Multi-Agent Bomberman Game

Which type of intelligent agent thrives in a competitive environment

Francisco Guedes
MEIC-A
Instituto Superior Técnico
Lisboa Portugal
francisco.guedes@tecnico.ulisboa.pt

Miguel Gonçalves
MEIC-A
Instituto Superior Técnico
Lisboa Portugal
migueldgons@gmail.com

Paulo Alexandre
MEIC-A
Instituto Superior Técnico
Lisboa Portugal
paulo.n.alexandre@tecnico.ulisboa.pt

ABSTRACT

In order to evaluate which type of intelligent agent would thrive in a competitive setting, we decided to adapt a famous arcade game called *Bomberman* where we will be matching different types of agents against each other. With our approach, we aim to compare the behavioral performance of different agent types using a set of empirical metrics to be able to understand which one is the most suitable for this task.

1 Introduction

Our motivation for this project is to be able to empirically compare different agent types in an *accessible*, *deterministic*, *discrete*, *dynamic* and *episodic* environment where the agents are competing against each other.

To achieve our motivation, we decided to adapt the arcade game *Bomberman* to assess which agent type (reactive, deliberative, reflex and a random decision based) better suits this environment.

Bomberman is a strategy game where four players fight each other with the objective of eliminating the other players with bombs.

Our implementation of the *Bomberman* game will be similar to the original game besides the fact that we will not have power-ups.

In the end, we hope to be able to understand which agent type excels in this scenario and to assess this we will use a set of metrics for comparison like the percentage the games won or the average survival time.

2 Background

Four players are placed in a maze-like grid where the starting point for each one is a different corner.

Each player can place a bomb in their given position and this bomb will explode. The bomb explosion occurs after a specific amount of time and its range is the same across all directions (up, right, down, left). The bomb does self-damage which means it can possibly eliminate the player that placed it.

The grid has two different types of blocks. One type is an indestructible block that is placed in a checkboard-like pattern across the whole grid. The other type of block is destructible and spawns randomly at the start of each match, these blocks can be only destroyed when hit by the bomb explosion.

The original game also has powerups that spawn randomly across the match, however we will not go too much into detail since in our approach we will not be implementing these.

The game ends whenever there is only a single player alive, or when the remaining players are eliminated at the same time, in this case the game ends in a draw. To prevent an infinite game, after a certain amount of time, the grid starts to slowly close by removing the outer boundaries reducing the total map size.

Figure 1: Original Bomberman match example

3 Environment

Our implementation of the *Bomberman* game will be similar to the original game besides the fact that we will not have *powerups*, since it would have a negative impact in the performance evaluation of the agents. This could possibly be biased towards a specific agent because of the randomness that the power-up spawn would introduce and consequently corrupting our data.

3.1 Properties

This environment is *accessible* because the agents will be able to obtain a complete, accurate and up-to-date data about the environment's state. It is also *deterministic* and *discrete* because any action selected from the action space has a specific guaranteed effect and, as a board game, the agents will have a finite number of possible actions and percepts.

At last, the environment also is *dynamic* and *episodic* since all the agents act at the same time and one's actions have impact on the perceptions of all other agents.

3.2 Architecture

The environment is a grid-like board with 11 cells height and 17 cells width.

Upon generating a new grid for a game, each cell has a probability of 0.3 of being a **destructible block**. When placing the destructible blocks, the agents' starting cell is taken in consideration so that they always have a possible play to get out.

After 30 seconds from the game start, the board will start to shrink by removing the outer margin. The following shrinkages will happen 10 seconds apart from each other.

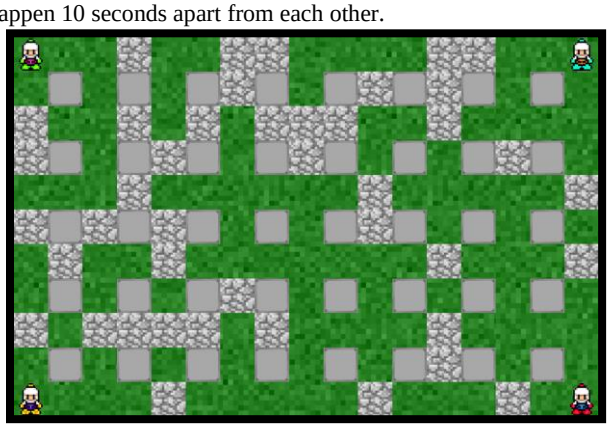

Figure 2: Sample game environment

4 Agents

Our implementation will be able to support four agents per game. The agents that will be put to test will be: a purely random agent, that will be used as a baseline for comparisons; a reactive agent; a deliberative agent and a reflex agent. All these agents will share the *mobility* property because they will be able to move freely within the environment. Besides the random agent, the other agents will also share the *autonomy* property since they all will be trying to achieve their goal – win the game.

With the existence of a purely random agent in our environment, we will be able to certify that the behavioral performance of the other agents is above the threshold given by this agent's empirical results.

Architecture. The agent action space is as follows: move in four different directions – move up, down, left, and right; place a bomb in its current position; and do nothing. Each agent has its own history that is only used for logging purposes. All the agents share the same think time with a default value of 0.75 seconds. This constant time corresponds to the time that the agent has to wait until it starts deciding its next action.

Movement Heuristic. Every intelligent agent – reactive, deliberative, and reflex – share a heuristic in order to move closer to the center of the board so that they do not get eliminated by the board border. This heuristic is only applied when the agent is able to move in multiple different directions. The heuristic computes the *Euclidean Distance* from the simulated position for each possible move to the central cell, if two or more moves share the same distance to the center, the agent chooses randomly.

Before settling with the *Euclidean Distance*, we experimented with the *Manhattan Distance*, but we observed that in some instances it would not be able to distinguish correctly between two paths, giving the same distance value to both of them, even though one was closer to the center cell.

4.1 Bomb

Each agent carries its own bomb. However, every bomb has the same properties except their explosion color. The bombs have an explosion radius of 2, this radius does not include the cell where the bomb is placed in. The bomb explodes after 7 seconds. The bomb has two different states depending on if it is currently placed in the board by its owner (an agent) – "placed" – or it is currently in its owner "backpack".

Whenever the agent places a bomb, the countdown timer for the explosion starts. Once it explodes, the explosion will dissipate after 1 second.

If the bomb explosion comes into contact with a **destructible block**, it will destroy that block and it will stop exploding in that direction. However, if the bomb explosion comes into contact with an agent, it will eliminate that **agent** and it will continue to explode in that respective direction.

Once the agent places a bomb in its position, it is able to move away from it to any direction. However, it is not able to move through it once it leaves that cell. As seen in the figure below:

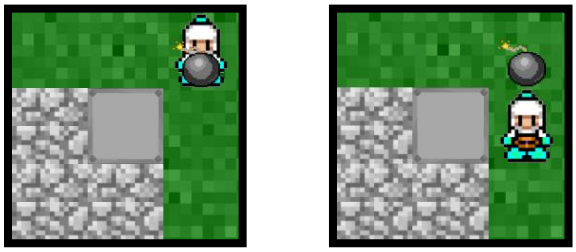

Figure 3: Bomb placement and movement

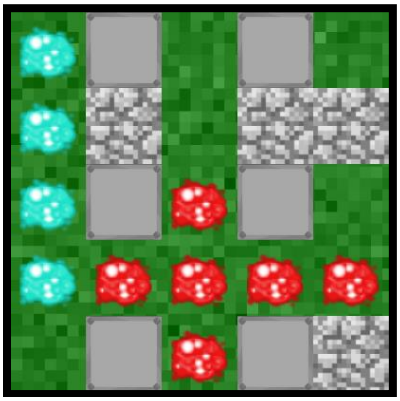

Figure 4: Two different bomb explosions

4.2 Random Agent

This agent is the simplest agent, it picks a random action from its action space. After computing this action, it waits *think time* before deciding its next action.

Figure 5: Random Agent

4.3 Reactive Agent

We chose a reactive agent because this type of agent is one that is equipped with a simple processing unit that can perceive and quickly react to changes in the environment. Since this is a competitive environment with quick changes, we expect that this type of agent will be able to thrive.

The reactive agent decision process is as follows: firstly, the agent looks if there is a bomb nearby by looking from its position to bomb radius whilst taking in consideration if the bomb in the vicinity is blocked by a block or not. If the agent is not in danger, it will look if there is an enemy agent nearby in the same way that it did for the bomb. If there is no enemy agent nearby it will check if there is a destructible block in an adjacent cell. If its sensors do not pick up anything the agent will check which moves are possible and decide one using the heuristic mentioned in **Movement Heuristic**.

React to Bomb. If the bomb is in the same cell as the agent – this only happens right after the agent places it – the agent will check which directions are free to move and decide based on its **movement heuristic**. If the bomb is placed in the same column as the agent the agent will prioritize horizontal movements, either left or right depending on if those cells are free or not. However, if the horizontal movement is impossible, the agent will check if it is possible to go in the opposite direction of the bomb and if it is, the agent will go in that direction. Otherwise, the agent does not move. This same thought process happens when the bomb is placed in the same row as the agent.

React to Agent. If the agent detected an unblocked enemy agent in the vicinity, and it currently has a bomb in its backpack, it places it. Otherwise, it does not react to the enemy agent.

React to Destructible. If the agent detected a destructible block in a cell right next to it, and it currently has a bomb in its backpack, it places it. Otherwise, it does not react to the destructible block.

Figure 6: Reactive Agent

4.4 Deliberative Agent

The deliberative agent contains an explicitly represented symbolic model of the world (board), which are its beliefs, and makes decisions via *symbolic reasoning* using the notions of desires and intentions. We hypothesize that this agent will have a weak performance because of its BDI (beliefs, desires, intentions) architecture which is characterized by a *think-and-plan* process. Due to this, it might not be able to match the speed at which the environment dynamically changes.

This agent starts the decision process by always updating its beliefs. After this, the agent will check if it already has a plan, if its intention has not already succeeded or if the intention is still possible. If so, the agent will select the first action in its current plan and verify if that action is sound. If the action is sound, the agent will execute it. Otherwise, the agent will use its reactive component to react to its current state. If none of the previous conditions are met, the agent will deliberate and build a plan accordingly. If no plan was possible to be generated, the agent will use its reactive component to react to its current state.

Deliberation. At this step, the agent has three possible desires: if there is a bomb nearby, it has the desire to escape from that bomb; if there is an enemy agent nearby, it has the desire to try to eliminate it; if there is a destructible block nearby, it has the desire to destroy it.

The deliberation process has a priority order for the agent's desires. Firstly, if the agent has the desire to escape from a bomb, its intention will be to escape from that bomb. If not, and if the agent has an unblocked enemy agent nearby, while having a bomb in its backpack, its intention will be to place a bomb to try to eliminate that enemy. If not, and if the agent detected a destructible block nearby and it has a bomb in its backpack, its intention will be to destroy that block.

Contrary to the reactive agent, the deliberative *look for enemy agent* and *look for destructible block* is more in-depth.

Deliberative Search. Meanwhile the reactive agent simply looks in its vicinity to check for danger or enemies, the deliberative agent actively searches for them in a wider range based on its beliefs. We achieved this by using a *Breadth-First Search* (BFS) so that the agent is able to find the closest enemy or destructible block to it. In order to slightly reduce computation load, we defined a *search threshold* for depth of the path. When searching for enemy agents, this *threshold* is 10, which means that the biggest path to an enemy agent is 10. When searching for destructible blocks, this *threshold* is 5.

Build Plan. Based on the intention that the agent settled in its deliberation, it will now build a plan in order to accomplish it. If the agent is not able to build a plan for its intention, it will use its **Reactive Component** in order to react to its current state.

Escape. When the agent is generating its plan, it will start by checking if moving to a different row or column immediately saves it from the bomb explosion or not. If it does, the plan will simply be that particular move. If not, the agent will start to compute possible paths in order to escape. This path generation is done similarly to the **Deliberative Search**, but the *threshold* for a possible path is the bomb diameter. If a path is found, the agent saves its action sequence as its plan.

Place Bomb. When the agent is generating its plan, it will start by checking if placing a bomb in its current position either eliminates an enemy or destroys a destructible block – depending on its intention. If it does, the plan will simply be that action. If not, the agent will start to compute possible paths in order to be able to eliminate or destroy the target. This path generation is done

similarly to the **Deliberative Search** with a *search threshold* of 10. If a path is found, the agent saves its action sequence as its plan.

Reactive Component. The deliberative reactive component behaves just like the **Reactive Agent**. However, after deciding on a *reactive action*, the deliberative agent verifies if this action is sound. If it is not, the agent does nothing. This implies that the reactive component shares the same **Movement Heuristic** as seen in the reactive agent.

Succeed Intention. In order to verify if the intention succeed or not, the agent checks two different things. If its intention were to try to eliminate an enemy or destroy a block, it checks if the bomb is already placed. If not, it means that the intention still has not succeeded. If the intention is to escape from bomb, it will look for a bomb in its vicinity, and if there is no bomb nearby it means that the intention of escaping from a bomb succeeded.

Impossible Intention. An intention becomes impossible when it is no longer achievable. In order to verify if the *destroy block* intention is still achievable, the agent checks its beliefs to see if the block still exists in the same position as when it deliberated. If the block no longer exists, it means that the intention is impossible. To check if the *try to eliminate enemy agent* intention is still possible, the agent verifies if placing a bomb in the deliberated position still eliminates an enemy or not.

Figure 7: Deliberative Agent

4.5 Reflex Agent

Our last choice was the reflex agent, which is similar to the reactive agent in the sense that both act accordingly to the current state of the environment. However, the difference is that the reflex agent has the ability to look to the consequences of its actions. We theorize that this additional behavior will provide additional insight into the game flow, therefore making a strong candidate for best behavior.

The reflex agent action decision process is similar to the reactive component of the deliberative agent.

The agent starts by looking for a bomb in the vicinity, if there is no bomb, it will look for an enemy agent nearby. If there is no enemy agent nearby, it will look for a destructible block and, if there is no destructible block right next to it, it will do a random *safe move* taking in consideration the **Movement Heuristic** as explained in section **Agents**.

The difference between this agent and the **Reactive Agent** is how it reacts to its current state. Both *react to agent* and *react to destructible* are the same process as in the **Reactive Agent**. However, it differs in how this agent reacts to a nearby bomb.

Bomb Reflex. This corresponds to the **React to Bomb** in section **4.3**, however the difference is: while the **Reactive Agent** simply reacts to its current state, this agent will also look to the consequence of its reaction. In order to achieve this, for each possible move in its current state, the agent will check if that action is able to get it to safety in future possible states. If this process verifies that multiple different moves are *safe moves*, the agent will use its **Movement Heuristic** to determine which move will be the most suited.

Figure 8: Reflex Agent

5 Comparative Analysis

In order to have more thorough comparison between the agents' behaviors we decided to test them with two different approaches. One of the approaches is just the normal game scenario that was explained up until this point. The other approach has a modification to the game environment where no **destructible blocks** spawn in, which means that the environment only has the checkerboard-like pattern with indestructible blocks.

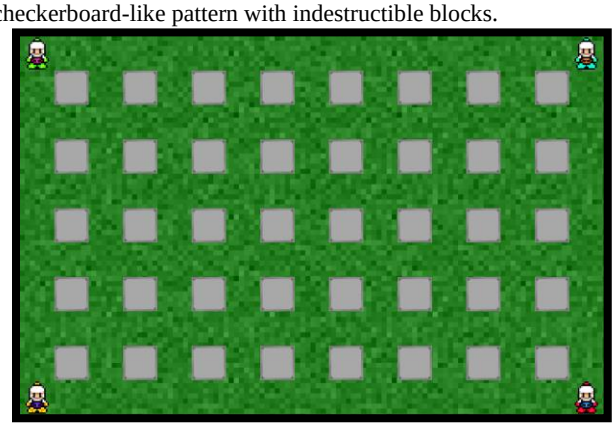

Figure 9: Checkboard environment

5.1 Normal Game Environment

For this scenario, we are going to the following metrics across all 4 agents:

- Percentage of game outcomes.
- Percentage of agent eliminations due to game border reduction.
- Percentage of agent eliminations due to their own bomb.
- Average number of blocks destroyed per game.
- Average number of actions per second.
- Average survival time per game in seconds.
- Average number of enemies eliminated per game.

The values for each of these metrics were obtained after running 9942 instances of our game implementation.

5.1.1 Percentage of game outcomes

As we predicted, the reflex agent was the best, winning 58.6% of games. In second place, came the deliberative agent with a 22.36% of win rate. To our surprise, the reactive agent win rate was much lower than the other intelligent agents, with a win rate of 4.2% which might be caused by its high self-elimination rate as we will explain shortly.

The percentage of draws was 14.6% which was much higher than what we initially expected. We theorize that this was caused mainly by the fact that the agents do not possess a sensor for the border shrinkage timer. Which implies that the agent is not aware if the border is about to collapse or not, this leads to many agents' eliminations to the border in the latter stages of the game.

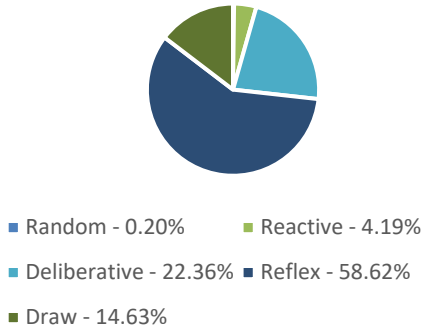

Figure 10: Percentage of won games for each agent.

5.1.2 Percentage of agent eliminations due to game border reduction

Even with our **Movement Heuristic** to try to diminish the deaths to border, this was still a fairly common cause of death.

As we can see in the pie chart that follows, the deliberative agent had a much higher percentage than the other agents. We believe that the main cause of this is how the deliberative agent computes its next action which make it fairly slower than other intelligent agents. Another thing that could possibly be causing this is how committed the agent is to its intention (commitment strategy) which would make it pursue its desire without taking in consideration the border, because the **Movement Heuristic** is only applied when the agent has no intention and its simply reacting to the environment.

The reflex agent also has a considerable percentage of deaths to the border, however we believe that is mostly from scenarios when the reflex agent is fighting another agent in a late stage of the game, where the board is already severely reduced.

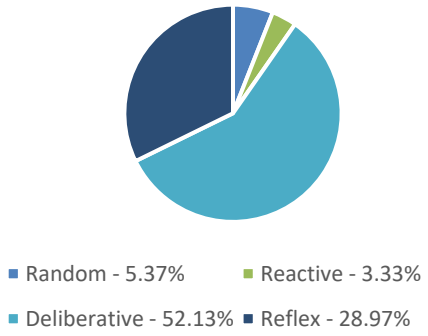

Figure 11: Percentage of agent eliminations to border reduction

5.1.3 Percentage of agent eliminations due to their own bomb

As we expected, the random agent has the highest percentage with a value of 89.900%, followed by the reactive agent with a percentage of 84.106%.

We believe that reactive agent high percentage is due to whenever it decides to place a bomb, if there are more than one free cell, it chooses the one that would place it closer to the center of the board. However, this can lead to scenarios where the agent places a bomb, and then gets itself stuck between blocks.

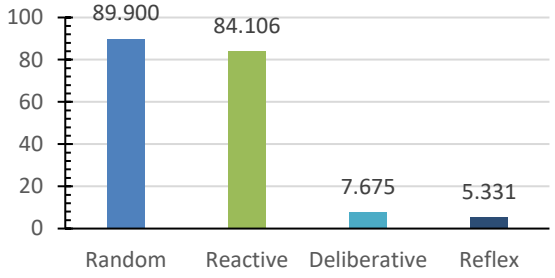

Figure 12: Percentage of agent deaths due to their own bomb

5.1.4 Average number of blocks destroyed per game

These values were in the range of expected values, with the reflex agent having the highest number of blocks destroyed, 5.412 blocks per game. The deliberative agent came in second with 4.054 blocks per game.

We believe that this is justified because while the reflex agent only tries to eliminate other agents when they are close to it, the deliberative looks if there are any enemies close to it in a wider range (*search threshold* distance), which means that in throughout the game, the deliberative will chase down opponents in order to eliminate them whilst the reflex simply waits for the enemies to be within its range, still destroying blocks while no enemy agents are nearby.

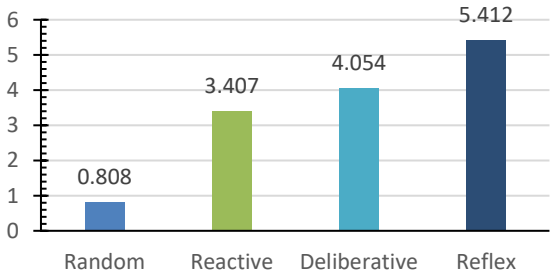

Figure 13: Average of blocks destroyed per game

5.1.5 Average number of actions per second

As we expected, the deliberative has a lower number of actions per second due to its higher computational time.

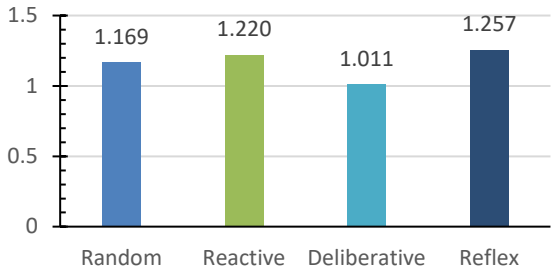

Figure 14: Average number of actions per second

5.1.6 Average survival time per game

The two agents with the highest survival time, in seconds, were the reflex and deliberative. This was expected based on their win rates since both of their win rates are much higher than the other two agents. Which means that in most games, these two are the final remaining agents, therefore their survival times would be the highest.

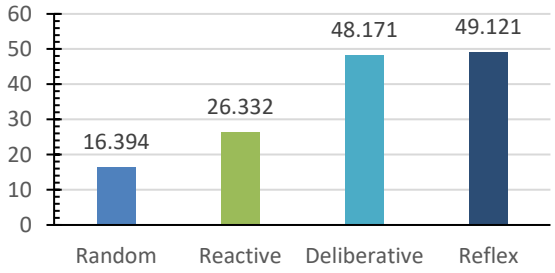

Figure 15: Average survival time per game in seconds

5.1.7 Average number of enemies eliminated per game

As we expected, the reflex agent has the best average out of all agents, with an average of 0.1584 enemies killed per game. However, this value is surprisingly low which we believe that is caused by how commonly half of the agents (random and reactive) eliminate themselves. This leads to a reduced number of opposite agent interactions that, when combined with how well the intelligent agents dodge bombs, reduces the number of agents killed by one another. On top of that, the border reduction also eliminates agents reducing further the opportunity of an agent eliminating another.

Still, the deliberative's average is much lower than the reflex's. We hypothesize that this is caused by the deliberative's BDI architecture, which means that the deliberative agent might take too long to decide to place a bomb that would eliminate another agent and, if it indeed decides to place a bomb, the enemy agent might have already changed position turning its intention into an impossible intention.

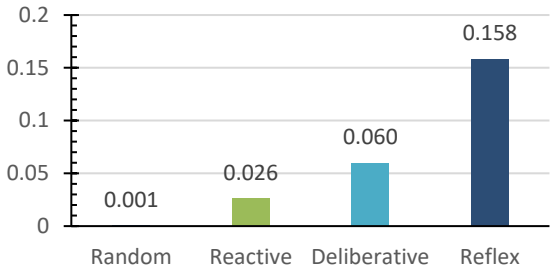

Figure 16: Average number of enemies eliminated per game

5.2 Checkerboard Game Environment

In order to boost the interaction between multiple agents, we decided to implement this additional environment.

In this environment, there are no blocks to destroy, there are just bombs to escape from or enemies to eliminate. With this, we were eager to see how our performance metrics would change based on additional agent interaction.

Besides the performance metrics, we were also interested to see the trending paths that each agent would take in absence of destructible blocks, to ensure that the **Movement Heuristic** is working correctly.

All this data was computed from 9726 game logs.

5.2.1 Percentage of game outcomes and percentage of border eliminations

To our surprise, the percentage of ties is much higher than expected when compared to the previous environment. However, we hypothesize that this was caused by the increased percentage of reflex agent eliminations to border. This would lead to a higher number of games tied since the reflex usually is one of the lasting agents.

We believe that the increased percentage of border shrinkage eliminations was caused by the fact that, since there are no destructible blocks, it would be much easier for an agent to escape an opponent's bomb.

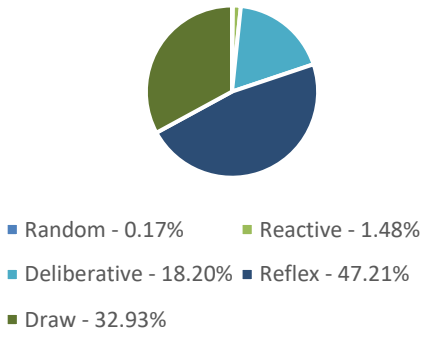

Figure 17: Percentage of game outcomes

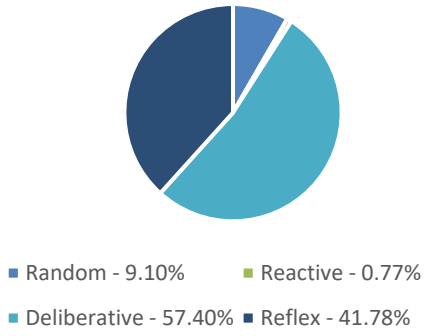

Figure 18: Percentage of border eliminations

5.2.2 Percentage of agent deaths due to their own bomb

As expected, the overall values for self-elimination were considerably lower than when there were destructible blocks. We were expecting this since the agents would not get themselves stuck in between blocks after placing a bomb.

However, the reactive's percentage was still much higher than the other two agents, we believe that this is caused by reactive's constant movements to the center cell without checking in advance if it is going to be in a bomb explosion range.

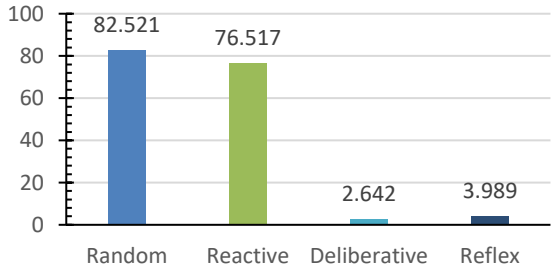

Figure 19: Percentage of agent deaths due to their own bomb

5.2.3 Average number of bombs placed per game

The reflex agent still had the highest number of bombs placed per game which was what we expected. However, the deliberative agent had a much lower number when compared to the previous environment. Like explained before, we believe that this was caused by how the BDI architecture works. This issue was especially noticeable in this environment since the only reason that an agent has to place a bomb is in order to try to eliminate an opponent.

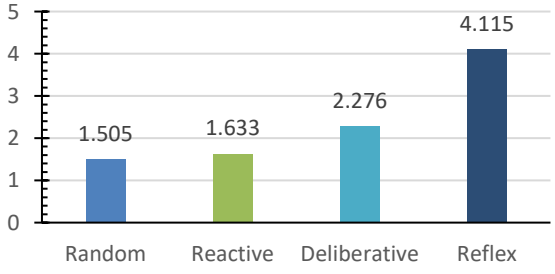

Figure 20: Average number of bombs placed per game

5.2.4 Average number of enemies eliminated per game

As expected from a more agent versus agent environment, the average number of enemies eliminated per game increased significantly for all the agents. As explained previously, we believe that this was caused because the agents only focus on two main objectives, which are eliminating opponents and escaping bombs.

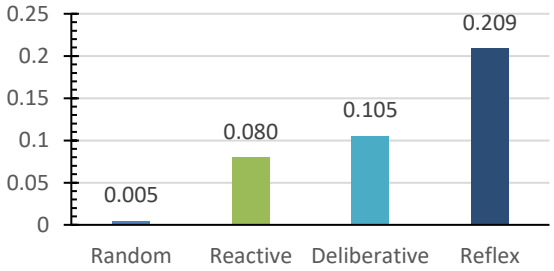

Figure 21: Average number of enemies eliminated per game

5.3 Movement Heuristic Validation

To validate our **Movement Heuristic**, we decided to analyze the logs from the environment with no destructible blocks to check if the agents were in fact following this heuristic to lead them closer to the center of the board.

The heatmap color gradient is based on the steps in that specific cell of the board.

We built the following heatmap based on the 9726 logs.

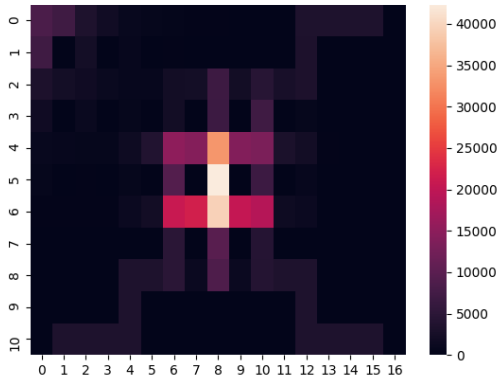

Figure 22: Trending paths heatmap with no destructible blocks

As we can see in the heatmap the intelligent agents – placed in top right, bottom left, and bottom right – always follow a path towards the middle cell.

To further verify the **Movement Heuristic**, we also did the same study for the normal game environment.

The following heatmap was computed from the 9941 game logs.

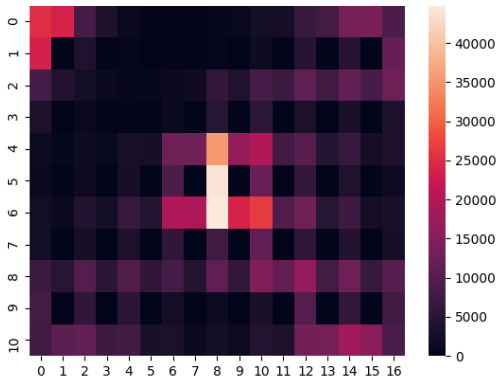

Figure 23: Trending paths heatmap with destructible blocks

In this heatmap we can also confirm that the agents are following the **Movement Heuristic** correctly, even though it is not as obvious as in the previous heatmap since the destructible blocks block the direct path to the center.

6 Conclusion

Based on all the data obtained throughout our experiences, we can firmly declare the **reflex agent as the most suited agent** for this environment.

This was what we hypothesized before the experiment because the **reflex** agent has quick reaction time, like the reactive agent, while still being careful with what will happen to it with each possible move considering future states.

To our surprise, the **deliberative** was better than expected. We think that, even though this agent has a slower decision-making due to its high computational times, it usually makes safer decisions, which made it survive a lot longer than expected. However, as we expected, the fact that its beliefs are not able to keep up with the quick environment changes were a detrimental factor to its performance.

When we initially thought about the **reactive** agent, we did not take into consideration its major flaw, where the agent decides a *not safe move*, getting itself stuck, or moving into a bomb explosion range, which leads to its high self-kill percentage. This was clearly noticeable to us after implementing.

However, all of these agents clearly showed intelligent and advanced behaviors when compared to the baseline, the **random** agent.

7 Implementation Issues

During our project development there were a couple of issues, but only one still remains. This bug, which is visually noticeable, occurs within the GUI library used – game2dboard. It happens when multiple changes occur at the same time to the board. For instance, when multiple agents place a bomb at the same time some of the bombs might not appear in the GUI, but we verified that they are indeed in the board, so the agents still behave accordingly. Another instance of this same bug occurs when more than 1 agent tries to move to a cell at the same time, in this case, a “ghost” image of one of the agents is left behind.