Knowledge Base accessible to the outside world. The Public Knowledge Base of all always-on cooperative agents binds them to a contract, while a private database can be of any schema private to the agent.

In Figure 4.5, we illustrate the contract that Public Knowledge Bases of all cooperating agents follow in the domain of Connect-4.

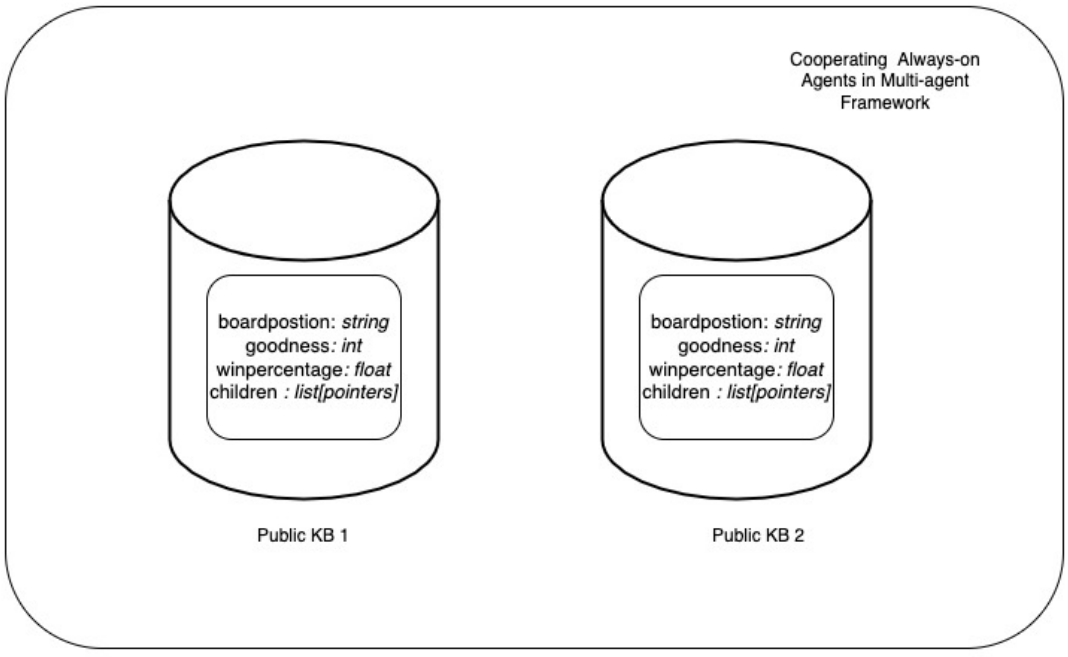


**Figure 4.5** Public Knowledge Base Contract

#### 4.2.2.3 Subscription Layer

The Subscription Layer enables agents to subscribe and unsubscribe to the Multi-agent Framework. Subscribing to the framework means that the agent agrees to cooperate with other agents by providing and receiving suggestions. Unsubscribe means the agent wants to stop cooperating with other agents. The Subscription Layer keeps track of all the always-on agents and their decisions to cooperate or not cooperate in idle time with others. The Subscription Layer between always-on cooperative agents is illustrated in Figure 4.6. The communication from agent to Subscription Layer follows a contract where the agent sends its name, whether it wants to subscribe or not, with a boolean true or false, and an access token to its Public Knowledge Base if it is subscribing. Figure 4.7 illustrates the contract between agents and the Subscription Layer. *Subscription Layer Algorithm* (Algorithm 7) gives the complete algorithm of the Subscription Layer.

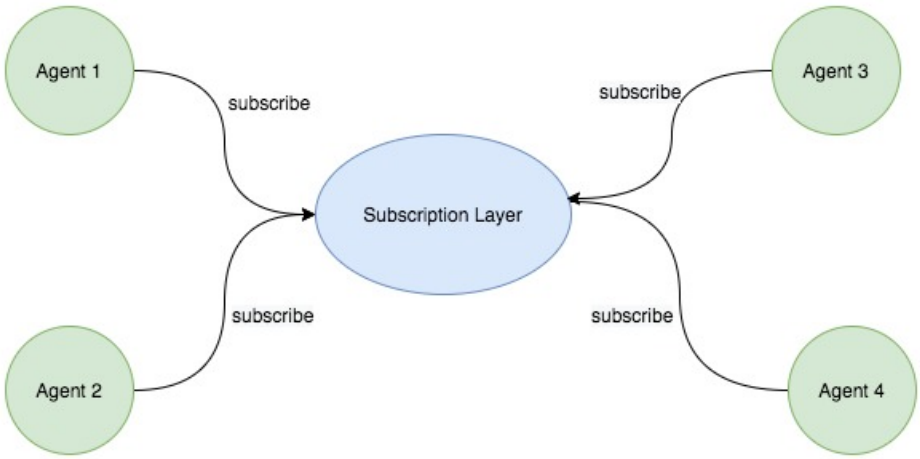


**Figure 4.6** Subscription Layer

**Algorithm 7** Subscription Layer Algorithm

1: $message$= agent's message
2: $globalhash$ = global hash map SL maintains
3: **if** $message[subscribe]$ is $true$ **then**
4: $agent$ = $message[agentname]$
5: $globalhash[agent]$ = $message[accesstoken]$
6: **return** Subscribed
7: **else**
8: $agent$ = $message[agentname]$
9: $globalhash[agent]$ = 0
10: **return** Unsubscribed
11: **end if**

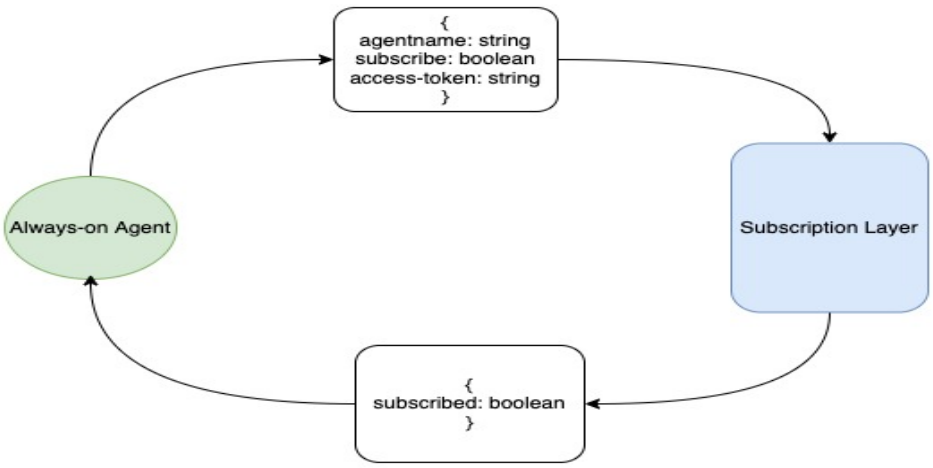


**Figure 4.7** Contract between Subscription Layer and Always-On Agents

#### 4.2.2.4 Communication Layer

Communication Layer handles all requests and responses to and from the always-on agents. An idle always-on cooperative agent A wants to analyze a situation S with other agents sends the Communication Layer an *analyze* request with its name and situation. Figure 4.8 shows the contract between the agent and the Communication Layer. The Communication Layer then connects to the Subscription Layer to get all the subscribed agents and gets suggestions from their Public Knowledge Base. It sends the list of suggestions and agents to the Voting Layer to decide the best among them. Figure 4.9 shows the contract between the Communication Layer and Voting Layer. After the Communication Layer receives the response from the Voting Layer, it incorporates the response in the Public Knowledge Base of the always-on agent. *Communication Layer Algorithm* (Algorithm 8) gives the complete algorithm that the Communication Layer uses.

**Algorithm 8** Communication Layer Algorithm

1: $message$= agent's message
2: $subscribedagents$ = $SubscriptionLayer$.getsubscribedagents()
3: $suggestions$ = List of suggestions
4: **for** $subscribedagent$ in $subscribedagents$ **do**
5: $currentsuggestion$ = $subscribedagent$.getsuggestion($message$.situation)
6: $suggestions$.append($currentsuggestion$,$subscribedagent$)
7: **end for**
8: $votedsuggestion$ = $VotingLayer$.getvotedsuggestion($suggestions$)

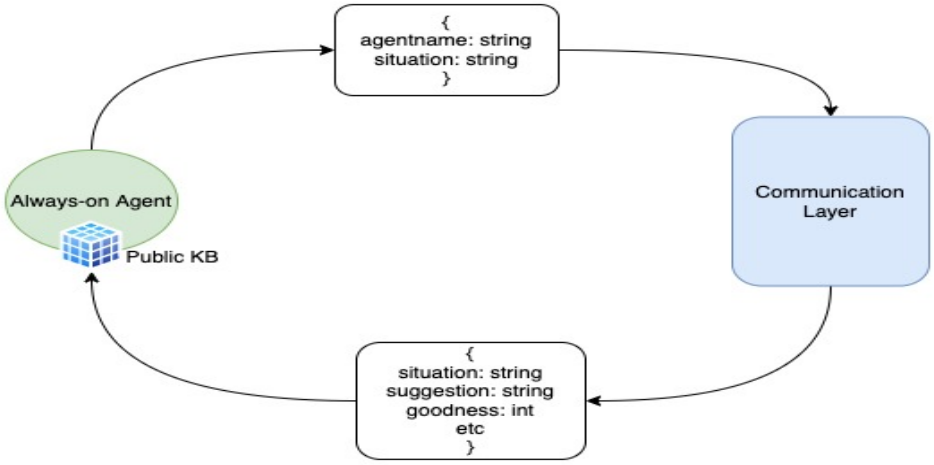


**Figure 4.8** Contract between Communication Layer and Always-On Agents

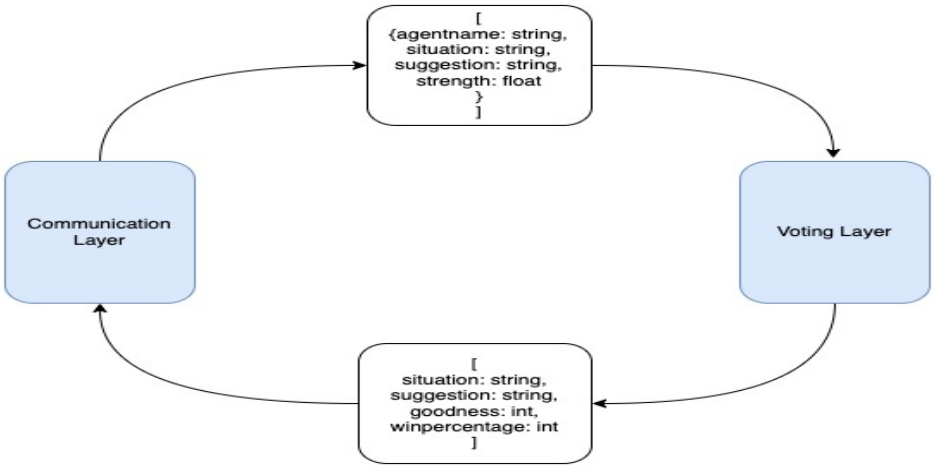


**Figure 4.9** Contract between Communication Layer and Voting Layer

#### 4.2.2.5 Voting Layer

Voting Layer is responsible for deciding the best suggestion among all the suggestions from always-on agents sent by the Communication Layer. Figure 4.9 illustrates the contract between the Communication Layer and Voting Layer. Given multiple suggestions by agents, below are different scenarios the Voting Layer can encounter.

1. No agent has a suggestion for the given situation.
2. Each agent has different suggestions for the given situation.
3. Different groups of agents have different suggestions.
4. All the agents have the same suggestion.

To address the above scenarios, the Voting Layer assigns a vote to a suggestion by taking the agent's strength, the goodness of the situation, and any attributes the agent send depending on the domain. In the case of Connect-4, the Voting Layer gets a list of a board position, the strength of agent suggesting, the goodness of the board position, and the percentage of wins the board position contributed. The question is how much weight is to be given for each of the attributes. We use the RandomForestClassifier classification method to find the feature importance of each of the parameters. The classifier takes individual agents' strength, goodness, and win percentage of board positions as features. Winning or losing (boolean) is the prediction label. Table 4.2 represents the sample training data along with the prediction label to the model. Around 10,000 samples are given to train the model post which the feature importance is calculated. The feature importance output by the model is tabulated in Table 4.3.

| Goodness | WinPercentage | Strength | Win |
|---|---|---|---|
| 40 | 0.2 | 2 | 0 |
| 66 | 0.6 | 4 | 1 |
| -75 | 0.04 | 2 | 0 |
| 105 | 0.7 | 5 | 1 |
| . | . | . | . |
| . | . | . | . |
| 52 | 0.5 | 5 | 0 |

**Table 4.2** Training data for RandomForestClassifier

| Feature | Importance |
|---|---|
| Goodness | 0.64 |
| Strength | 0.29 |
| WinPercentage | 0.05 |

**Table 4.3** Feature importance

Given each input's feature importance, the Voting Layer assigns a vote to a board position by giving the weights according to the feature's importance. Given board position B suggested by an agent with strength $\text{Strength}_\text{i}$ along with situation's Goodness $\text{Goodness}_\text{i}$ and $\text{WinPercentage}_\text{i}$, the vote of a board position is calculated as a weighted sum of Goodness, Strength, and WinPercentage of board position as given in the Equation 4.1. If multiple agents suggest the same board position, then $\text{vote}_\text{i}$ is the sum of the individual votes. The board position with the highest vote (Equation 4.2) is sent to the Communication Layer. *Voting Layer Algorithm* (Algorithm 9) gives the complete algorithm that the Voting Layer uses.

$$vote = 0.64 * Goodness + 0.29 * Strength + 0.05 * WinPercentage \tag{4.1}$$

$$votedsuggestion = suggestion\ corresponding\ to\ highest\ vote \tag{4.2}$$

**Algorithm 9** Voting Layer

```
1: inputlist= input from Communication Layer
2: votes = votes for each suggested boardposition
3: boardpositions = inputlist.boardpositions
4: for each boardpostion in boardpositions do
5:     if boardposition in votes.keys then
6:         votes[boardposition] =
   votes[boardposition] + 0.64*Goodness+0.29*Strength+0.05*WinPercentage
7:     else
8:         votes[boardposition] = 0.64*Goodness+0.29*Strength+0.05*WinPercentage
9:     end if
10: end for
11: suggestionwithmaximumvotes = max(votes, key=votes.get)
12: return suggestionwithmaximumvotes
```

## 4.3 Idle-Time Analysis of multiple Always-On Cooperative Agents workflow

Let A1, A2, A3 are always-on cooperative agents in the system. Below we discuss their idle time and their active time of them.

### 4.3.1 Idle time of Always-On Cooperative Agents

1. Once the agents get idle time, they subscribe to the Subscription Layer by sending the request. The agents also initialize their Public Knowledge Base with schema adhering to the contract. For example in Connect-4, the contract is presented in Figure 4.5.

2. The agents are subscribed, and the Subscription Layer stores the access token to each of the subscribed agent's Public Knowledge Base.

3. Each agent analyzes the games it played privately using the Analyzed Decision Strategy used in Chapter 3. The agent finishes its analysis and populates its Private Knowledge Base and Public Knowledge Base. It identifies the situations where it wants to do one or all of the following

   (a) Exception Handling - When an agent does not know what to do in a situation and needs help from others.

   (b) Confidence Boosting - When an agent knows what to do in a situation but wants to boost confidence by taking other's opinions on the same.

   (c) Different Experiences - When an agent knows what to do next but wants to try new experiences.

4. The identified situations are sent to the Communication Layer to seek suggestions from other agents.

5. The Communication Layer contacts the Subscription Layer to get the list of agents subscribed and the access-tokens for all of their Public Knowledge Bases.

6. The Communication Layer then gets the best action from each of the cooperating agent's Public Knowledge Bases. It sends the list to the Voting Layer to get the best suggestion.

7. The Voting Layer uses the formulation in Section 4.2.2.5 to get the best suggestion and returns that to the Communication Layer.

8. Communication Layer receives the final suggestion from the Voting Layer and inserts it into the Private Knowledge Base of the agent that asked for help.

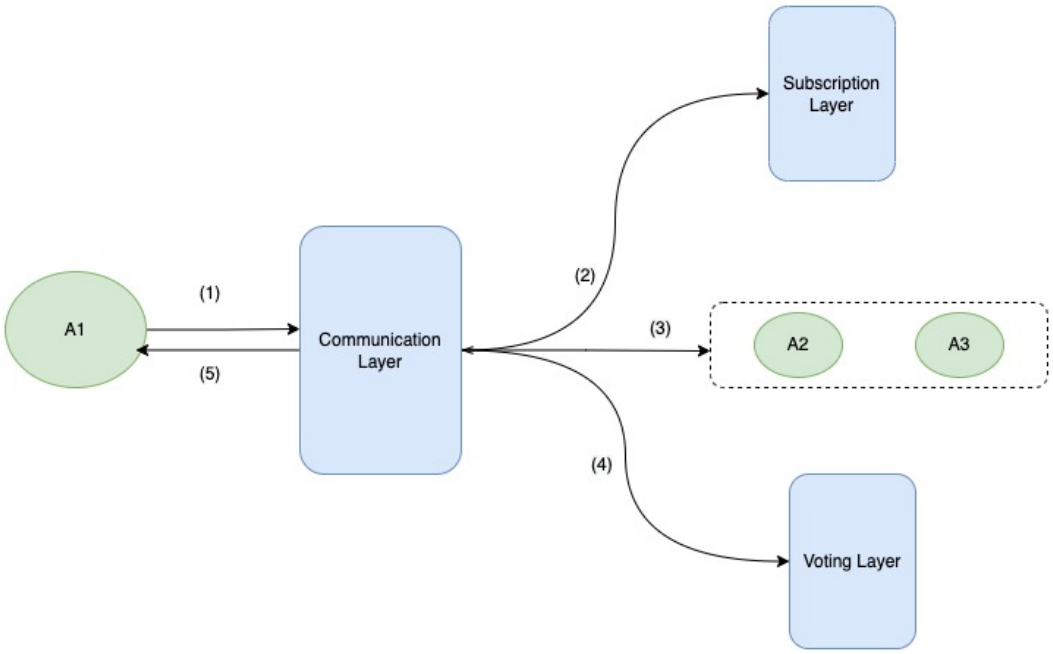


**Figure 4.10** Communication Layer Messages

In case of Connect-4 agents, we mention the Communication Layer messages involved in Figure 4.10. We go through each of them in detail below. In this scenario, always-on agents A1, A2, and A3 are subscribed, and A1 seeks help from Communication Layer

1. Agent A1 requests Communication Layer for help at a board position.
   *Request* - {agentname:A1, situation:{ ".....1.....2.....22....1211." }}

2. Communication Layer contacts Subscription Layer to get all the subscribed agents and their Public Knowledge Base.
   *Request* - "subscribedagents"
   *Response* - {subscribedagents: [{A2:token},{A3:token}] }

3. Communication Layer then contacts all the subscribed agents for suggestions at that particular board position. The best board positions to go to from the current board position are retrieved from each of the Public Knowledge Bases based on goodness and winpercentage (Analyzed Decision Strategy used in Chapter 3).
   *Request* - {agentname:A1, situation: ".....1.....2.....22....1211." }
   *Response* - [ {agentname:A2, situation: ".....1.....2.....22....1211.", suggestion: ".....1.....2.....22....1211.", goodness:30, strength:2, winpercentage:40}, {agentname:A3, situation: ".....1.....2.....22....1211.", suggestion: ".....1.....2.....22....1211.", goodness:20, strength:3, winpercentage:10} ]

4. Communication Layer contacts Voting Layer to choose the best out of all board positions to go to.
   *Request* - [ {agentname:A2, situation: ".....1.....2.....22....1211.", suggestion: ".....1.....2.....22....1211.", goodness:30, strength:2, winpercentage:40}, {agentname:A3, situation: ".....1.....2.....22....1211.", suggestion: ".....1.....2.....22....1211.", goodness:20, strength:3, winpercentage:10} ]
   *Response* - { agentname:A2, situation: ".....1.....2.....22....1211.", suggestion: ".....1.....2.....22....1211.", goodness:20, winpercentage:10}

5. Communication Layer sends the message back to the agent.
   Response - {situation: ".....1.....2.....22....1211.", suggestion: ".....1.....2.....22....1211.", goodness:20, winpercentage:10}

### 4.3.2 Active time of Always-On Cooperative Agents

In active time, the agent does the tasks assigned to achieve its goals in active time. These tasks and their results are stored in a log file. Given a situation in active time, the agent gets the suggestion from its Private Knowledge Base, Public Knowledge Base, and chooses the solution with the best goodness among the two.

## 4.4 Evaluation

We evaluate always-on cooperative agents in the domain of Connect-4. Our always-on Connect-4 agents are deployed in the Multi-agent Framework proposed and use a traditional minimax game engine

with different depths against the stronger opponent. Each of our always-on agents plays plays Connect-4 games with the opponent, and they cooperate in idle time to perform better. We make an assumption that the agents get idle time after every 100 games.

In idle time, the cooperative agents first finish the private analysis using the experience-based learning [14] technique for analyzing and assigning goodness to each board position. As in De Jong and Schultz's [14], the agents store all recorded moves in their Private Knowledge Base in their idle time. Once the Private Knowledge Base is formed with all the recorded games, our agents start with each leaf node. Using the parent pointers, agents propagate values to their parents in a traditional min-max fashion. This process assigns eventual outcomes to each node, capturing its strength from experience. An example for the same is shown in Figure 3.9, each board encountered is given a value based on the games in the experience. Once the agent finishes the analysis, it pushes the analysis to the Public Knowledge Base.

After the private analysis is done, each agent finds the board positions in which it wants suggestions from others and sends them to the Communication Layer. The Communication Layer gets back with the best-suggested move at that board position, and that is incorporated into the Public Knowledge Base. We evaluate each of the strategies proposed that always-on agents can incorporate to cooperate with other agents in the below subsections.

### 4.4.1 Evaluation of Exception Handling Strategy

We evaluate *Exception Handling Strategy* on a group of agents playing Connect-4. For evaluation, we use always-on cooperative Connect-4 agents using a traditional minimax game engine of depths 2, 3, and 4, respectively, to play against an opponent of depth 5. In idle time each cooperative always-on agent analyzes the games it played and retrieves the board positions in which it has no knowledge of the next move to be taken. The retrieved board positions are sent to the Communication Layer to get suggestions from other agents. When the Communication Layer returns with the best suggestion from cooperative agents, it is stored in the agent's Public Knowledge Base. In active time, the agent plays games against its opponent, and when it encounters a position that's not present in its Private Knowledge Base, it checks for the position in Public Knowledge Base and chooses to play that move. By default, the opponent engine has greater depth and significant competition. All the experiments have been repeated a minimum of fifty times, that is, in total 75,000 games against an opponent of depth 5, which does not use idle time analysis strategies. The win/loss data distribution metrics like mean, standard deviation, and ranges across runs for depth 2, 3 and 4 are tabulated in Tables 4.4, 4.5 and 4.6 respectively. We observe that, as the number of games played by the agent increases and the agents cooperate more, the win/loss ratio of our Connect-4 agents increases.

We illustrate how always-on depth 2 agent cooperating with depth 3 and 4 agents is improving its win/loss performance against depth 5 agent (Figure 4.11.a). We illustrate how always-on depth 3 agent cooperating with depth 2 and 4 agents is improving its win/loss performance against depth 5 agent (Figure 4.12.a). We also illustrate how always-on depth 4 agent cooperating with depth 2 and 3 agents

is improving its win/loss performance against depth 5 agent (Figure 4.13.a). We see that the agents using exception handling and cooperating with other agents perform better than the analyzing always-on agents that are non-cooperative.

| No of games | Mean Win/Loss Ratio ($\mu$) | Standard Deviation ($\sigma$) | Min | Max |
|---|---|---|---|---|
| 1-100 | 0.16 | 0.05 | 0.07 | 0.24 |
| 100-200 | 0.37 | 0.07 | 0.21 | 0.47 |
| 200-300 | 0.48 | 0.07 | 0.31 | 0.58 |
| 300-400 | 0.57 | 0.19 | 0.32 | 0.87 |
| 400-500 | 0.63 | 0.12 | 0.41 | 0.89 |

**Table 4.4** Depth2 against Depth5 agent using Exception Handling Strategy

| No of games | Mean Win/Loss Ratio ($\mu$) | Standard Deviation ($\sigma$) | Min | Max |
|---|---|---|---|---|
| 1-100 | 0.25 | 0.04 | 0.17 | 0.32 |
| 100-200 | 0.4 | 0.12 | 0.16 | 0.60 |
| 200-300 | 0.58 | 0.23 | 0.32 | 0.97 |
| 300-400 | 0.67 | 0.23 | 0.42 | 1.12 |
| 400-500 | 0.71 | 0.17 | 0.46 | 0.97 |

**Table 4.5** Depth3 against Depth5 agent using Exception Handling Strategy

| No of games | Mean Win/Loss Ratio ($\mu$) | Standard Deviation ($\sigma$) | Min | Max |
|---|---|---|---|---|
| 1-100 | 0.61 | 0.11 | 0.50 | 0.84 |
| 100-200 | 0.79 | 0.13 | 0.60 | 1.04 |
| 200-300 | 0.85 | 0.25 | 0.52 | 1.31 |
| 300-400 | 0.89 | 0.26 | 0.57 | 1.33 |
| 400-500 | 0.95 | 0.14 | 0.66 | 1.15 |

**Table 4.6** Depth4 against Depth5 agent using Exception Handling Strategy

### 4.4.2 Evaluation of Confidence Boosting Strategy

We evaluate *Confidence Boosting Strategy* on a group of agents playing Connect-4. For evaluation, we use cooperative always-on Connect-4 agents using a traditional minimax game engine of depths 2, 3, and 4, respectively, to play against depth 5 opponent. In idle time each cooperative always-on agent analyzes the games it played and retrieves the board positions in which it has less confidence on the next move to be taken. In Connect 4, the confidence of move is decided by the score of board position we land in using the move. The score of a board position is calculated using Equation 3.2. The board positions with score less than 10 (mean of scores) are identified. The retrieved board positions are sent to the Communication Layer to get suggestions from other agents. When Communication Layer returns with a suggestion from cooperative agents, it is stored in the agent's Public Knowledge Base. In active time, the agent plays games against its opponent, and when it encounters a position that it is not

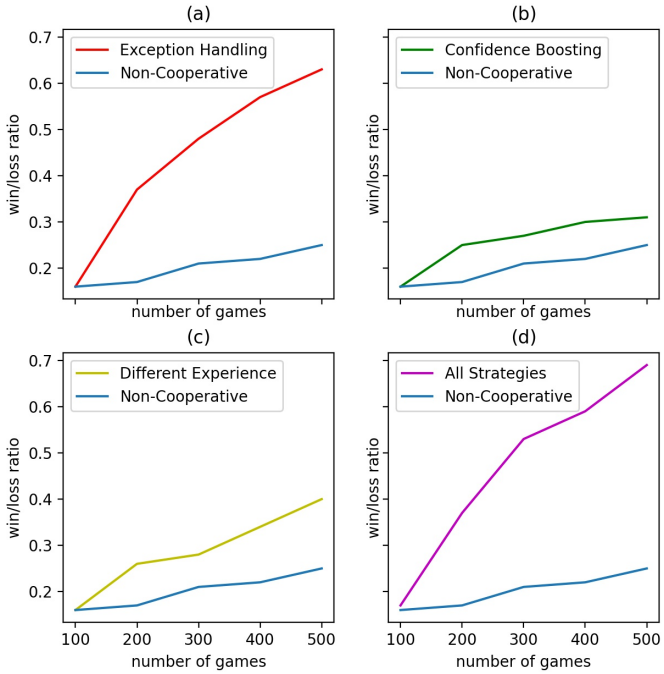


**Figure 4.11** Depth2 agent cooperating with Depth3, Depth4 against Depth5

confident with, it checks for the position in Public Knowledge Base and chooses to play that move if that move is better than its move based on the score. By default, the opponent engine has greater depth and significant competition. All the experiments have been repeated a minimum of fifty times, that is, in total 75,000 games against an opponent of depth 5, which does not use idle time analysis strategies. The win/loss data distribution metrics like mean, standard deviation, and ranges across runs for depth 2, 3 and 4 are tabulated in Tables 4.7, 4.8 and 4.9 respectively. We observe that, as the number of games played by the agent increases and the agents cooperate more, the win/loss ratio of our Connect-4 agents increases.

We illustrate that always-on depth 2 agent cooperating with depth 3 and 4 agents is improving its win/loss performance against depth 5 agent (Figure 4.11.b). We illustrate that always-on depth 3 agent cooperating with depth 2 and 4 agents is improving its win/loss performance against depth 5 agent (Figure 4.12.b). We illustrate that always-on depth 4 agent cooperating with depth 2 and 3 agents is improving its win/loss performance against depth 5 agent (Figure 4.13.b). We see that the agents with depth 2 and depth 3 using confidence boosting strategy and cooperating with other agents perform better than the analyzing always-on agents that are non-cooperative. In the case of depth 4 agent, the agent could not get a confidence boost from weaker agents and could not make significant improvement.

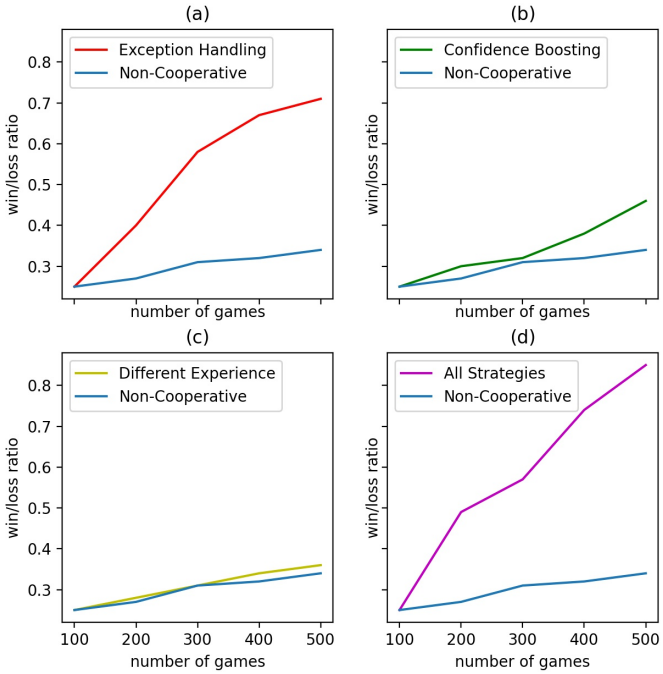


**Figure 4.12** Depth3 agent cooperating with Depth2, Depth4 against Depth5

| No of games | Mean Win/Loss Ratio ($\mu$) | Standard Deviation ($\sigma$) | Min | Max |
|---|---|---|---|---|
| 1-100 | 0.16 | 0.05 | 0.07 | 0.24 |
| 100-200 | 0.25 | 0.09 | 0.09 | 0.46 |
| 200-300 | 0.27 | 0.05 | 0.14 | 0.36 |
| 300-400 | 0.30 | 0.15 | 0.13 | 0.60 |
| 400-500 | 0.31 | 0.10 | 0.21 | 0.63 |

**Table 4.7** Depth2 against Depth5 agent using Confidence Boosting Strategy

### 4.4.3 Evaluation of Different Experiences Strategy

We evaluate *Different Experiences Strategy* on a group of agents playing Connect-4. We use cooperative always-on Connect-4 agents for evaluation using a traditional minimax game engine of depths 2, 3, 4, respectively, to play against the depth 5 opponent. In idle time each always-on cooperative agent analyzes the games it played and retrieves the board positions in which it wants to explore a different move to be taken. This is chosen if the agent is already confident of the move to take. In Connect 4, the confidence of move is decided by the board position score we land in using the move. The score of a board position is calculated using Equation 3.1. The board positions with a score greater than or equal to 10 (mean of scores) are identified. The retrieved board positions are sent to the Communication Layer to get suggestions from other agents. When Communication Layer returns with a suggestion from

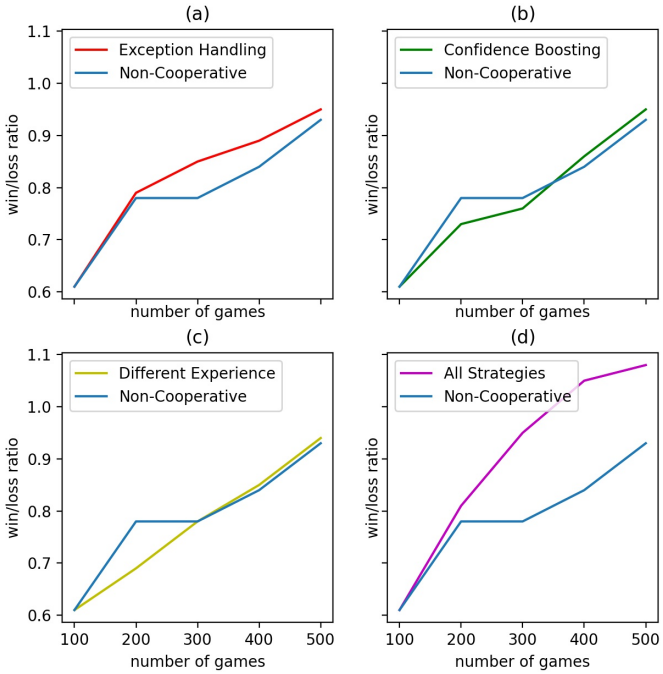


**Figure 4.13** Depth4 agent cooperating with Depth2, Depth3 against Depth5

| No of games | Mean Win/Loss Ratio ($\mu$) | Standard Deviation ($\sigma$) | Min | Max |
|---|---|---|---|---|
| 1-100 | 0.25 | 0.08 | 0.11 | 0.33 |
| 100-200 | 0.30 | 0.07 | 0.20 | 0.46 |
| 200-300 | 0.32 | 0.11 | 0.15 | 0.54 |
| 300-400 | 0.38 | 0.11 | 0.16 | 0.56 |
| 400-500 | 0.46 | 0.07 | 0.32 | 0.59 |

**Table 4.8** Depth3 against Depth5 agent using Confidence Boosting Strategy

cooperative agents, it is stored in the agent's Public Knowledge Base. In active time, the agent plays games against its opponent. When it encounters a position that it is already very confident with, it checks for the position in Public Knowledge Base and explores to play that move if that move is better than its own based on score. By default, the opponent engine has greater depth and significant competition. All the experiments have been repeated a minimum of fifty times, that is, in total 75,000 games against an opponent of depth 5 that does not use idle time analysis strategies. The win/loss data distribution metrics like mean, standard deviation, and ranges across runs for depth 2, 3 and 4 are tabulated in Tables 4.10, 4.11 and 4.12 respectively.

We illustrate that always-on depth 2 agent cooperating with depth 3 and 4 agents is improving its win/loss performance against depth 5 agent (Figure 4.11.c). We illustrate that always-on depth 3 agent cooperating with depth 2 and 4 agents is improving its win/loss performance against depth 5 agent

| No of games | Mean Win/Loss Ratio ($\mu$) | Standard Deviation ($\sigma$) | Min | Max |
|---|---|---|---|---|
| 1-100 | 0.61 | 0.06 | 0.56 | 0.76 |
| 100-200 | 0.73 | 0.17 | 0.48 | 1.3 |
| 200-300 | 0.76 | 0.20 | 0.49 | 1.55 |
| 300-400 | 0.86 | 0.38 | 0.51 | 1.4 |
| 400-500 | 0.95 | 0.33 | 0.60 | 1.51 |

**Table 4.9** Depth4 against Depth5 agent using Confidence Boosting Strategy

(Figure 4.12.c). We illustrate that always-on depth 4 agent cooperating with depth 2 and 3 agents is improving its win/loss performance against depth 5 agent (Figure 4.13.c). We see that the agents with depth 2 and 3 using different experiences and cooperating with other agents perform better than the always-on analyzing agents that are non-cooperative. In the case of depth 4 agent (Figure 4.13.c) the cooperative agent is not showing significant improvement than the non-cooperative one who contemplates with itself because it is trying to gather different experiences from lower depth agents. It could not collect experiences better than what it already has.

| No of games | Mean Win/Loss Ratio ($\mu$) | Standard Deviation ($\sigma$) | Min | Max |
|---|---|---|---|---|
| 1-100 | 0.16 | 0.03 | 0.09 | 0.19 |
| 100-200 | 0.26 | 0.09 | 0.13 | 0.36 |
| 200-300 | 0.28 | 0.04 | 0.19 | 0.31 |
| 300-400 | 0.34 | 0.1 | 0.23 | 0.51 |
| 400-500 | 0.40 | 0.05 | 0.30 | 0.44 |

**Table 4.10** Depth2 against Depth5 agent using Different Experiences Strategy

| No of games | Mean Win/Loss Ratio ($\mu$) | Standard Deviation ($\sigma$) | Min | Max |
|---|---|---|---|---|
| 1-100 | 0.25 | 0.02 | 0.18 | 0.28 |
| 100-200 | 0.28 | 0.01 | 0.26 | 0.33 |
| 200-300 | 0.31 | 0.07 | 0.15 | 0.38 |
| 300-400 | 0.34 | 0.06 | 0.17 | 0.40 |
| 400-500 | 0.35 | 0.09 | 0.20 | 0.42 |

**Table 4.11** Depth3 against Depth5 agent using Different Experiences Strategy

### 4.4.4 Summary

All the results of the strategies mentioned above are summarized in Table 4.13 where an event is a strategy in use, and cooperating agents are the agents with which the agent we are evaluating is cooperating, depth of agent we are evaluating is in column agents depth, w/l non-cooperative is the win/loss ratio agent attains by the end of 500 games (where the agent gets idle every 100 games and contemplates with itself), w/l cooperative is the win/loss ratio agent attains the end of 500 games (where the agent gets idle every 100 games and cooperates with others). We show the details for each strategy,

| No of games | Mean Win/Loss Ratio ($\mu$) | Standard Deviation ($\sigma$) | Min | Max |
|---|---|---|---|---|
| 1-100 | 0.61 | 0.19 | 0.46 | 0.73 |
| 100-200 | 0.69 | 0.12 | 0.59 | 0.77 |
| 200-300 | 0.78 | 0.11 | 0.65 | 0.82 |
| 300-400 | 0.85 | 0.07 | 0.73 | 0.85 |
| 400-500 | 0.94 | 0.10 | 0.79 | 1.0 |

**Table 4.12** Depth4 against Depth5 agent using Different Experiences Strategy

and we compare the win/loss ratio of agents cooperating versus non-cooperating. We also show always-on agent employing all the proposed strategies for depths 2, 3 and 4 in Figure 4.11.d, Figure 4.12.d and Figure 4.13.d respectively. Among the individual strategies employed by agents, it is seen that exception handling strategy in idle-time cooperation helps improve the performance of all the agents most. We observe that agent employing all strategies performs better than individual strategies, and employing all strategies help improve the performance of all the agents significantly (Table 4.13).

| Event | Cooperating Agents | Agents Depth | W/L Non-Cooperative | W/L Cooperative |
|---|---|---|---|---|
| Exception Handling | Depth3, Depth4 | Depth2 | 0.25 | 0.63 |
| Confidence Boosting | Depth3, Depth4 | Depth2 | 0.25 | 0.31 |
| Different Experiences | Depth3, Depth4 | Depth2 | 0.25 | 0.40 |
| All Strategies | Depth3, Depth4 | Depth2 | 0.25 | 0.69 |
| Exception Handling | Depth2, Depth4 | Depth3 | 0.34 | 0.71 |
| Confidence Boosting | Depth2, Depth4 | Depth3 | 0.34 | 0.46 |
| Different Experiences | Depth2, Depth4 | Depth3 | 0.34 | 0.35 |
| All Strategies | Depth2, Depth4 | Depth3 | 0.34 | 0.85 |
| Exception Handling | Depth2, Depth3 | Depth4 | 0.93 | 0.94 |
| Confidence Boosting | Depth2, Depth3 | Depth4 | 0.93 | 0.95 |
| Different Experiences | Depth2, Depth3 | Depth4 | 0.93 | 0.94 |
| All Strategies | Depth2, Depth3 | Depth4 | 0.93 | 1.01 |

**Table 4.13** Results Summary

*Chapter 5*

# Applications

In previous chapters, we proposed different strategies for idle time driven decision making and evaluated them on Connect-4. In this chapter, we discuss the application of idle time driven decision making in other domains. We present how we use the proposed strategies in chess and show some initial experiments. We also briefly discuss on how we can use idle time of agents in other domains.

## 5.1 Idle time Analysis in Chess

Chess is a competitive and very popular recreational board game played between two players. It is played on a square chessboard with 64 squares arranged in an eight-by-eight grid [43]. We conduct an analysis of idle time driven improved decision-making for an always-on agent playing chess.

### 5.1.1 Analysis Database

To represent a chessboard position, we have to represent the contents of the board. We use english alphabets to represent the pieces on the chessboard where capital letters represent the white pieces, and small letters represent the black pieces. Space represents an empty square on board. Standard mapping of characters to pieces is followed as shown in Table 5.1. Each row in a chessboard is represented as a 8-character string which is a concatenation of contents of each square from left to right in that row. The board position is represented as a 64-character string which is a concatenation of each row from top to bottom.

Standard Algebraic Notation (SAN) [42] is used to represent the moves in a game. An integer represents the frequency of each move. Table 5.2 shows sample nodes in the analysis database where board position, move for that board position, and frequency of that move are the columns.

### 5.1.2 Evaluation

For evaluation of the idle time analysis in chess playing always-on agents, we use open source chess engines. To set up the interaction with engines, Universal Chess Interface (UCI) [44] which is a protocol

| Piece | Black Piece Representation | White Piece Representation |
|---|---|---|
| Rook | r | R |
| Bishop | b | B |
| Knight | n | N |
| Queen | q | Q |
| Pawn | p | P |
| King | k | K |

**Table 5.1** Chess Piece Representation

| Board Position | Move | Frequency |
|---|---|---|
| r bqkb rppp pppp n n p<br>P N N PPP PPPPR BQKB R | Bf4 | 4 |
| r qkb rppp ppp n p p n<br>P bN N B PPP PPPPR QKB R | h3 | 5 |

**Table 5.2** Sample Nodes in Analysis Database for chess

for communicating with different engines is used. UCI is used to open a local chess engine process and the following experiments are conducted using it. The agent uses the chess engine's choice of next move if it is not possible to use the stored data for that board position. The agent stores game as board positions and moves. We do not store the opening moves in the database.

**Evaluation of Frequent Decision Strategy:** For the evaluation of Frequent Decision Strategy, our chess agent uses an open-source chess engine *glaurung* [11] with depth 1 to play as white against the opponent agent, which uses *glaurung* engine with depth 0 as black. As by default *glaurung* engine favors black; this is potential competition. Figure 5.1 illustrates the win/loss ratio of chess agent using Frequent Decision Strategy (FDS) in idle time versus chess agent not using idle time.

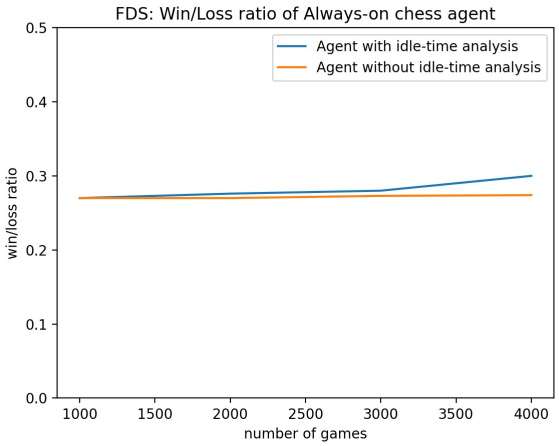


**Figure 5.1** Chess Frequent Decision Strategy

**Evaluation of Analyzed Decision Strategy:** For the evaluation of Analyzed Decision Strategy, our agent uses an open-source chess engine *glaurung* engine with depth 1 to play as white against the opponent agent, which uses *glaurung* engine with depth 0 as black. As by default *glaurung* engine favors black; this is a potential competition. In idle time the always-on agent uses "lichess.org," [18], an on-line chess game provider for analyzing the games it lost and getting the annotated pgn. Figure 5.2 gives a sample pgn file, and Figure 5.3 gives the analyzed annotated version of the given pgn. The annotated pgn and has alternate moves for the wrong moves in the given pgn. For example, at move number 25, it is annotated that a better move is Rd1 instead of Rf1 for that board position. We parse these annotated files and store the alternate better moves in the analysis database. We except the opening moves while doing this. In a new game, if the board position appears in the analysis database, the move corresponding to that is chosen. Figure 5.4 illustrates the win/loss ratio of chess agent using analyzed decision strategy in idle time versus chess agent not using idle time. The results are shown against an opponent chess agent without idle time analysis.

```
[Event "EU-chT prel"]
[Site "Aarhus"]
[Date "1971.09.09"]
[Round "5"]
[White "Uhlmann, Wolfgang"]
[Black "Larsen, Bent"]
[Result "1-0"]
[WhiteElo "2570"]
[BlackElo "2660"]
[ECO "A42"]

1.c4 g6 2.e4 Bg7 3.d4 d6 4.Nc3 e5 5.dxe5 dxe5 6.Qxd8+ Kxd8 7.f4 Nc6 8.fxe5 Be6
9.Bg5+ Kc8 10.Nf3 h6 11.Bf4 g5 12.Be3 Nge7 13.O-O-O Nxe5 14.Nd5 N7g6 15.Bd4 b6
16.c5 c6 17.Ba6+ Kb8 18.Ne7 Nxe7 19.Bxe5+ Bxe5 20.Nxe5 Kc7 21.Bc4 Rad8 22.Bxe6 fxe6
23.Rd6 bxc5 24.Rxe6 Rh7 25.Rf1 Rd4 26.Rf7 Rxf7 27.Nxf7 Kd7 28.Rxh6 Rc4+ 29.Kd2 Rxe4
30.g4 Nd5 31.h3 Rd4+ 32.Ke2 Re4+ 33.Kd3 Rd4+ 34.Ke2 Re4+ 35.Kd2 Rd4+ 36.Kc2 Nb4+
37.Kb3 Rd3+ 38.Kc4 Rd5 39.a3 Nd3 40.Kc3 Ke7 41.Rh7 Kf6 42.Nxg5 Kg6 43.Rh8 Nf2
44.Rg8+ Kh6 45.Ne6 Ne4+ 46.Kc2 Rd6 47.g5+ Kh5 48.Rh8+ Kg6 49.Nxc5  1-0
```

**Figure 5.2** Sample pgn file

```
[Event "EU-chT prel"]
[Site "https://lichess.org/ENGR4ShH"]
[Date "1971.09.09"]
[White "Uhlmann, Wolfgang (2570)"]
[Black "Larsen, Bent (2660)"]
[Result "1-0"]
[WhiteElo "?"]
[BlackElo "?"]
[PlyCount "97"]
[Variant "Standard"]
[TimeControl "-"]
[ECO "A42"]
[Opening "Modern Defense: Averbakh Variation"]
[Termination "Normal"]
[Annotator "lichess.org"]

1. c4 g6?! { (-0.08 → 0.54) Inaccuracy. Best move was e5. } (1... e5 2. e3 Nf6 3. Nf3 e4 4. Nd4
Nc6 5. d3 Bb4+ 6. Bd2 Bxd2+ 7. Nxd2 Nxd4 8. exd4) 2. e4 Bg7 3. d4 d6 4. Nc3 { A42 Modern Defense:
Averbakh Variation } e5 5. dxe5 dxe5 6. Qxd8+ Kxd8 7. f4 Nc6 8. fxe5 Be6 9. Bg5+ Kc8 10. Nf3 h6
11. Bf4 g5 12. Be3 Nge7 13. O-O-O Nxe5 14. Nd5 N7g6 15. Bd4 b6? { (-0.06 → 1.03) Mistake. Best
move was c6. } (15... c6 16. Ne7+ Nxe7 17. Nxe5 f6 18. Nf3 Re8 19. h3 b6 20. Bc3 Kc7 21. Be2 c5
22. b3) 16. c5 c6 17. Ba6+ Kb8 18. Ne7 Nxe7 19. Bxe5+ Bxe5 20. Nxe5 Kc7 21. Bc4 Rad8?! { (1.36 →
1.97) Inaccuracy. Best move was Rhd8. } (21... Rhd8 22. Bxe6) 22. Bxe6 fxe6 23. Rd6 bxc5 24. Rxe6
Rh7 25. Rf1? { (2.42 → 1.34) Mistake. Best move was Rd1. } (25. Rd1 Rxd1+ 26. Kxd1 Kb7 27. Kc2
Nc8 28. Rxc6 c4 29. Rxc4 Nb6 30. Rc5 Na4 31. Rd5 Nb6) 25... Rd4? { (1.34 → 2.45) Mistake. Best
move was Rd6. } (25... Rd6 26. Rxd6 Kxd6 27. Nc4+ Kc7 28. Rf6 Ng8 29. Rg6 Ne7 30. Re6 Nc8 31. Na5
Ne7 32. Nb3) 26. Rf7? { (2.45 → 0.67) Mistake. Best move was Rd1. } (26. Rd1 Kb7 27. Rxd4 cxd4
28. Kd2 h5 29. Kd3 c5 30. Nd7 Kc7 31. Nxc5 Nc6 32. Kc4 g4) 26... Rxf7 27. Nxf7 Kd7 28. Rxh6 Rc4+
29. Kd2 Rxe4 30. g4 Nd5 31. h3 Rd4+ 32. Ke2 Re4+ 33. Kd3 Rd4+ 34. Ke2 Re4+ 35. Kd2 Rd4+ 36. Kc2?!
{ (0.77 → 0.00) Inaccuracy. Best move was Kc1. } (36. Kc1 Ne3 37. b3 Rd1+ 38. Kb2 Rd2+ 39. Kb1
Rd1+ 40. Kb2 Rd2+ 41. Kc3 Rxa2 42. Ne5+ Kd8) 36... Nb4+?! { (0.00 → 0.77) Inaccuracy. Best move
was Ne3+. } (36... Ne3+ 37. Kc3 Nd1+ 38. Kc2 Ne3+ 39. Kc3 Nd1+ 40. Kc2 Ne3+) 37. Kb3 Rd3+ 38. Kc4
Rd5 39. a3 Nd3 40. Kc3?! { (0.70 → 0.00) Inaccuracy. Best move was Nxg5. } (40. Nxg5 Nxb2+ 41.
Kb3 Nd1 42. Ne4 Re5 43. Nf6+ Ke7 44. Kc4 Nb2+ 45. Kc3 Na4+ 46. Kd3 Nb2+) 40... Ke7? { (0.00 →
1.69) Mistake. Best move was Nf2. } (40... Nf2 41. Kc4 Ne4 42. Rg6 Ke7 43. Nh6 Rd4+ 44. Kb3 Rd3+
45. Kc4 Rd4+ 46. Kb3 Rd3+ 47. Kc4) 41. Rh7 Kf6 42. Nxg5 Kg6 43. Rh8 Nf2? { (1.08 → 2.08) Mistake.
Best move was Kxg5. } (43... Kxg5 44. Rh5+ Kf4 45. Rxd5 cxd5 46. Kxd3 a5 47. b3 Kg5 48. Kd2 Kh4
49. Kc2 Kg5 50. Kc3) 44. Rg8+ Kh6 45. Ne6 Ne4+?! { (1.98 → 2.85) Inaccuracy. Best move was
Nd1+. } (45... Nd1+) 46. Kc2 Rd6? { (2.80 → 4.10) Mistake. Best move was a6. } (46... a6 47. Rh8+
Kg6 48. Nf4+ Kg7 49. Nxd5 Kxh8 50. Ne3 Kg7 51. h4 Kg6 52. Kd3 Nf6 53. Kc4) 47. g5+ Kh5? { (3.94 →
4.99) Mistake. Best move was Kh7. } (47... Kh7 48. Rg7+ Kh8 49. Re7 Kg8 50. h4 Rd2+ 51. Kc1 Re2
52. Nxc5 Nc3 53. Rxa7 Nd5 54. Ra4) 48. Rh8+? { (4.99 → 3.21) Mistake. Best move was Nf4+. } (48.
Nf4+ Kh4 49. g6 Rd2+ 50. Kb3 Rd7 51. g7 Nf6 52. Rh8+ Kg5 53. Nh5 Kg6 54. Nxf6 Rxg7) 48... Kg6 49.
Nxc5 { Black resigns } 1-0
```

**Figure 5.3** Annotated pgn file

**Evaluation of Online Analysis Decision Strategy:** For evaluation of the Online Analysis Decision Strategy, we show the quicker decision making of our chess agent. Our chess agent uses an open-source chess engine *stockfish* [37] with depth 1 to play against the opponent agent which also uses *stockfish* with depth 0. In Table 5.3 the first column gives the opponent response time, which is the time given to the opponent per move in milliseconds. The second column shows how the difference between the average response time (ART) per move without using online analysis and using online analysis changes with the opponent response time. We see in Table 5.3 that the decrease in ART per move increases as the opponent takes more time to respond. This decrease in average response time shows that the agent's idle time when an opponent is responding can decrease the agent's response time.

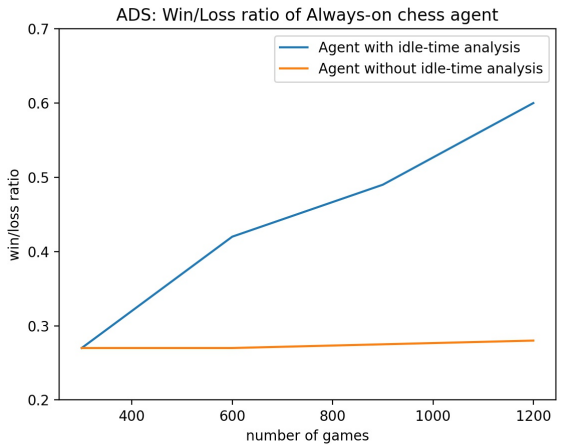


**Figure 5.4** Chess Analyzed Decision Strategy

| Opponent Move Time (milli sec) | Decrease in Average Response Time |
|---|---|
| 80 | 0.11 |
| 100 | 0.66 |
| 125 | 0.9 |
| 150 | 0.92 |
| 175 | 1.04 |
| 200 | 1.06 |

**Table 5.3** Results of Online Analysis Decision Strategy

## 5.2 Application to other domains

We can apply idle time driven decision making to various domains. We can use them in vast warehouses that often employ robot agents to manage the products [36]. A warehouse needs to store the products and make them available to the buyers as and when required. They need to sort the products and optimize the time for picking and packing in active time. In idle time, the agents in the warehouse can employ idle time strategies to do the below

- Contemplate the paths they have taken in the active time to find optimal ways to pick or pack.
- Contemplate the frequency of orders, the weight of products, and fragility to optimize the product arrangement in the warehouse.

While using idle time, the agents can also cooperate with peer agents to find optimal ways to pick or pack using proposed Multi-agent Framework (Section 4.2.2). The agents can cooperate with peers in the below scenarios.

- Exception Handling - If the agent encounters a situation where it does not know what to do, it employs an exception handling strategy. In this case, when the agent is new to the warehouse and

has failed to pick a product, it cooperates with peer agents to get the path followed by them in idle time.

- Confidence Boosting - If the agent encounters a situation where it does not have confidence in a particular action it took, it employs a confidence boosting strategy. In this case, when the agent is not confident about the path it took to pick a product, it checks with cooperating peers in idle time on how they did it.

- Different Experiences - If the agent wants to explore other choices rather than taking the ones it knows, it employs a different experiences strategy. In this case, when the agent wants to check other paths to pick a product, it cooperates with other agents to understand what other techniques they follow.

As detailed above, the warehouses employing agents can incorporate idle-time driven decision making and improve their performance.

Personal assistant agents for calendar management can use idle time driven decision-making as well. The agent can use idle time to understand the user's preferences and employ different strategies discussed to analyze the user's decisions and improve the user's schedule. The agent can also use idle time to cooperate with peer personal assistant agents and get their suggestions wherever needed to improve the scheduling activities.

*Chapter 6*

# Conclusion

This thesis attempts to incorporate and evaluate idle time driven decision making in always-on agents. The contribution of this thesis is twofold: First, we presented always-on agents and idle time utilization in them. We defined two types of idle times (i) when an agent is not assigned any task (ii) when an agent is assigned a task but has some time intervals during the execution of the task. We presented and evaluated different mechanisms that agents can use in idle time like Frequent Decision Strategy (FDS), Analyzed Decision Strategy (ADS), and Online Analysis Decision Strategy (OADS). We conducted our empirical study on always-on agents playing Connect-4 and evaluated each strategy based on the number of games won and the decrease in response time. We show that an agent can analyze its decisions in idle time and apply that for better decision-making.

Secondly, we introduced idle time analysis in always-on cooperative agents. We formulated scenarios in which the agent has a rationale to cooperate with other agents
(i) Exception Handling - When an always-on agent does not know what to do in a situation.
(ii) Confidence Boosting - Always-on agent when not confident of what to do in a situation and wants to boost its confidence.
(iii) To obtain different experiences- Always-on agent wants to gather different experiences than it has currently.

We presented a Multi-agent Framework that cooperating always-on agents can use in the above scenarios and elaborated on its components Subscription Layer, Communication Layer, and Voting Layer. The proposed Multi-agent Framework helps the always-on cooperative agents use their idle time to cooperate in groups. We evaluated cooperating always-on agents of different strengths and showed improved decision-making in them when they analyzed in groups in idle time. Our results show that the agents can perform better when they use idle-time to cooperate with peer agents.

Our evaluations proved our hypothesis that an always-on agent can benefit by using idle time rather than being idle.

## 6.1 Future Work

With this thesis, we attempted to incorporate and evaluate idle time driven decision making in always-on agents. Our methods are inspired by human contemplation and the need for explainability in intelligent systems. They open many doors of opportunity for further research in exploring idle time in always-on agents. We hope to extend this work to a system where always-on agents summarize their idle time learnings and are more like humans.

One limitation of our work is that we used direct communication between peer agents (discussed in Section 2.4.1), and there are cases where direct communication may not work. As part of future work, we want to investigate indirect communication between peer agents in complex domains. We also want to make use of idle time in cooperation between cross-domain peer agents. It will be exciting to apply idle time driven decision making in always-on agents to domains like health-care and transport.

# Related Publications

1. **Empirical Evaluation of Idle-Time Analysis Driven Improved Decision Making by Always-On Agents**, Garapati, Sravyasri, and Kamalakar Karlapalem, 2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC).

# Bibliography


[1] Hamid R Berenji and David Vengerov. Advantages of cooperation between reinforcement learning agents in difficult stochastic problems. In *Ninth IEEE International Conference on Fuzzy Systems. FUZZ-IEEE 2000 (Cat. No. 00CH37063)*, volume 2, pages 871–876. IEEE, 2000.

[2] Cristiano Castelfranchi. Commitments: From individual intentions to groups and organizations. In *ICMAS*, volume 95, pages 41–48, 1995.

[3] Georgios Chalkiadakis and Craig Boutilier. Coordination in multiagent reinforcement learning: A bayesian approach. In *Proceedings of the second international joint conference on Autonomous agents and multiagent systems*, pages 709–716, 2003.

[4] Victor Chan, Pradeep Ray, and Nandan Parameswaran. Mobile e-health monitoring: an agent-based approach. *IET communications*, 2(2):223–230, 2008.

[5] Zhiyuan Chen and Bing Liu. Lifelong machine learning. *Synthesis Lectures on Artificial Intelligence and Machine Learning*, 10(3):1–145, 2016.

[6] Felipe Leno Da Silva, Garrett Warnell, Anna Helena Reali Costa, and Peter Stone. Agents teaching agents: a survey on inter-agent transfer learning. *Autonomous Agents and Multi-Agent Systems*, 34(1):1–17, 2020.

[7] Gregory Dudek, Michael Jenkin, Evangelos Milios, and David Wilkes. A taxonomy for swarm robots. In *Proceedings of 1993 IEEE/RSJ International Conference on Intelligent Robots and Systems (IROS'93)*, volume 1, pages 441–447. IEEE, 1993.

[8] Lee Alan Dugatkin. *Cooperation among animals: an evolutionary perspective*. Oxford University Press on Demand, 1997.

[9] Stan Franklin and Art Graesser. Is it an agent, or just a program?: A taxonomy for autonomous agents. In *International Workshop on Agent Theories, Architectures, and Languages*, pages 21–35. Springer, 1996.

[10] Sravyasri Garapati and Kamalakar Karlapalem. Empirical evaluation of idle-time analysis driven improved decision making by always-on agents. In *2018 IEEE 42nd Annual Computer Software and Applications Conference (COMPSAC)*, volume 2, pages 141–146. IEEE, 2018.

[11] glaurung. glaurung. https://chessprogramming.wikispaces.com/Glaurung.

[12] Brian C Gunia, Long Wang, LI Huang, Jiunwen Wang, and J Keith Murnighan. Contemplation and conversation: Subtle influences on moral decision making. *Academy of Management Journal*, 55(1):13–33, 2012.

[13] Nicholas R Jennings and Michael Wooldridge. Applications of intelligent agents. In *Agent technology*, pages 3–28. Springer, 1998.

[14] Kenneth A. De Jong and Alan C. Schultz. Using experience-based learning in game playing, 1988.

[15] W. T. Katz and S. Pham. Experience-based learning experiments using go-moku. In *Conference Proceedings 1991 IEEE International Conference on Systems, Man, and Cybernetics*, pages 1405–1410 vol.2, Oct 1991.

[16] Bart Kosko. Neural networks and fuzzy systems: a dynamical systems approach to machine intelligence/book and disk. *Vol. 1Prentice hall*, 1992.

[17] Richard Kraut. *Aristotle on the human good*. Princeton University Press, 1991.

[18] lichess. lichess.org. https://en.lichess.org/.

[19] Magnus Ljungberg and Andrew Lucas. The oasis air traffic management system. 1992.

[20] Pattie Maes. Agents that reduce work and information overload. In *Readings in human–computer interaction*, pages 811–821. Elsevier, 1995.

[21] Maja J Mataric. Learning to behave socially. In *Third international conference on simulation of adaptive behavior*, volume 617, pages 453–462. Citeseer, 1994.

[22] Pragnesh Jay Modi, Manuela Veloso, Stephen F Smith, and Jean Oh. Cmradar: A personal assistant agent for calendar management. In *International Bi-Conference Workshop on Agent-Oriented Information Systems*, pages 169–181. Springer, 2004.

[23] Ndedi Monekosso and Paolo Remagnino. Phe-q: A pheromone based q-learning. In *Australian Joint Conference on Artificial Intelligence*, pages 345–355. Springer, 2001.

[24] Hyacinth S Nwana. Software agents: An overview. *The knowledge engineering review*, 11(3):205–244, 1996.

[25] Liviu Panait and Sean Luke. Cooperative multi-agent learning: The state of the art. *Autonomous agents and multi-agent systems*, 11(3):387–434, 2005.

[26] Charles David Chanel Reeve. *Action, Contemplation, and Happiness*. Harvard University Press, 2012.

[27] Bradley J Rhodes. The wearable remembrance agent: A system for augmented memory. *Personal Technologies*, 1(4):218–224, 1997.

[28] Mark B. Ring. Child: A first step towards continual learning. *Machine Learning*, 28(1):77–104, Jul 1997.

[29] Stuart Jonathan Russell, Peter Norvig, John F Canny, Jitendra M Malik, and Douglas D Edwards. *Artificial intelligence: a modern approach*, volume 2. Prentice hall Upper Saddle River, 2003.

[30] Jürgen Schmidhuber. Deep learning in neural networks: An overview. *Neural networks*, 61:85–117, 2015.

[31] Mahendra Sekaran and Sandip Sen. To help or not to help. In *Proceedings of the Seventeenth Annual Conference of the Cognitive Science Society*, pages 736–741, 1995.

[32] Sandip Sen. Reciprocity: a foundational principle for promoting cooperative behavior among self-interested agents. In *Proceedings of the Second International Conference on Multiagent Systems*, volume 315321, 1996.

[33] Sandip Sen and Mahendra Sekaran. Using reciprocity to adapt to others. In *International Joint Conference on Artificial Intelligence*, pages 206–217. Springer, 1995.

[34] Candace Sidner, Timothy Bickmore, Charles Rich, Barbara Barry, Lazlo Ring, Morteza Behrooz, and Mohammad Shayganfar. An always-on companion for isolated older adults. In *14th Annual SIGdial meeting on discourse and dialogue*, 2013.

[35] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354, 2017.

[36] Amazon Staff. Fulfilmentcenters. https://www.aboutamazon.co.uk/working-at-amazon/about-our-fulfilment-centres.

[37] stockfish. stockfish. https://stockfishchess.org/.

[38] Peter Stone and Manuela Veloso. Multiagent systems: A survey from a machine learning perspective. *Autonomous Robots*, 8(3):345–383, 2000.

[39] Richard S Sutton and Andrew G Barto. *Reinforcement learning: An introduction*. MIT press, 2018.

[40] Milind Tambe. Recursive agent and agent-group tracking in a real-time dynamic environment. In *ICMAS*, pages 368–375, 1995.

[41] Ming Tan. Multi-agent reinforcement learning: Independent vs. cooperative agents. In *Proceedings of the tenth international conference on machine learning*, pages 330–337, 1993.

[42] Wikipedia. Algebraic notation (chess) — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Algebraic\%20notation\%20(chess)&oldid=1031078019, 2021. [Online; accessed 22-July-2021].

[43] Wikipedia. Chess — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Chess&oldid=1034479278, 2021. [Online; accessed 22-July-2021].

[44] Wikipedia. Universal Chess Interface — Wikipedia, the free encyclopedia. http://en.wikipedia.org/w/index.php?title=Universal\%20Chess\%20Interface&oldid=1058786945, 2021. [Online; accessed 30-December-2021].

[45] Michael Wooldridge. *An introduction to multiagent systems*. John Wiley & Sons, 2009.

[46] Li D Xu. Case based reasoning. *IEEE potentials*, 13(5):10–13, 1994.