information

Article

A Hexagon Sensor and A Layer-Based Conversion Method for Hexagon Clusters

Jun-Ho Kim and Hanul Sung *

Department of Game Design and Development, Sangmyung University, Seoul 03016, Republic of Korea; gyulari7160@gmail.com
* Correspondence: hanul.sung@smu.ac.kr

Abstract: In reinforcement learning (RL), precise observations are crucial for agents to learn the optimal policy from their environment. While Unity ML-Agents offers various sensor components for automatically adjusting the observations, it does not support hexagon clusters—a common feature in strategy games due to their advantageous geometric properties. As a result, users can attempt to utilize the existing sensors to observe hexagon clusters but encounter significant limitations. To address this issue, we propose a hexagon sensor and a layer-based conversion method that enable users to observe hexagon clusters with ease. By organizing the hexagon cells into structured layers, our approach ensures efficient handling of observation and spatial coherence. We provide flexible adaptation to varying observation sizes, which enables the creation of diverse strategic map designs. Our evaluations demonstrate that the hexagon sensor, combined with the layer-based conversion method, achieves a learning speed up to 1.4 times faster and yields up to twice the rewards compared to conventional sensors. Additionally, the inference performance is improved by up to 1.5 times, further validating the effectiveness of our approach.

Keywords: hexagon sensor; reinforcement learning; hexagon cluster; observation; conversion; layer-based conversion; strategic game map design

Citation: Kim, J.-H.; Sung, H. A Hexagon Sensor and A Layer-Based Conversion Method for Hexagon Clusters. *Information* **2024**, *15*, 747. https://doi.org/10.3390/info15120747

Academic Editors: Emilio Matricciani, Heming Jia and Zhigang Chu

Received: 24 October 2024
Revised: 19 November 2024
Accepted: 21 November 2024
Published: 22 November 2024

1. Introduction

Reinforcement learning is a branch of machine learning that is effective for finding optimal solutions in complex environments. An agent receives rewards from the environment based on its actions and solves problems by finding the optimal policy that maximizes the reward. The policy that determines the agent's actions is continuously updated through a neural network based on the rewards received for actions in the current state. Therefore, it is important for the agent to perceive information about the state through its observations to recognize the current state and state changes.

Unity ML-Agents is an open-source toolkit that provides reinforcement learning environments built on the Unity engine. It enables users to train intelligent agents for games and simulations, offering various sensor components to facilitate the agent's collection of observations. The sensor components automatically collect observations, eliminating the requirement for users to manually input observations into the neural network [1].

In many real-world problems, numerous complex elements are created or removed dynamically. Therefore, agents must observe environments where objects of various data types—such as scalar data, multidimensional lists, and image data—are continuously changing. For this reason, ML-Agents provides several sensors that support various shapes, types, and formats, enabling the agent to adapt to the specific requirements of diverse scenarios.

Among the various shapes used for the complex placement of cells, hexagon clusters have gained popularity due to their unique geometric properties. Compared to square grids, hexagon clusters offer more edges. The increased number of edges leads to more

neighboring cells, offering additional movement choices. If agents need to move to a certain cell from the current cell, many movement options diversify the paths between them. Thus, agents have to consider more complex strategies when they move. This characteristics make them popular in fields that require the identification of spatial relationships and strategic movements, such as war games [2–4], strategy games [5,6], strategy role-playing games (SRPGs), and spatial mapping [7,8]. In fact, games developed on square grids are improved by transitioning to hexagon clusters, enabling more strategic gameplay [9]. Especially since strategy games and SRPGs rely on strategic decision-making—such as selecting attack routes, avoiding rough terrain, and positioning units—rather than on players' skills, diverse strategies are essential for designing attractive gameplay. Therefore, agents need to ascertain the spatial features and the relationships of cells as observations to effectively learn the gameplay. However, not only are there no sensors specifically designed for hexagon clusters but the existing sensors also fail to conform with the unique information types required by hexagon clusters. Despite the significant impact sensors have on learning costs and performance, they are not supported, causing substantial challenges for researchers and developers.

Although there have been attempts to overcome these limitations with the existing sensors, it remains a significant difficulty. In particular, some sensors demand considerable computational power and are primarily designed for visual or collision-based data, which makes them excessive for map design. To create a strategic game, a variety of map shapes are essential. The number of empty cells varies with the shape of the map, resulting in fluctuations in the observation size. However, other sensors restrict their observations to fixed sizes, making them unsuitable for the development of diverse maps. Although an existing sensor supports variable observation sizes, it has difficulty identifying which cell's information is being collected due to the presence of empty cells. While the position of the cells within a hexagon cluster is crucial, using existing sensors to explicitly collect position data along with other observations leads to poor learning performance. Since the sensor does not collect empty observations, it is also impossible to deduce the position from the observation structure.

To overcome these limitations, we propose a hexagon sensor that facilitates the collection of observations for hexagonal clusters, along with a new layer-based conversion method that optimizes the sensor's performance. To verify the effectiveness of our hexagon sensor, we conducted experiments in which agents navigated a hexagon-cluster-based map. The agent employing our hexagon sensor and the new conversion method learned 1.4 times faster and achieved twice the reward value compared to the others. In the inference performance evaluation, our method demonstrated an up to 1.5 times higher performance than the competing approaches.

2. Related Works

2.1. Sensor Components of Unity ML-Agents

Many studies have utilized the existing sensor components offered by Unity ML-Agents to address real-world problems. Studies utilizing the RayPerceptionSensor have trained agents by observing the environment through collision detection between objects and rays that substitute the agent's field of view [10,11]. The RayPerceptionSensor enables the observation of the environment from the agent's perspective. However, in complex environments, it exhibits poor learning performance, and its dependence on visual information makes it difficult to collect non-visual data, such as values internally assigned to the objects. These limitations make it challenging to use the RayPerceptionSensor to observe hexagon-cluster-based maps. Other studies have used the BufferSensor, which outperforms the RayPerceptionSensor and enables the collection of non-visual data [12,13]. The buffer sensor supports variable observation collection. However, if objects have no observable values, the buffer sensor does not observe them. This results in the information used to infer the cell positions lacking information from empty cells. This presents challenges when observing hexagon-cluster-based maps.

2.2. Addressing Hexagon Cluster Information

Many studies have explored handling hexagon cluster data using various approaches, but limitations persist concerning the computational cost and grid conversion methods. Since utilizing hexagon clusters allows for a clearer representation of image data and enables complex interactions in simulations [14], studies have focused on converting input data into hexagonal formats for Convolutional Neural Networks (CNNs) or designing novel CNNs capable of directly processing hexagon cluster formats [15–17]. However, utilizing CNNs based on image data to observe hexagon clusters results in high computational costs [18]. Additionally, existing studies have used position coordinate conversion methods, such as the axial coordinate system or the double-width coordinate system [7,8], to observe hexagon clusters. Since these methods approximate the hexagon clusters using square grids, they face challenges in identifying the precise positions of the hexagon cells.

3. Background and Motivation

Observations are essential for an agent to learn a policy, as they provide information about changes in the environment's state. Without observations, an agent will struggle to learn a policy effectively, making it difficult to solve a problem.

Figure 1 illustrates the result of navigating to find the destination cell in a hexagon-cluster-based map without observations. The agent chooses a neighboring cell to move to based on its selected action. It receives positive or negative rewards based on the cell selected. If the agent reaches the destination cell more quickly, it receives a higher reward for achieving this goal. Without any observations of the cells, the agent learns to reach the destination solely by relying on the rewards it receives for its actions. To evaluate the learning performance, the learning results are analyzed using TensorBoard metrics [19]. The results show that the cumulative reward, representing the mean cumulative episode reward across all agents, did not improve. Similarly, the value estimate, reflecting the expected return when following the current policy, did not improve either. Furthermore, the value loss, representing the mean loss of the value function update, and the entropy, reflecting the randomness of the model's decisions, did not decrease. This indicates that the agent was unable to learn effectively in this environment.

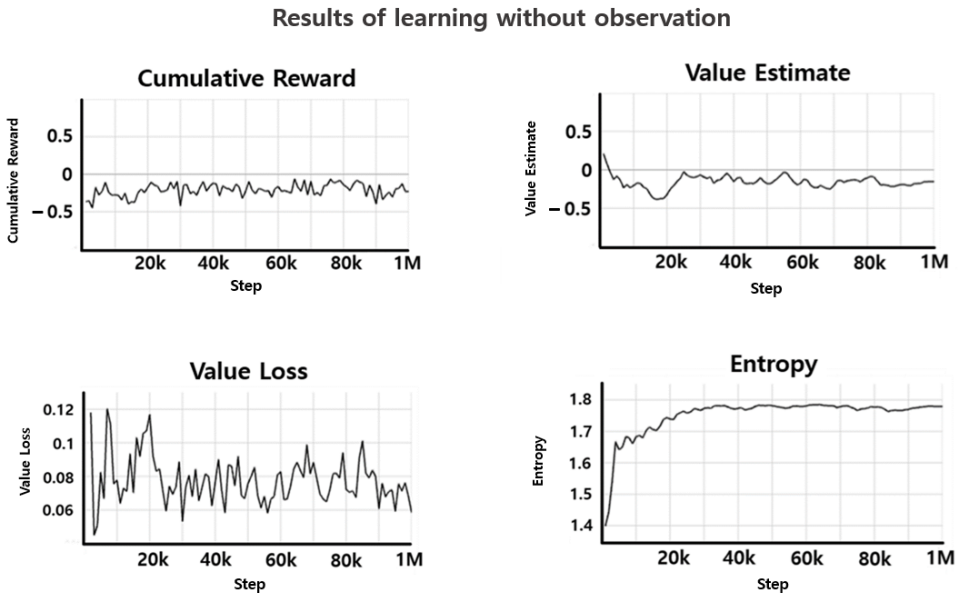

Figure 1. Graph of results of learning in a hexagon-cluster-based map without sensor components provided which collect observations.

Unity ML-Agents provides sensor components that enable the efficient collection of observations. However, the existing sensor components pose challenges, as they do not support the shapes and types of observations required for hexagon clusters. The choice of the

sensor components depends on the required shapes and types of observations [20]. When ML-Agents' built-in sensors are not directly suitable for the experimental environment, designing custom sensors may be necessary [21,22].

The vector sensor supports various data types via vectors, but it only allows fixed-length observations. To generate a hexagon-cluster-based map that allows for more diverse designs, empty cells that do not interact with the agent are added. Since the agent treats empty cells as non-existent, it does not collect observations for these cells. As the number of empty cells changes to create diverse maps for each episode, the observation size varies accordingly. Consequently, the vector sensor cannot be used in this context. Unity ML-Agents also provides sensors that use image data from Unity's camera objects, such as the Camera Sensor and the Render Texture Sensor. Additionally, the Grid Sensor divides the surrounding space into grids and collects observations of object collisions within the grid [23]. However, these sensors require significant computational resources and focus on collecting visual or collision-based data rather than the internal data on objects [24,25]. Another sensor that utilizes collision data is the RayPerceptionSensor, which collects observations based on ray collisions, as depicted in Figure 2. Similar to the grid sensor, it primarily focuses on visual data. Because of these overheads and characteristics, these sensors are unsuitable for hexagon-cluster-based maps. As an alternative, the buffer sensor can be used with hexagon-cluster-based maps. The buffer sensor writes values into a buffer and then uses them as observations. It supports variable-sized observations and requires fewer resources since it is not image-based.

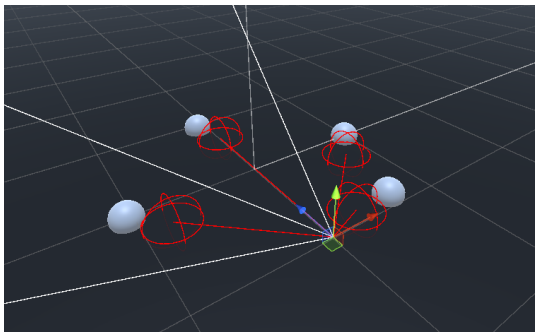

Figure 2. RayPerceptionSensor that utilizes collision data to collect observations.

However, the buffer sensor also has limitations in observing hexagon-cluster-based environments. The key information about each cell includes its internal value, channel, and position. Internal values provide data to help the agent achieve its goals more effectively. Channels represent the characteristics of the cells and are categorized by the common types of interactions between cells and the agent. For instance, a channel is classified based on whether it gives a reward, imposes a penalty, or changes an agent's attributes, such as position, velocity, or size. Position data are essential for the agent to identify the cell locations accurately, enabling it to understand the cell distribution, proximity, and distances within the cluster. For this reason, existing studies have assigned coordinates to cells by approximating the hexagon cluster as a grid using methods such as axial coordinate conversion, offset coordinate conversion, and double-width coordinate conversion [26,27].

Figure 3 illustrates an example of approximation using axial coordinate conversion. In this conversion, similar to a rectangular grid, a horizontal x-axis and a vertical y-axis are employed, but the x-axis is tilted diagonally. In a rectangular grid, movement to neighboring cells is achieved by adjusting either x or y by ±1. However, a hexagonal grid, with six directions, requires simultaneous changes to both the x and y coordinates for movements. As shown in Figure 2, moving to neighboring cells in the upper-right or lower-left directions requires simultaneous adjustments to both the x and y coordinates, such as (+1, −1) or (−1, +1).

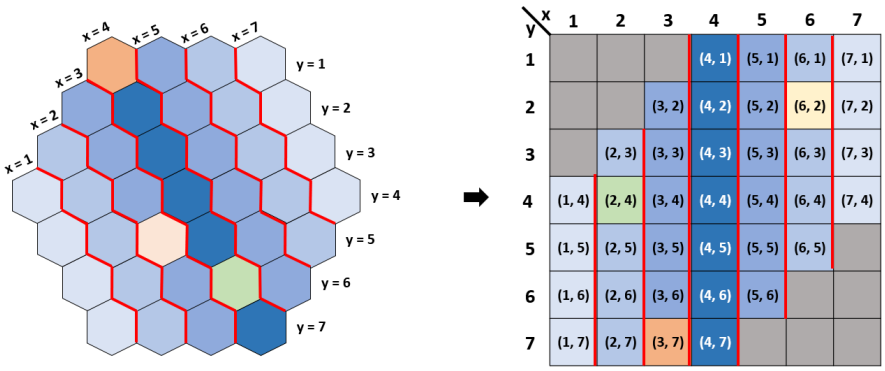

Figure 3. Axial coordinate conversion that approximates the hexagon cluster on a grid based on a diagonally tilted horizontal x-axis and a vertical y-axis.

The converted coordinates are traversed sequentially in row-major order and flattened into a one-dimensional index. By utilizing this position index, the cell's position—the key information—is inferred without explicitly writing the position data into the observation, which reduces the observation size and enables more efficient learning.

In a hexagon cluster without empty cells, as shown in Figure 4, all the cells are observed sequentially according to their position indexes so that the observation array index corresponds to the cell's position index. Therefore, the observation index allows the cell's position to be inferred from the information written at that index in the observation array. For instance, since the cell corresponding to position index 6 is always collected as the seventh observation, the information written at observation index 6 in the observation array corresponds to the cell at (2, 5), which is the coordinate before the position index was flattened.

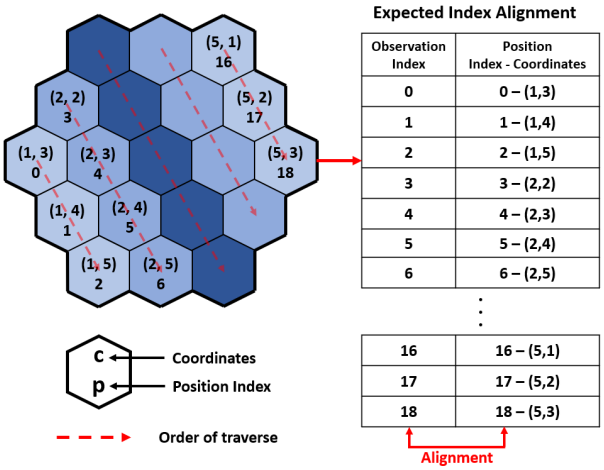

Figure 4. Structure for sequentially observing cells in a hexagon cluster without empty cells based on their position index.

However, in a hexagon cluster with empty cells, the buffer sensor does not collect observations for empty cells. When the sensor sequentially observes the cells by position index, if an empty cell is encountered, the position index increases to observe the next cell, while the observation index remains the same, as no observation has been collected. This leads to a misalignment between the observation index and the position index. In the example of the area with empty cells highlighted in red in Figure 5, the position indexes of the cells in this area are 1, 4, and 6, rather than 0, 1, and 2, because of the misalignment caused by the empty cells. Due to misalignment, it becomes impossible to infer the cell positions directly from the observation index.

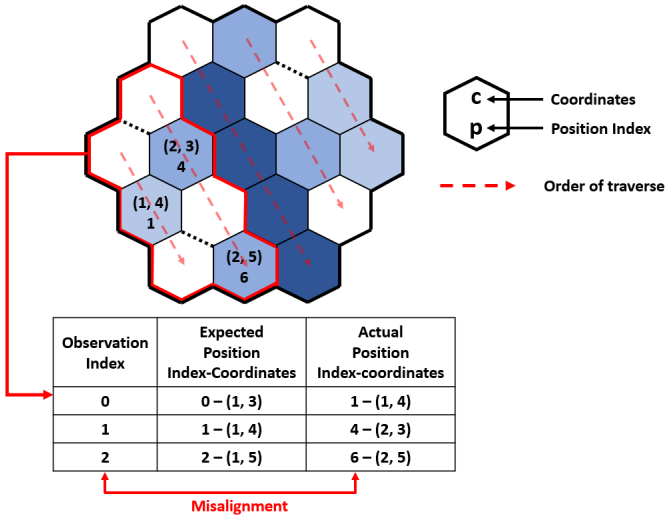

Figure 5. Structure of sequentially observing cells in a hexagon cluster with empty cells, excluding the empty cells, based on their position index.

To address this limitation, one approach is to directly include the coordinates of the cells in the observations collected by the buffer sensor. In this case, the observation size increases, as coordinate data are included. Additionally, each observation includes both coordinate data and internal values. This leads to ambiguity about what each value in the observation represents, negatively impacting the agent's learning performance.

Figure 6 demonstrates the observation structure of the buffer sensor when the coordinates are explicitly included in the observation. In this observation structure, the buffer sensor observes that the cell highlighted in red, as depicted in Figure 6, is positioned at coordinates (5, 5), belongs to channel 2, and has an internal value of 3. The coordinates are separated into x and y values during observation collection. Internal values are written only in the space corresponding to the channel to which the cell belongs, within the multiple allocated spaces for each channel. The buffer sensor collects observations only for non-empty cells [13].

Figure 7 shows the experimental results using this buffer sensor. In the experiment, the buffer sensor transforms the observation table shown in Figure 5 into a two-dimensional array, where the dimensions represent the maximum number of observable objects and the observation size. During this conversion, the buffer sensor sets the values of unassigned channels and the padding from empty cells to 0. This experiment was conducted in the same environment used for the experiment in Figure 1. In this experiment, although the maximum reward the agent receives is set to 1, both the cumulative reward and the value estimate converge at approximately 0.5. The entropy steadily decreases, indicating that the agent relies on policy-based actions. An increase in value loss indicates that the value function has been updated incorrectly. The results demonstrate that the agent, designed to integrate multiple data types into a single observation, failed to find the optimal solution, resulting in degraded learning performance.

For these reasons, we propose a novel sensor and conversion method for the hexagon cluster.

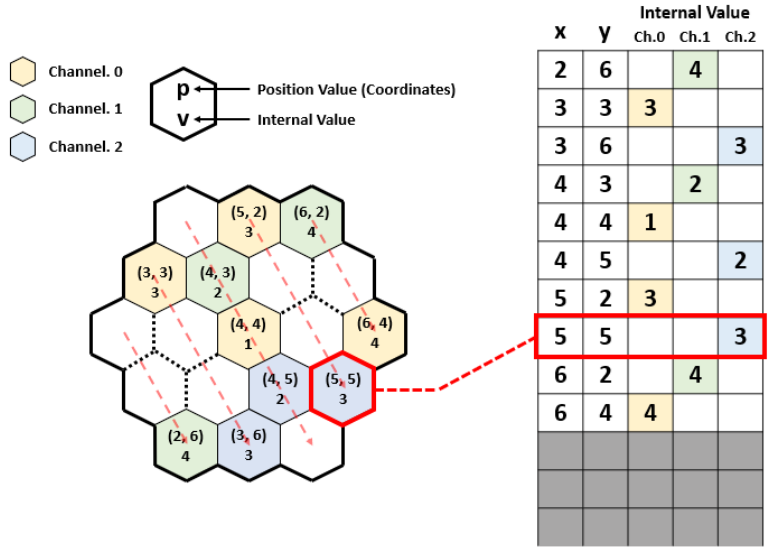

Figure 6. Overview of the observation collected by the buffer sensor. The observation consists of the x, y values of the coordinates converted based on axial coordinate conversion and the internal values assigned to each cell.

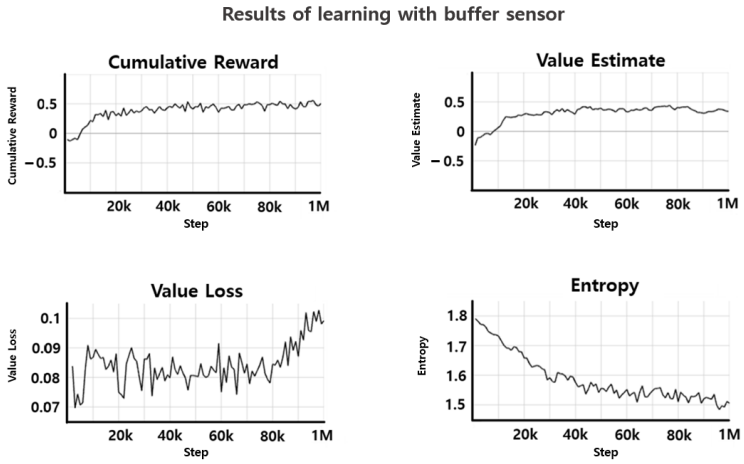

Figure 7. Graph of results of learning in a hexagon-cluster-based map with the buffer sensor. The failure to improve in the learning metrics, except for entropy, indicates that the agent has been trained based on an incorrect policy.

4. Design

4.1. The Hexagon Sensor

This section presents the newly developed hexagon sensor, designed to overcome the limitations encountered when observing a hexagon cluster using the buffer sensor.

As seen in Figure 8, the buffer sensor collects observations for each cell. Unlike the buffer sensor, the hexagon sensor, as depicted in Figure 9, gathers information from all the cells in each channel into an array to collect the observations in one go. Due to structural differences, the hexagon sensor collects observations from empty cells, unlike the buffer sensor. Including empty cells in the observations ensures that the collected structure aligns with the actual cell composition in the hexagon cluster. In contrast, the buffer sensor excludes empty cells from its observations, discarding any information about them. In terms of the input format, since the hexagon sensor gathers all the cells into an array rather than on a cell-by-cell basis, a transposed version of the buffer sensor's input format is used.

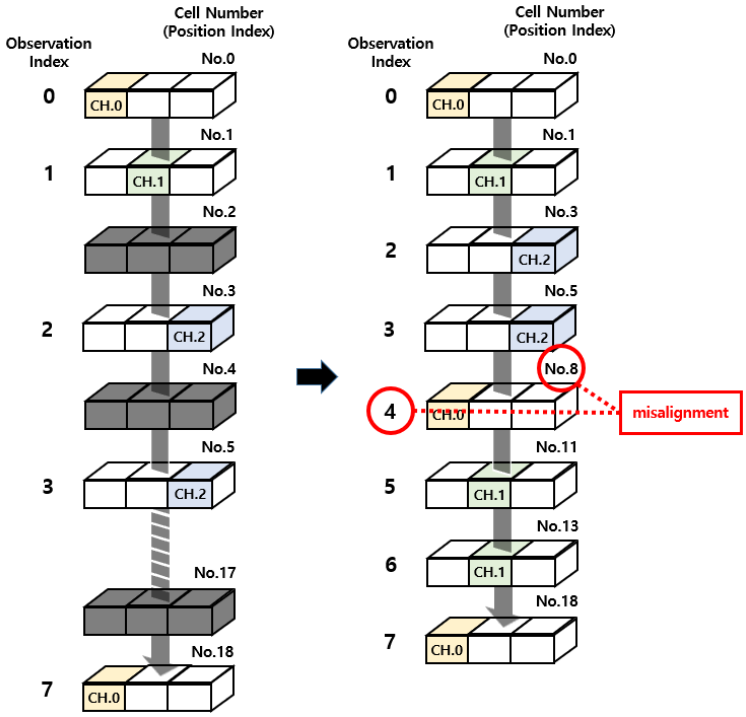

Figure 8. Observation structure of the buffer sensor; excluding empty cells from the observation results in the misalignment between the position index and the observation index.

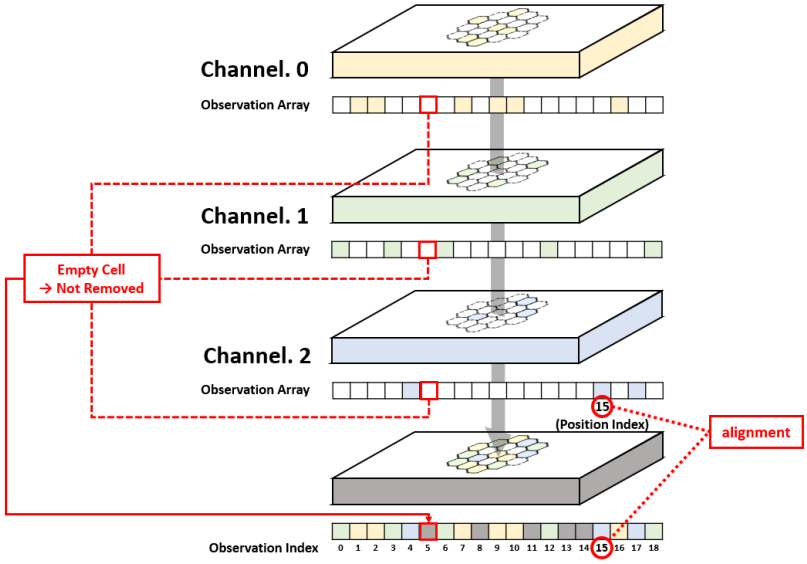

Figure 9. Observation structure of the hexagon sensor, which collects information by gathering the values written by all cells into an array, resulting in alignment between the position index and the observation index.

The hexagon sensor traverses all cells in the order determined by the position index from the conversion method, writing the internal value to the corresponding index in the observation array for the cell's channel. If no value is written to a channel array index, the corresponding cell does not belong to that channel. Thus, if there is an index with no value written in the observation arrays across all channels, this indicates an empty cell. Since the observations are collected into an array, the empty cell is retained in the observation; the index for which no value is written due to the empty cell is represented as an indication that "this cell does not belong to the channel". For instance, although index 5,

as illustrated in Figure 9, is an empty cell for which no values are written in any channel, it remains in the observation array. Furthermore, the internal value of the cell at position index 15 is written at observation index 15 in the observation array, despite the presence of several empty cells preceding it. This observation structure ensures that the position index aligns with the observation index.

Figure 10 illustrates how a hexagon sensor collects the cell information from a hexagon cluster into an observation array using this structure. The internal value of the red-highlighted cell is written at index 15 in the observation array for channel 2, corresponding to the cell's position index. Since each cell's internal value is written to the observation index corresponding to its position index, each observation index in the array directly corresponds to a specific cell. If the value for a cell is written to observation index 8, the cell corresponds to position index 8. Similarly, if the value for another cell is written to observation index 17, the cell corresponds to position index 17. Thus, the hexagon sensor allows the cell's position to be inferred from the observation index, without explicitly including the position information.

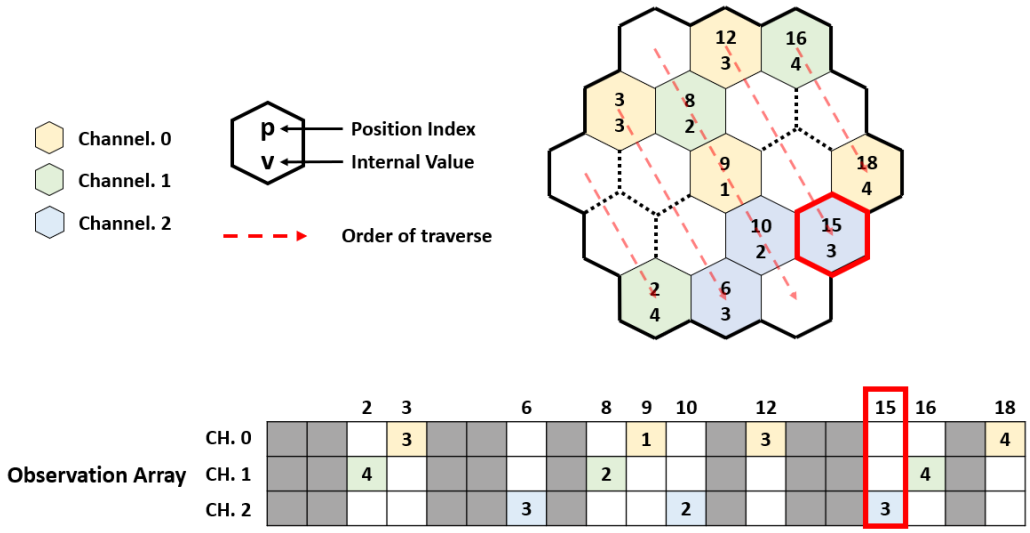

Figure 10. Overview of the hexagon sensor observing the cells in a hexagon cluster, with position indexes assigned based on axial coordinate conversion.

4.2. Layer-Based Conversion

Existing conversion methods represent the cell positions by approximating the hexagon cluster as a grid and converting them into coordinates. Due to this approximation, unnecessary coordinates that extend beyond the boundaries of the hexagonal cluster are created, as seen in the gray area in Figure 3 in Section 3. Therefore, the hexagon sensor makes additional efforts to exclude the coordinates in the gray area from the observations. Additionally, since the conversion method flattens the coordinates into indexes in row-major order, the hexagon sensor cannot identify the relationships between a cell and the others in the cluster from the index. The index fails to identify adjacent cells and provides no information about the distance from the central cell.

To address these issues, we propose a new conversion method for the hexagon cluster optimized for the hexagon sensor.

The new conversion method, referred to as layer-based conversion, represents the hexagon cluster as a structure of multiple layers that expand outward from the center. Figure 11 illustrates the structure of the hexagon cluster converted using layer-based conversion. In this method, a layer is a group of cells that are equidistant from the center. The cells are indexed sequentially from the inner (near-center) layers to the outer (farther) layers, and within the same layer, they are indexed in the clockwise direction. Due to its geometric characteristics, each new layer added to the hexagon cluster increases the

number of cells by 6. Consequently, the position index, determined by the cell's layer and its order within the layer, is calculated using the equation below.

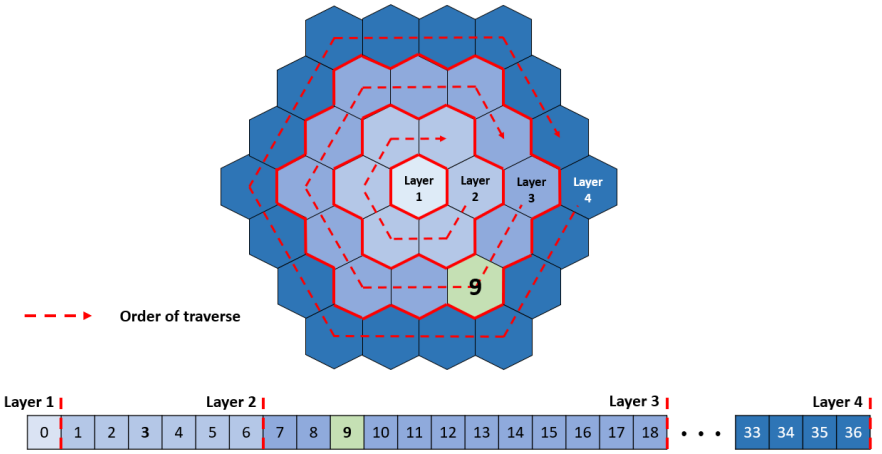

Figure 11. A conversion method that divides the hexagon cluster into multiple layers from the center and assigns position indexes to the cells according to their layers.

$$\text{position index} = 3(l-1)(l-2) + k \quad (l \text{ is layer}, k \text{ is order within layer})$$

For instance, the cell highlighted in green belongs to layer 3 and is indexed at 9, following the cells in the lower layers and the preceding cells in layer 3. The hexagon sensor completes the indexing process after converting the coordinates for the last cell in the highest layer into a position index. Therefore, unlike grid conversion methods, the hexagon sensor does not access coordinates outside the hexagonal cluster's boundary. As a result, there is no need to filter out these coordinates for observations. Moreover, the layer-based conversion method helps in understanding the relationships between cells through their indexes. Indexes for lower layers are closer to the center, and cells within the same layer are assigned to adjacent observation indexes. Therefore, an agent located in the central cell recognizes that it is more likely to interact with cells with lower-layer indexes and that cells with consecutive indexes are at similar distances from it.

Figure 12 illustrates the improved observation structure and features of the hexagon sensor with the layer-based conversion method. The cells highlighted in red are assigned index numbers based on their distance from the central cell. The cell at index 8 is farther from the center than the cell at index 2, and the cell at index 11 is farther than both. Cells highlighted in green belong to the same layer and are adjacent, with consecutive index numbers from 14 to 17. In contrast, in Figure 10, using the grid conversion method, there is no correlation between the distance from the center and the relationship between the indexes for the cells that occupy the same positions as the red cells. Although the cells in the same position as the green cells are adjacent, they have index numbers 3, 7, 12, and 16, making it impossible to infer the relationships between them.

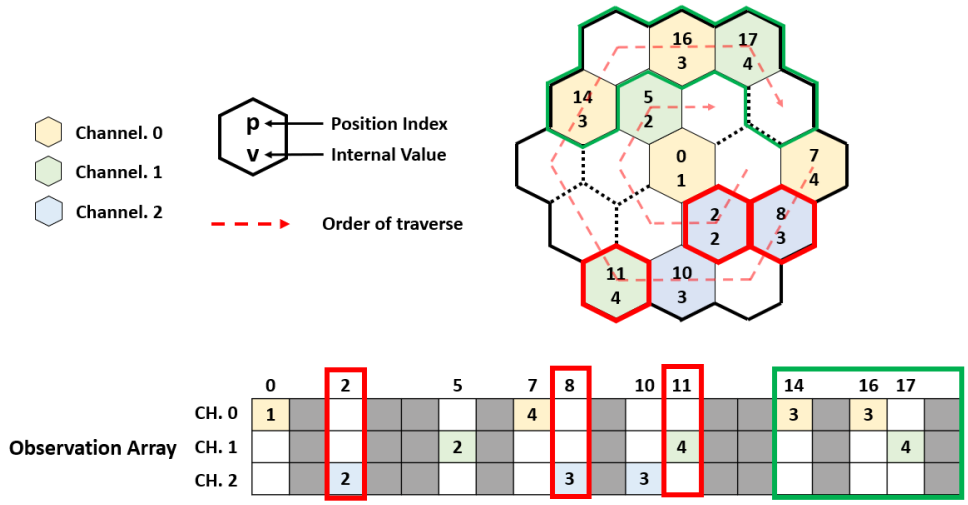

Figure 12. Overview of the hexagon sensor observing the cells in a hexagon cluster, with position indexes assigned based on layer-based conversion.

5. The Experimental Environment

To prove the advantages of the hexagon sensor, we designed an environment in which the agent learns within a hexagon-cluster-based map.

5.1. The Map Design Based on the Hexagon Cluster

The map used in the experiment is a hexagon cluster, symmetrical in all directions from the center. This hexagon cluster consists of five layers, totaling a maximum of 61 cells. In this experiment, each channel represents three characteristics for cells: positive, negative, or destination. Each channel determines the type of reward the agent receives when it moves to the cell. For example, in the gameplay, a positive cell represents terrain in which items are located or that provides advantages to the agent, while a negative cell represents terrain in which enemies are located or that imposes penalties on the agent. The agent receives a +reward for a positive cell, a -reward for a negative cell, and a significant +reward for the destination cell, which ends the episode. The agent's goal in this experiment is to find the optimal path for reaching the destination cell by progressively moving from its current position to adjacent cells. Also, each cell is assigned an internal value. This internal value is the distance to the destination, defined as the minimum number of cells the agent must pass through to reach the destination cell, rather than the straight-line distance. The agent identifies which cells will bring it closer to the destination cell by observing the distance values.

5.2. Creation of the Cells

The cells within the map are created in accordance with the following rules. One cell is randomly assigned as the destination cell when the map is initially generated with the exception of the central cell, which is always created. All the other cells, except for the central and destination cells, have a 50% chance of being created or remaining empty. If a cell is created, it has a 50% probability of being assigned into either the positive or negative channel. Also, the map is generated to ensure that the path between all cells is connected. Cells are defined as belonging to the same set if a path exists between them. If two sets are not connected due to the presence of empty cells, the agent may be unable to reach the destination cell if the agent and the destination cell belong to different sets. In this case, the agent is prevented from achieving its goal, regardless of the actions it selects. To avoid creating isolated sets of cells, the creation of a cell that would be isolated without adjacent cells is canceled. Once the map is fully generated, it is verified to ensure all the cells belong to the same set, guaranteeing the map's connectivity. If the map is divided into two or more disconnected sets, the map is discarded, and a new one is generated.

Figure 13 illustrates the final configuration of the map composed of cells created based on the rules described in Section 5.2.

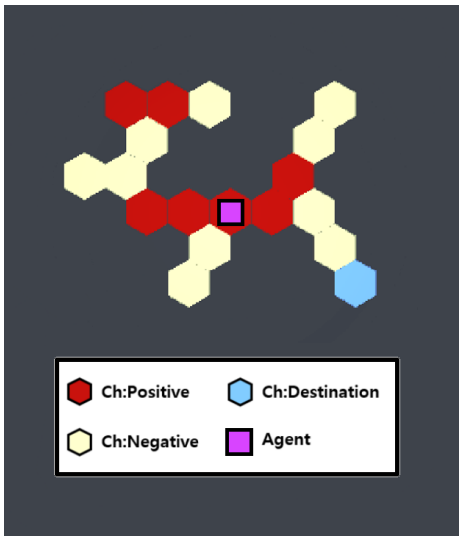

Figure 13. An example of a generated map. The map consists of 13 positive cells, 7 negative cells, and 1 destination cell, all of which are connected without any disconnections.

5.3. Sensor Range

The sensors used in the experiment are set to observe three layers surrounding the agent. Thus, the sensors collect observations for 19 cells, and the observed cells change as the agent moves. The position index of a cell is an absolute position index within the entire hexagon cluster, not a relative position index based on the agent. Thus, the position index of each cell remains fixed even as the agent moves. Figure 14 illustrates an example of how the cells to be observed vary as the agent moves within the hexagon-cluster-based map.

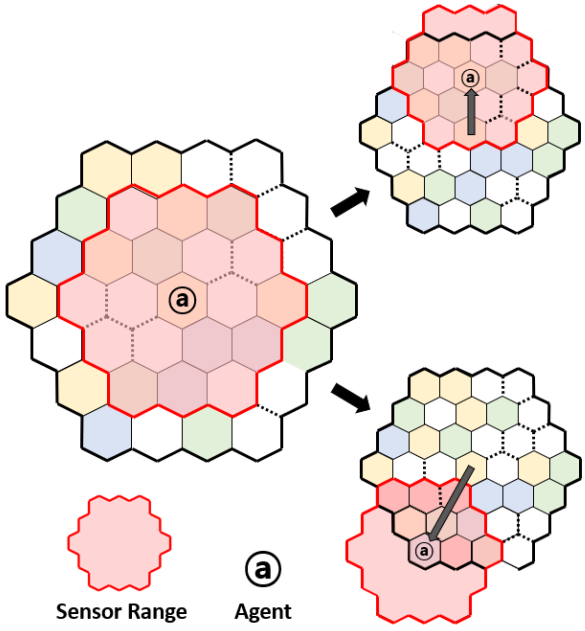

Figure 14. The observable range of the hexagon sensor covering three layers surrounding the agent used in the experiment.

5.4. The Agent

At the beginning of each episode, the agent starts at the central cell. As shown in Figure 15, the agent selects one of six possible movement directions. Based on the selected action, the agent moves to one of the six cells adjacent to its current cell.

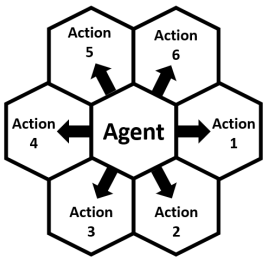

Figure 15. This figure demonstrates the directions of movement from the current cell based on the actions selected by the agent.

The agent is restricted from moving to empty cells or beyond the boundaries of the hexagon cluster. The agent receives a −reward and triggers an error if it selects an invalid move. When the agent accumulates 50 errors in a single episode, the episode is immediately terminated, and a new episode begins. Additionally, if the agent fails to reach the destination cell before the current Unity frame count exceeds 1000 frames, the episode is also considered a failure, and a new episode begins.

5.5. Rewards

The rewards are determined based on Table 1 and include a movement reward, a goal achievement reward, an error penalty, a continuous penalty, and an excessive frame count penalty. The continuous penalty, applied at every Unity frame, encourages the agent to keep moving toward its goal. The excessive frame count penalty is applied if its goal is not achieved within the predefined number of frames, preventing the agent from becoming stuck in a single episode.

Table 1. Reward table.

Types of Reward	Reward Value
Move to a positive cell	+0.001
Move to a negative cell	−0.0005
Reaches the destination cell	$1 - (0.001 * \text{Timer})$ [1]
Error penalty	−0.02
Continuous penalty	−0.0005
Excessive frame count penalty	−1

[1] "Timer" refers to the Unity frame count from the beginning of the episode to the current step.

5.6. Hyper-Parameters and the Environment's Spec

The hyper-parameters for this learning are configured as shown in Figure 16. The maximum number of steps is set to 1 million, and the Proximal Policy Optimization (PPO) algorithm is used for learning [28]. The specifications of the hardware and software versions used for the experiment are provided in Table 2.

Table 2. Experiment specification.

Hardware and Software	Specification and Version
CPU	Ryzen 5 5600X (6 cores)
GPU	NVIDIA GeForce RTX 3060ti
Memory	Samsung RAM DDR5-4800 (32 GB)
Storage	Samsung PM981a (1 TB)
ML-Agents release version	Release 21
ML-Agents Unity package	3.0.0
ML-Agents Python package	1.0.0
Python	3.10.13
PyTorch	1.13.1

```
behaviors:
  HexTest:
    trainer_type: ppo
    hyperparameters:
      batch_size: 128
      buffer_size: 10240
      learning_rate: 0.0003
      beta: 0.01
      epsilon: 0.2
      lambd: 0.95
      num_epoch: 3
      learning_rate_schedule: linear
    network_settings:
      normalize: false
      hidden_units: 128
      num_layers: 2
      vis_encode_type: simple
    reward_signals:
      extrinsic:
        gamma: 0.99
        strength: 1.0
    keep_checkpoints: 10
    max_steps: 1000000
    time_horizon: 256
    summary_freq: 10000
```

Figure 16. Hyper-parameter settings in the configuration file for learning based on Unity ML-Agents.

6. Performance Results

We conducted experiments to evaluate the performance of the hexagon sensor in comparison with the buffer sensor. The agents were categorized into three types based on the sensor and the conversion method:

- Agent A: Converts the position of cells into coordinates using the axial coordinate conversion method and then collects observations for which the coordinates are explicitly written using the buffer sensor.
- Agent B: Converts the position of the cells into coordinates using the axial coordinate conversion method but collects observations using the hexagon sensor, without explicitly writing the coordinates.
- Agent C : Collects observations using the hexagon sensor with the layer-based conversion method, without converting the positions of the cells into coordinates.

Learning performance is evaluated by comparing the changes in rewards and the learning data obtained during the learning process for each agent. To evaluate the inference performance, we embedded the learned model into the agent and repeated the experiment. We compared the rate of achieving the goal after running 100,000 episodes.

6.1. Learning Performance Evaluation

Figure 17 demonstrates the changes in the rewards and learning data obtained during the learning process of each agent.

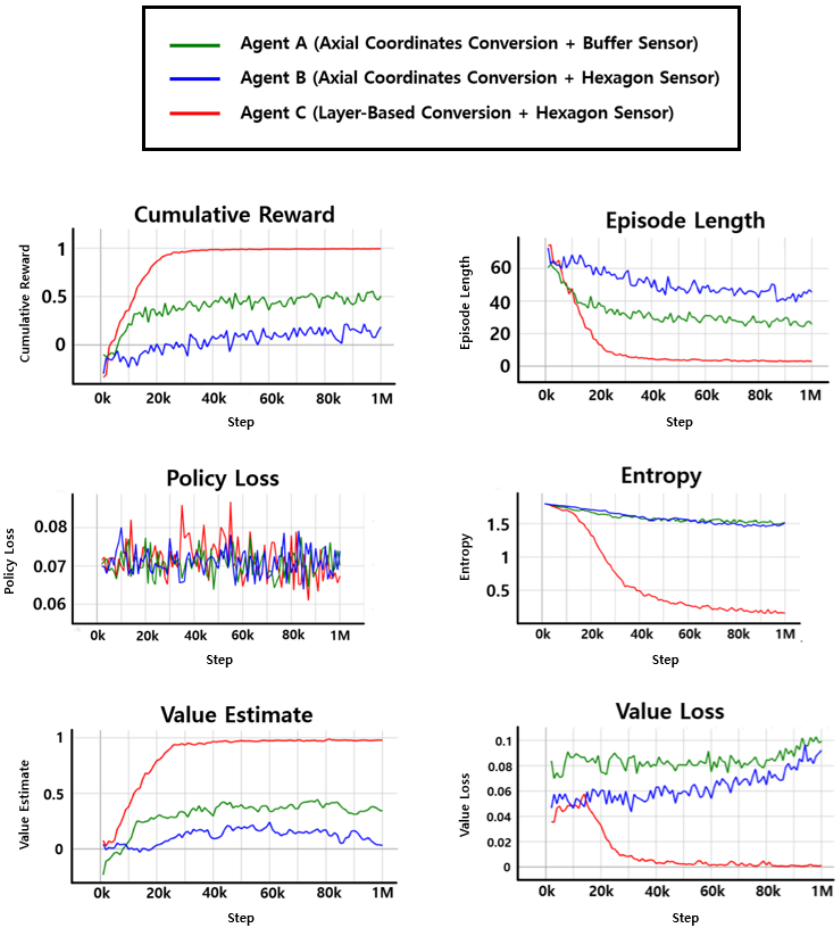

Figure 17. Graph of results of learning for each agent regarding the reward, episode, and policy. The agent using the hexagon sensor and the layer-based conversion method proposed in this paper is represented as Agent C and marked in red on the graph.

First, the cumulative reward graph, which represents the mean cumulative episode reward of the agents, illustrates how the rewards change throughout the learning process. According to the graph of results, the cumulative reward of all agents steadily increases until it stabilizes at a certain value. The convergence of the cumulative rewards reflects that the policy is no longer being updated to achieve higher rewards, suggesting the agent has completed its learning. A higher cumulative reward reflects that the agent received more rewards during the episode, suggesting that it reached the destination cell more frequently while receiving fewer negative rewards. At the conclusion of the learning, Agent C received a reward of 0.9942, which is close to the maximum possible reward in the experimental environment, while Agent A received a reward of 0.5061, and Agent B received an even lower reward of 0.1863. Agent C, which used the hexagon sensor and layer-based conversion, performed more successfully in achieving its goal compared to the other agents.

Next, the episode length, representing the average length of the episodes completed by the agents, indicates how quickly the agents achieved their goal. The results show that the episode length of all of the agents consistently decreased; however, the pace of this decrease and the value at the end of learning varied. In this result, Agent C demonstrated a faster reduction in episode length compared to the other agents, achieving its goal 8 times faster than Agent A and 15 times faster than Agent B.

The actions taken by the agent during learning are evaluated through policy loss and entropy. Policy loss refers to the value of the loss function used during policy updates; a lower value indicates that the agent effectively learns a policy to choose actions that lead to the expected returns. Entropy represents the randomness of the agent's actions; as this value decreases, the agent increasingly relies on policy-based actions rather than randomly selecting from a variety of actions. While Agent C's policy loss decreases, Agent A shows little change, and Agent B's policy loss increases instead. Additionally, although the entropy values for all of the agents decrease, Agent C's entropy decreases more rapidly and significantly than that of the others. So, Agent C has learned the policy more effectively, and it has selected actions based on learned policy more frequently compared to the other agents.

Since the agent updates its policy to choose actions that yield the highest return from the current state, the accuracy with which the state-value function—calculating the expected return of a state—is predicted is also used to evaluate learning performance. An agent that fails to accurately predict the state-value function will learn an incorrect policy. The accuracy of the agent's state-value function predictions is assessed using the value estimate and the value loss. The value estimate represents the estimated value of the state-value function under the current policy, while the value loss indicates the difference between the estimated state-value function and the actual return. A lower value loss signifies that the value estimate closely approximates the actual return. For successful learning, the value estimate should increase while the value loss decreases. As shown in the results, similar to the cumulative reward, Agent C's value estimate increases and approaches the maximum reward, whereas that of the other agents either stops increasing at a lower value or even decreases. As for the value loss, only Agent C's value decreases, while the other agents' value loss increases. These results indicate that Agent C predicts the state-value function more accurately, enabling more precise policy updates compared to those of the other agents.

Furthermore, it is not only important how successfully an agent learns but also the speed at which the agent completes learning, as this is a significant factor from a cost perspective. Figure 18 illustrates the results of measuring the learning time for each agent to complete 1 million steps. Agent A, using the buffer sensor, took 58.55 min, whereas Agents B and C, using the hexagon sensor, took 27.66 min and 34.23 min, respectively. This shows that using the hexagon sensor enables learning to be completed 1.71 to 2.11 times faster, depending on the conversion method.

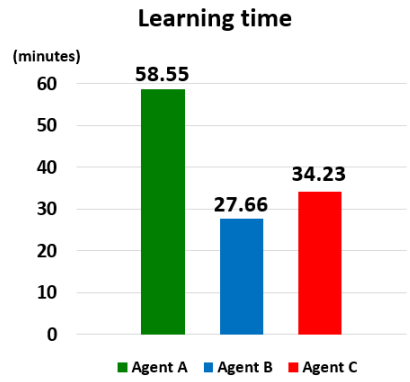

Figure 18. Learning time required by each agent to reach the predetermined max steps.

Based on a comprehensive comparison of the learning performance, Agent C, which utilizes the layer-based conversion method and the hexagon sensor, clearly outperforms the other agents in both its effectiveness and learning speed, offering significant cost advantages.

6.2. Inference Performance Evaluation

To verify whether high learning performance leads to a superior problem-solving ability, we evaluated the inference performance. The learned model was embedded into the agents, and experiments were conducted for 100,000 episodes. According to the results, Agent A reached the destination cell in 82.841% of the 100,000 episodes, and Agent B did in 66.078%. In contrast, Agent C reached the destination cell in 99.983% of the episodes, approximately 1.2 times and 1.5 times more often than the rates of Agents A and B, respectively. These goal achievement rates demonstrate that Agent C outperforms both Agent A and Agent B in terms of inference performance.

7. Conclusions

In this paper, we introduced a hexagon sensor along with a layer-based conversion method to address the limitations of the existing sensors in collecting observations from hexagon clusters. Our approach effectively supports variable-sized observations in environments where the number of empty cells dynamically changes due to diverse map designs, by supporting variable-sized observations and introducing a new observation structure that collects information from all cells. This structure ensures the alignment of the cell position indexes with the observation indexes, enabling the inference of cells' positions using only their indexes while reducing the observation size. In the evaluations, our approach achieved an up to 2.11 times better learning performance and an up to 1.5 times higher inference performance compared to that of the other approaches.

Author Contributions: Software, J.-H.K.; Supervision, H.S. All authors have read and agreed to the published version of the manuscript.

Funding: This research was funded by a 2021 Research Grant from Sangmyung University (2023-A000-0002).

Institutional Review Board Statement: Not applicable.

Informed Consent Statement: Not applicable.

Data Availability Statement: The original contributions presented in the study are included in the article, further inquiries can be directed to the corresponding author.

Conflicts of Interest: The authors declare no conflict of interest.

References

1. Technologies, U. Unity ML-Agents Toolkit - Sensors. Available online: https://github.com/Unity-Technologies/ml-agents/tree/develop/com.unity.ml-agents/Runtime/Sensors (accessed on 10 October 2024).
2. Wang, H.; Tang, H.; Hao, J.; Hao, X.; Fu, Y.; Ma, Y. Large Scale Deep Reinforcement Learning in War-games. In Proceedings of the 2020 IEEE International Conference on Bioinformatics and Biomedicine, BIBM 2020, Seoul, Republic of Korea, 16–19 December 2020; pp. 1693–1699. https://doi.org/10.1109/BIBM49941.2020.9313387.
3. Cannon, C.T.; Goericke, S. Using Convolution Neural Networks to Develop Robust Combat Behaviors Through Reinforcement Learning. Ph.D. Thesis, Naval Postgraduate School, Monterey, CA, USA, 2021.
4. Roland, E.; Wisthoff, B.; Kelleher, E. *Hex Size in JTLS*; US ROLANDS & ASSOCIATES Corporation: Monterey, CA, USA, 2002.
5. Kooij, G.V.D. Actor-Critic Catan: Reinforcement Learning in High-Strategic Environments. Bachelor's Thesis, Utrecht University, Utrecht, The Netherlands, 2020.
6. Blixt, R.; Ye, A. Reinforcement learning AI to Hive, 2013. Available online: https://www.diva-portal.org/smash/record.jsf?pid=diva2:668701 (accessed on 10 October 2024).
7. Apuroop, K.G.S.; Le, A.V.; Elara, M.R.; Sheu, B.J. Reinforcement learning-based complete area coverage path planning for a modified htrihex robot. *Sensors* **2021**, *21*, 1067. https://doi.org/10.3390/s21041067.
8. Zhu, Y.; Wang, Z.; Chen, C.; Dong, D. Rule-Based Reinforcement Learning for Efficient Robot Navigation with Space Reduction. *IEEE/ASME Trans. Mechatron.* **2022**, *27*, 846–857. https://doi.org/10.1109/TMECH.2021.3072675.
9. Smulders, B.G.J.P. Optimizing Minesweeper and Its Hexagonal Variant with Deep Reinforcement Learning. Bachelor's Thesis, Tilburg University, Tilburg, The Netherlands, 2023.
10. Saetre, S.M. Laying The Foundation For an Artificial Intelligence-Powered Extendable Digital Twin Framework For Autonomous Sea Vessels. Master's Thesis, NTNU, Trondheim, Norway, 2022.

11. Nämerforslund, T. Machine Learning Adversaries in Video Games Using reinforcement learning in the Unity Engine to Create Compelling Enemy Characters, 2021. Available online: https://www.diva-portal.org/smash/record.jsf?pid=diva2:1583775 (accessed on 10 October 2024).
12. Persson, H. Deep Reinforcement Learning for Multi-Agent Path Planning in 2D Cost Map Environments using Unity Machine Learning Agents toolkit, 2024. Available online: https://www.diva-portal.org/smash/record.jsf?pid=diva2:1867292 (accessed on 10 October 2024).
13. Elmaraghy, A.; Ruttico, P.; Restelli, M.; Montali, J.; Causone, F. Towards reviving the Master Builder: Autonomous design and construction using Deep Reinforcement Learning. Master's Thesis, Politecnico di Milano, Milano, Italy, 2022.
14. Birch, C.P.; Oom, S.P.; Beecham, J.A. Rectangular and hexagonal grids used for observation, experiment and simulation in ecology. *Ecol. Model.* **2007**, *206*, 347–359. https://doi.org/10.1016/j.ecolmodel.2007.03.041.
15. Jacquemont, M.; Antiga, L.; Vuillaume, T.; Silvestri, G.; Benoit, A.; Lambert, P.; Maurin, G. Indexed Operations for Non-rectangular Lattices Applied to Convolutional Neural Networks. In Proceedings of the 14th International Conference on Computer Vision Theory and Applications VISAPP, Prague, Czech Republic, 25–27 February 2019.
16. Luo, J.; Zhang, W.; Su, J.; Xiang, F. Hexagonal convolutional neural networks for hexagonal grids. *IEEE Access* **2019**, *7*, 142738–142749. https://doi.org/10.1109/ACCESS.2019.2944766.
17. Shilon, I.; Kraus, M.; Büchele, M.; Egberts, K.; Fischer, T.; Holch, T.L.; Lohse, T.; Schwanke, U.; Steppa, C.; Funk, S. Application of deep learning methods to analysis of imaging atmospheric Cherenkov telescopes data. *Astropart. Phys.* **2019**, *105*, 44–53. https://doi.org/10.1016/j.astropartphys.2018.10.003.
18. Verma, A.; Meenpal, T.; Acharya, B. Computational Cost Reduction of Convolution Neural Networks by Insignificant Filter Removal. *Sci. Technol.* **2022**, *25*, 150–165.
19. Technologies, U. Unity ML-Agents Toolkit - Using Tensorboard. Available online: https://github.com/Unity-Technologies/ml-agents/blob/develop/docs/Using-Tensorboard.md/ (accessed on 27 September 2024).
20. Rubak, M.; Göschka, F.H.D.P.D.M.D.D.K.M. Imitation Learning with the Unity Machine Learning Agents Toolkit. Master's Thesis, University of Applied Sciences FH Campus Wein, Wien, Austria, 2021.
21. Andersson, P. Future-proofing Video Game Agents with Reinforced Learning and Unity ML-Agents, 2021. Available online: https://www.diva-portal.org/smash/record.jsf?pid=diva2:1605238 (accessed on 10 October 2024).
22. mbaske. Grid Sensors for Unity ML-Agents. Available online: https://github.com/mbaske/grid-sensor (accessed on 10 October 2024).
23. Steward, J. Procedural Content Generation: Generating Procedural Game Content Using Machine Learning. Ph.D. Thesis, California State University, Northridge, Northridge, CA, USA, 2022.
24. Juliani, A.; Berges, V.P.; Teng, E.; Cohen, A.; Harper, J.; Elion, C.; Goy, C.; Gao, Y.; Henry, H.; Mattar, M.; et al. Unity: A General Platform for Intelligent Agents. *arXiv* **2018**, arXiv:1809.02627.
25. Tervo, A. Effects of Curriculum Learning on Maze Exploring DRL Agent Using Unity ML-Agents. Master's Thesis, University of Turku, Turku, Finland, 2022.
26. Hoogeboom, E.; Peters, J.W.T.; Cohen, T.S.; Welling, M. HexaConv. *arXiv* **2018**, arXiv:1803.02108.
27. Patel, A. Hexagonal Grids, 2013. Available online: https://www.redblobgames.com/grids/hexagons/ (accessed on 27 September 2024).
28. Schulman, J.; Wolski, F.; Dhariwal, P.; Radford, A.; Klimov, O. Proximal Policy Optimization Algorithms. *arXiv* **2017**, arXiv:1707.06347.