Optimizing Multi-Agent Coordination in Grid Exploration: Implementation of Advanced Attack Strategies

Teammates: Marc Soda
Matt Wong
Karthick Sivakumar
Professor: Jeff Heflin

Introduction

In the rapidly evolving field of multi-agent systems, efficient coordination and competition among agents play a crucial role in solving complex problems. The Multi-Agent Programming Contest (MAPC) serves as a testing ground for the latest techniques and strategies in agent coordination and competition. The 2022 MAPC scenario presents an intriguing grid world exploration challenge, wherein agents must work together to acquire blocks and assemble them into intricate patterns while competing against rival teams. This paper presents our team's approach to enhancing agent performance in this dynamic environment by designing and implementing an advanced AttackIntention strategy.

The objectives of this project are to develop a team of agents capable of efficiently exploring the grid world, acquiring blocks, and assembling them into specified patterns. To achieve these goals, we built upon the existing codebase of team MMD from the MAPC competition and introduced our custom AttackIntention strategy to improve agent collaboration and competitive edge. This paper provides a detailed analysis of the design, implementation, and integration of the AttackIntention strategy, along with the results of testing and evaluation in the competition environment.

Background

Contest Overview

The Multi-Agent Programming Contest (MAPC) is an annual event aimed at promoting research and development in the field of multi-agent systems. The contest presents a unique platform for researchers, developers, and students to test and compare their approaches to agent coordination, communication, and competition in a controlled environment. Each year, the contest organizers propose a new scenario that requires participants to tackle complex problems by designing and implementing innovative multi-agent strategies.

Grid World Exploration Specifics

Grid World Exploration involves teams of agents navigating a grid world, acquiring blocks, and assembling them into complex patterns. The grid world consists of various types of cells, such as obstacles, clear cells, and goal zones. Each team has a base in the grid world from which their agents start exploring the environment. The agents must cooperate within their team to efficiently explore the grid, gather information about block locations, and coordinate their actions to acquire and assemble the blocks into specified patterns.

In addition to intra-team collaboration, the scenario also involves inter-team competition. Multiple teams of agents share the same grid world, competing for the same blocks and goal zones. This competitive aspect introduces an additional layer of complexity, as agents must not only cooperate with their teammates but also outperform rival teams in acquiring and assembling the blocks.

Rules and Constraints

The 2022 MAPC imposes a set of rules and constraints that govern the agents' actions and interactions in the grid world. Some of the key rules include:

- Each agent can perform a limited number of actions per time step, such as moving, observing, and executing block manipulation actions.
- Agents can only observe and interact with their immediate surroundings, limiting their knowledge of the grid world.
- The grid world is subject to dynamic changes, such as the appearance of new blocks or changes in cell types.

These rules and constraints present numerous challenges for the participating teams, requiring them to develop efficient and robust strategies for agent coordination, communication, and competition.

Team MMD's Code

Agent Decision-Making Process

The decision-making process of an agent is managed through an instance of the Intention-Handler class, defined in the intentionHandler.py script. This handler serves to manage the agent's intentions, keeping them organized in a priority queue. The queue allows the agent to structure its actions based on the priority of the tasks at hand, ensuring that the most pressing intentions are addressed first. The insertIntention method is used to add new intentions into the priority queue, while the finishCurrentIntention method is used to mark the current intention as completed and remove it from the queue.

Initialization and Base Intention Setting

Upon initialization, an instance of the IntentionHandler class is assigned an ID and sets its default role to "Explorer". It also initializes a priority queue of intentions and sets the basic intentions for the agent, namely 'Explore', 'Update Map', and 'Idle', through the initializeBaseIntention method.

Intention Management

The management of intentions is central to the functionality of this class. The insertIntention method allows the handler to add new intentions into the queue based on their priority. The handler can then assess if its current intention is related to a task with the isCurrentIntentionRelatedToTask method. When an intention is completed, it is removed from the queue using the finishCurrentIntention method.

Local Intention Generation

The handler's generateOptions method allows the generation of local intentions based on the agent's observations. For instance, it can create an EscapeIntention if the agent finds itself in need to escape, thus providing a mechanism for the agent to respond to its environment dynamically.

Filtering Intention Options and Decision-Making

The filterOptions method is used to select the current intention from the priority queue. This method also handles special circumstances such as the need for the agent to escape. Depending on the nature of the current intention and the role of the agent, appropriate actions are taken to manage the transition.

Role Management and Task Abandonment

The setIntentionRole and getIntentionRole methods allow for setting and retrieving the agent's role, respectively. This functionality is vital for enabling the agent to adopt diverse roles, such as a coordinator, block provider, or single block provider. The abandonCurrentTask method provides the agent with the ability to abandon its current task if necessary.

Intention Coordinate Management

Lastly, the handler has the capability to manage the coordinates associated with all intentions via the updateIntentionCoordinatesByOffset and normalizeIntentionCoordinates methods. These methods ensure that the agent's intentions are always correctly aligned with the environment.

In summary, the IntentionHandler class offers a robust and adaptive decision-making framework for an agent, enabling it to effectively manage and prioritize its intentions, adapt to environmental changes, and carry out complex tasks in a multi-agent system.

Design and Implementation of AttackIntention

AttackIntention extends beyond the basic exploration function, imbuing the agent with the capability to proactively seek out and incapacitate opposing agents. In the absence of enemy agents within the vicinity, the agent defaults to the exploration function, mapping out and learning about unknown areas. The AttackIntention class is a complex construct, built from a series of interwoven methods that cooperate to form a comprehensive decision-making process. The following sections provide a detailed breakdown of these integral methods, shedding light on how they collectively contribute to the functionality of the AttackIntention class.

getPriority

This method returns a fixed priority value (6.0). This is used by the system to decide the order in which agents should act or which tasks should be prioritized. We choose the value 6.0 because we didn't want the agent to prioritize exploring and attacking over working and constructing, as construction is the only way to we earn points.

planNextAction

This method decides the agent's next action based on its current observation. It combines the attack and exploration intentions:

- First, it checks if there's an enemy agent that can be cleared. If there is, the agent finishes its task and returns a ClearAction to reduce the enemy's energy.
- If there's no enemy in range, the agent resorts to the exploration part of its intention. It checks the nature of the target area and decides to either travel to unknown areas, detach blocks if it's carrying more than allowed, or adopt a reserved role that it hasn't adopted yet.
- If none of these conditions are met, the agent starts looking for a new target, which could be a close unknown coordinate or a random far one if a close unknown one can't be found.

getClearableEnemy

This method supports the attack intention by finding enemy agents within shooting range. It first finds coordinates that contain enemy agents and are not in the goal zones. If multiple such coordinates exist, it chooses the closest one. If there's only one, it chooses it with a 70% probability.

checkFinished

This method checks if the agent has finished its task. It returns True if the total map count is 1 and the map is fully explored, indicating that the agent has completed its attack or exploration intention.

updateCoordinatesByOffset and normalizeCoordinates

These methods adjust the coordinates of the agent's intentions based on a given offset or normalize them. This helps in maintaining accurate tracking of the agent's position and its intentions.

explain

This method provides a human-readable description of the agent's current intention. If the agent is traveling, it indicates the coordinate it's exploring towards.

planRoleAdoptPlan

This method is part of the exploration intention. It plans for the agent to adopt a given role if it has any reserved roles that it hasn't adopted yet. It then returns the next action of this intention.

Testing and Evaluation

To test our implementation, we engaged in a series of strategic confrontations. Specifically, we pitched the base agent, MMD, against the second and third place winners from a recent competition, GOALdigger and FIT BUT. These agents provided a formidable benchmark due to their success and proven capabilities in the official competition. In these tests, our MMD agent consistently outperformed both GOALdigger and FIT BUT, indicating the effectiveness of the original teams implemented strategies.

During the initial phases of testing our attack implementation, we noticed an unexpected behavior related to the attack feature when we tested our agents against the original MMD agents. We observed that while the attack feature was indeed affecting the performance of the opposing agent, it was also significantly hindering the performance of our agent. After a thorough analysis, we identified the cause to be the improperly balanced priority settings. Our attack feature had been assigned a higher priority than the constructor, which was leading to a detrimental impact on the performance of our agent.

Once we adjusted these priority settings, correcting the imbalance between the attack feature, a subset of explore intention, and the constructor role, our agents began to consistently outperform the opposing teams. This was a consequence of the aggressive nature of our strategy, which prioritized conflict and disruption of the opponent's operations over the pursuit of exploration and monetary gains.

Despite this, the fact that our agents could consistently outperform the 2nd and 3rd rank agents was a significant achievement. It demonstrated that our agent was capable of adapting to the tactics of a diverse range of opponents and consistently finding ways to outmaneuver them. This gave us great confidence in our agent's performance in a competitive setting.

The following are the results of our agents against the original MMD agents:

Table 1: Priority 6.0

Test	MMD	KMM
1	350	560
2	410	560
3	430	410

Table 2: Priority 4.0

Test	MMD	KMM
1	580	310
2	470	120
3	530	480

Table 3: Priority 1.0

Test	MMD	KMM
1	250	590
2	420	430
3	260	490

Tournament Performance

During the tournament, our agents demonstrated excellent performance, underlining the effectiveness of our strategies and algorithms. In a series of highly competitive matches, we managed to consistently outperform other teams, marking six consecutive victories. This outstanding performance was largely due to our efficient algorithms, which enabled our agents to quickly adapt to the evolving circumstances on the field.

In the second game against Team Beta, we decided to experiment with our strategic priorities. Given our strong performance up to that point, we felt we were in a position to take some calculated risks. Thus, we increased the priority of our attack intention to a priority of 4.0 with the aim of preventing the other team from moving and securing a more dominant victory. While this approach had proven detrimental during our testing phase, we were willing to accept the potential downside for the chance of achieving a more emphatic victory.

However, as our testing had previously indicated, this adjustment led to an unforeseen issue. Increasing the priority of the attack intention resulted in our agent stalling and becoming unresponsive during the later stages of the competition. The agent's decision-making process was significantly hampered, leading to a reduction in overall performance. Despite this setback, we still managed to secure the victory, demonstrating the robustness of our agent's core strategies and the resilience of our team.

Following this incident, we decided to revert to prioritizing the constructor intention over the attacking intention. Our experiences in testing and during the tournament had shown us that a well-balanced approach, where construction activities were given priority, was key to securing consistent and dominant victories. The constructor intention is vital in establishing infrastructure and securing resources, which form the backbone of any successful strategy.

In retrospect, our performance during the tournament served to validate our strategic decisions and the effectiveness of our agent's algorithms. Despite some experimental adjustments leading to temporary setbacks, our agent demonstrated resilience and the ability to consistently outperform competitors.

Individual Contributions

Marc Soda

Marc played an integral role in the success of the project, with his contributions spanning from the initial stages to the final execution. He selected the MMD repository from the contenders of the 2022 Multi-Agent Programming Contest. This crucial decision was made based on the quality of the code and documentation, as well as the team's proficiency with the language used in the repository.

Marc's also setup the Guacamole environment and ensured seamless integration of the agent server framework with the initial (and future) state of the MMD repository.

Furthermore, Marc excelled in debugging the attack intention, assuring its flawless operation amidst the complex moving parts of the infrastructure. This task required a comprehensive understanding of the system and acute problem-solving skills.

Additionally, Marc conducted a series of benchmarks, pitting the original MMD repository against our final version. These tests were vital in assessing our progress and the effectiveness of our modifications. Marc's wide-ranging contributions, encompassing selection, setup, debugging, and performance testing, were pivotal to our project's success.

Matthew Wong

Matt's role in the project was indispensable. His primary task was understanding the intricate code from the original repository. With repository size of >8000 lines, this responsibility required a significant investment of time and analytical skills due to the shear size and complexity.

In addition to this, Matt collaborated with Marc to establish the server infrastructure, a key component for executing the agents for testing locally and on the guacamole server. This task called for technical expertise and effective team collaboration, aspects Matt adeptly handled.

Matt also took on the crucial responsibility of running the agents during the class tournament. His diligent management and insight of the agents were important to running the agents so that we have something to show during the competition. Matt's contributions undeniably played a significant role in the project's achievements.

Karthick Sivakumar

Karthick primarily focused on the conceptualization of the attack function. He devised strategic approaches and translated them into viable algorithmic solutions, significantly enhancing the agent's competitive edge.

Beyond his technical contributions, Karthick also took on the task of documenting the project. He adeptly wrote substantial portions of the paper, succinctly encapsulating the team's methodologies, results, and analyses. His encapsulation of complex concepts clearly and concisely made the paper both informative and accessible.

In summary, Karthick's contributions were pivotal to the project's success, from the conception of the attack function to the detailed documentation in the final paper.

Conclusion

In conclusion, our project successfully implemented an attack intention strategy in a multi-agent grid exploration environment, demonstrating considerable effectiveness. Our agents were proficient at both exploration and incapacitating opposing agents, resulting in an enhanced performance in competitive scenarios.

The true measure of our strategy's success was observed in the tournament, where our agents secured six consecutive victories. Despite a momentary setback when we experimented with an increased priority for the attack feature, our agents demonstrated resilience and adaptability, returning to a balanced approach prioritizing construction activities, which proved crucial for consistent success.

Overall, our agents' outstanding performance underscores the potential of advanced, well-balanced strategies in multi-agent systems, particularly in highly competitive environments.