1. Introduction

This report will evaluate two AI techniques, reinforcement learning and the rule-based system, implemented in a top-down shooter game. Two agents will fight each other, and the agent who can kill the other fastest will be the winner.

2. Background

2.1 Reinforcement Learning

Reinforcement learning is a method in machine learning. An agent will take action in an environment with the data it observed and its state. Then the environment will return the agent's reward and its next state back. So the agent will take these data into a next action. The goal of this kind of learning is to achieve the possible maximum reward.

2.2 ML-Agents toolkit

Juliani et al (2018) define the Unity Machine Learning Agents Toolkit (ML-Agents) as "an open-source Unity plugin that enables games and simulations to serve as environments for training intelligent agents." Machine learning methods, such as reinforcement learning, imitation learning, are used to train agents. This is done with Python API.

2.3 Ray Perception

ML-Agents provides a script for doing ray casts outward from the agent providing which specific objects are in the agents' sights and distance between those objects and agents. This can make the agent knows objects in the environment.

3. Methodology

3.1 Game Mechanics

At the start of the game, players will be spawned in random positions. The goal of this game is to kill other players as fast as possible. Each player has 100 health points and 12 bullets per 1 clip. The player can move freely in both 2 axes (X and Y) and can shoot the projectile in any angles relative to the player. Each projectile can cause 50 damage points. There are 0.3 seconds time cooldown for each firing and 1.15 seconds for reloading time. If the players run out of bullets, the game will reload the gun for those players automatically. If the players have zero or fewer health points, they will be considered as a dead player and will be removed from the game area. The game also can spawn random walls for players to escape the other players' aiming sight by iterating the area coordinates one by one in both X and Y axis if the random number is more than the threshold the wall will be spawned.

3.2 Reinforcement Learning

The ML-Agents toolkit was used to implement because it provides examples and documentation that can be followed easily, and it manages all the communication between the game engine and Tensorflow framework which is a popular machine learning framework.

3.2.1 Training Environment

The training environment consists of nine play areas. Each area is 15x15 and has a wall spawning threshold at 0.95. Agents have maximum steps of 5000. The agent is trained with the same kind of agent only.

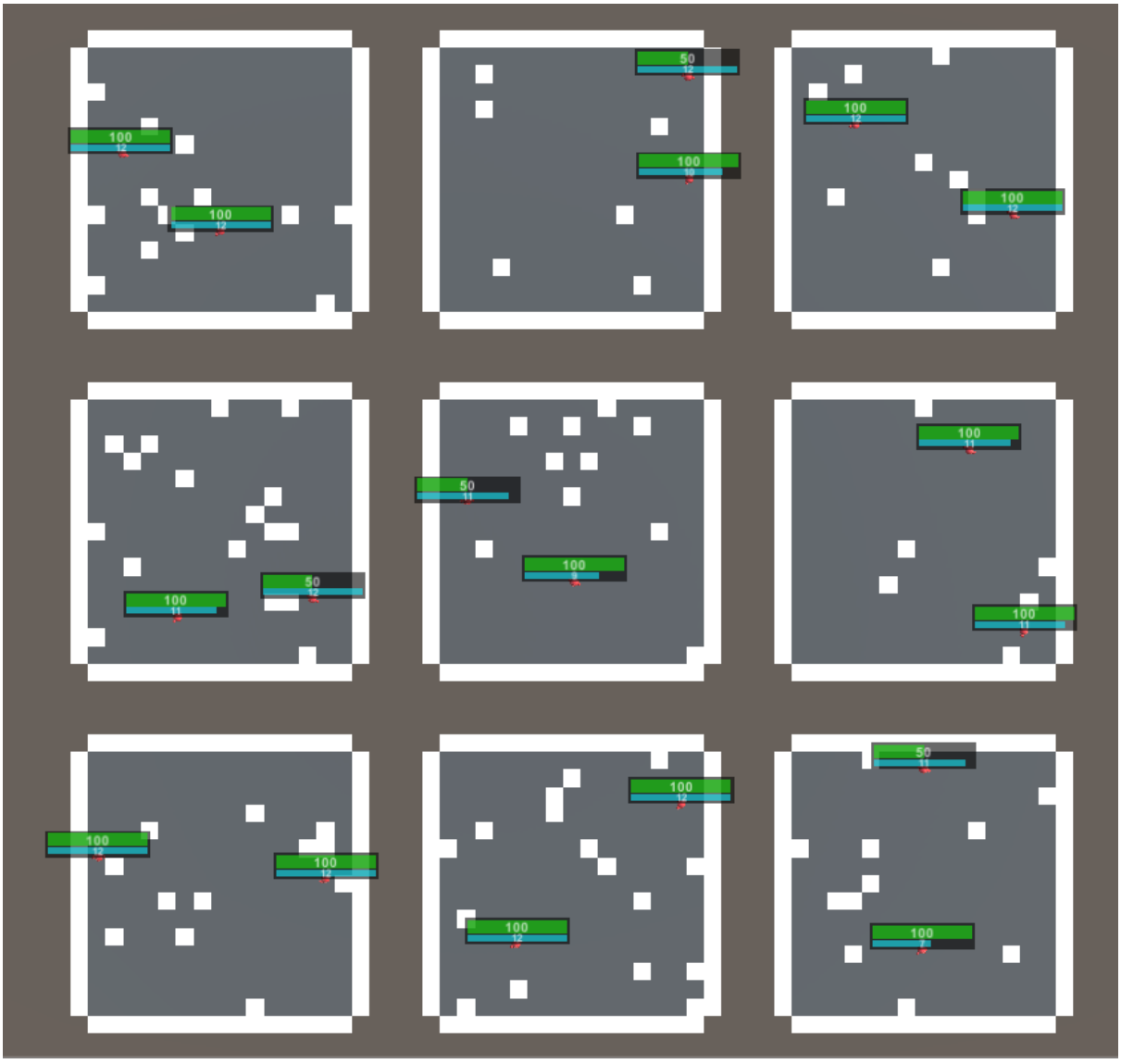


Figure 1 A picture of a training environment

3.2.2 Agent Setup #1

This setup is my first setup for reinforcement learning. The vector action space type is continuous (a series of real-valued floats) because the agent can use the value from output in the model directly to the game.

Vector Observation (Vector Input)

- Player Health
  Current player health point normalized to 0.0 - 1.0
- Player Ammo
  Current player ammo normalized to 0.0 - 1.0 (current ammo divided by max ammo)
- Reloading status
  If a player is reloading, this value will be 1 otherwise 0.
- Ray perceptions
  Detect if there are other players, walls or projectiles on rays that cast from 0, 15, 30, 45, 60,

75, 90, 105, 120, 135, 150, 165, 180 degrees relative to the agent. Each ray has this data structure:
    - Type of object identified by using the array index
        - If object on ray is a player, value in [0] is 1, otherwise the value is 0.
        - If object on ray is a random wall, value in [1] is 1, otherwise the value is 0.
        - If object on ray is a wall of play area, value in [2] is 1, otherwise the value is 0.
        - If object on ray is a projectile, value in [3] is 1, otherwise the value is 0.
        - If object on ray is none of the above, value in [4] is 1, otherwise the value is 0.
    - Distance of object divided by max distance of the ray which is 60.
- Agent's current position relative to game area normalized to -1.0 - 1.0 in X and Y axis.

Vector Action (Vector Output)

- Horizontal movement
    - Value 0 is no movement
    - Value -1 is moving left.
    - Value 1 is moving right.
- Vertical movement
    - Value 0 is no movement.
    - Value -1 is moving down.
    - Value 1 is moving up.
- Aiming angles relative to the agent
  Degrees of angle from up direction normalized to -1.0 - 1.0

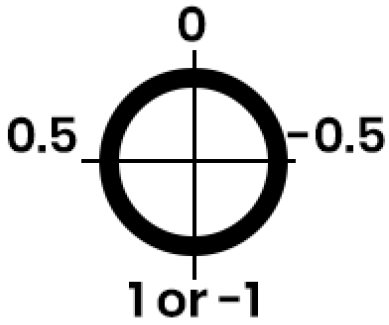


Figure 2 A picture illustrating the relation between value and aiming angles

- Fire
  If value is in range -1 to 0, agent won't fire a projectile and if it is in 0 to 1, the agent will fire.

3.2.3 Agent Setup #2

The change from the first setup is aiming mechanic of the agent so only vector action is slightly different.

Vector Action (Vector Output)

- Horizontal movement
    - Value 0 is no movement
    - Value -1 is moving left.
    - Value 1 is moving right.
- Vertical movement

    - Value 0 is no movement.
    - Value -1 is moving down.
    - Value 1 is moving up.
- Rotating direction
    - Value 0 is no rotation.
    - Any value less than 0 is counterclockwise.
    - Any value more than 0 is clockwise.
- Rotating speed
    - The agent will rotate using this value multiplied by 40 degrees at one game update frame.
- Fire
  If value is in range -1 to 0, an agent won't fire a projectile and if it is in 0 to 1, the agent will fire one.

3.2.4 Agent Setup #3

The change from the second setup is changing vector action space type to discrete (a series of integers).

Vector Action (Vector Output)

- Horizontal movement
    - Value 0 is no movement
    - Value 1 is moving left.
    - Value 2 is moving right.
- Vertical movement
    - Value 0 is no movement.
    - Value 1 is moving up.
    - Value 2 is moving down.
- Rotating direction
    - Value 0 is no rotation.
    - Value 1 is counterclockwise.
    - Value 2 is clockwise.
- Rotating speed
    - The agent will rotate using this mapping at one game update frame.
        - Value 0 = 1 degree
        - Value 1 = 5 degrees
        - Value 2 = 10 degrees
        - Value 3 = 20 degrees
        - Value 4 = 40 degrees
- Fire
  If value is 0, agent won't fire projectile and if it is 1, agent will fire.

3.2.5 Trainer Hyperparameters

Setup #1 and Setup #2

```
default:
    trainer: ppo
    batch_size: 2048
    beta: 1.0e-2
    buffer_size: 16384
    epsilon: 0.2
```

```
    hidden_units: 384
    lambd: 0.95
    learning_rate: 1e-3
    max_steps: 5e5
    memory_size: 256
    normalize: true
    num_epoch: 3
    num_layers: 2
    time_horizon: 256
    sequence_length: 64
    summary_freq: 2000
    use_recurrent: false
    vis_encode_type: simple
    reward_signals:
        extrinsic:
            strength: 1.0
            gamma: 0.99
```

Setup #3

```
ShootBrainDiscrete:
    hidden_units: 128
    normalize: false
    beta: 5.0e-3
    batch_size: 1024
    buffer_size: 10240
    max_steps: 5.0e5
    time_horizon: 64
    summary_freq: 1000
```

3.2.6 Rewarding

- -0.05 penalty for each action that the agent takes
  The agent needs to kill another player as fast as possible.
- -0.5 penalty if agents' projectiles hit the wall
  The agent should not shoot randomly at the wall.
- -1 penalty if agents have 0 or less health point
  The agent fails the killing goal.
- -1 penalty if agents are hit by other players' projectiles
- +1 reward if agents' projectiles can cause damage to another player

3.3 Rule-based system

The agent will have these data for making actions:

- In front of agent object detection using ray perception
- A sphere collider detecting projectiles with a small radius
- Closest player position
- Wall detection in 4 directions (North, South, East, West)

With 2 systems run simultaneously in a game frame:

- Movement System
  - Every move decision has a random number of moving times between 0 and 1 second.

    - If there's a wall beside an agent (in 4 directions), move in the opposite direction.
- Aiming System
    - An agent will lock the target on the closest player.
    - If an agent can see another player (no object in the way), it will fire projectiles.

3.3.1 Aiming System

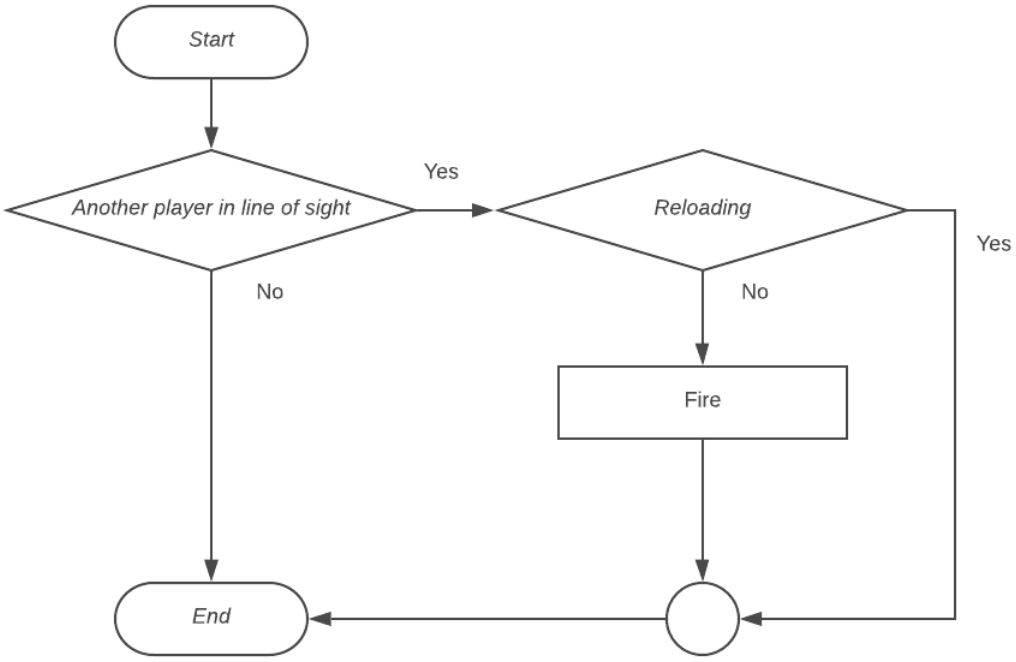


Figure 3 A flowchart of the aiming system

3.3.2 Movement System

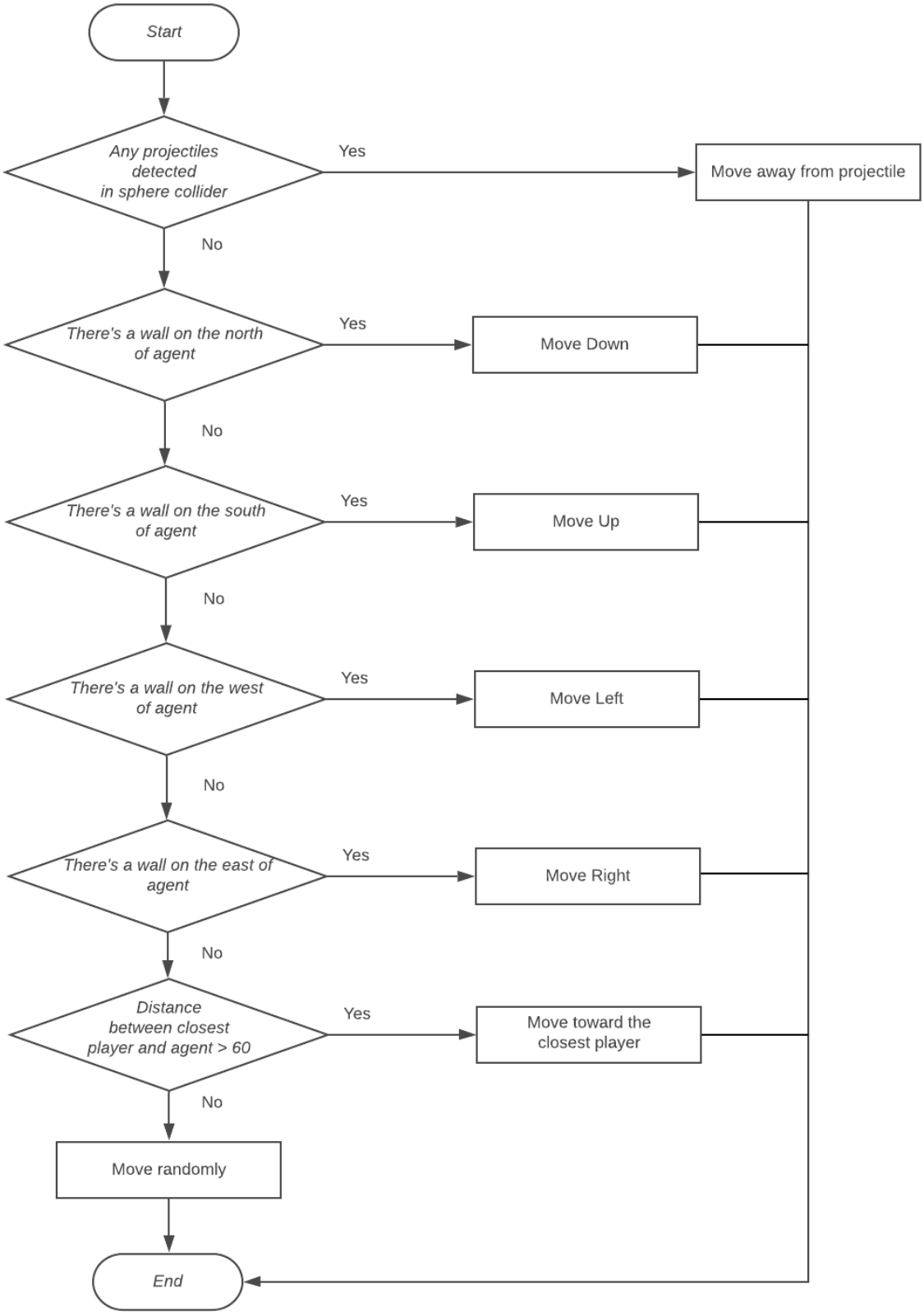


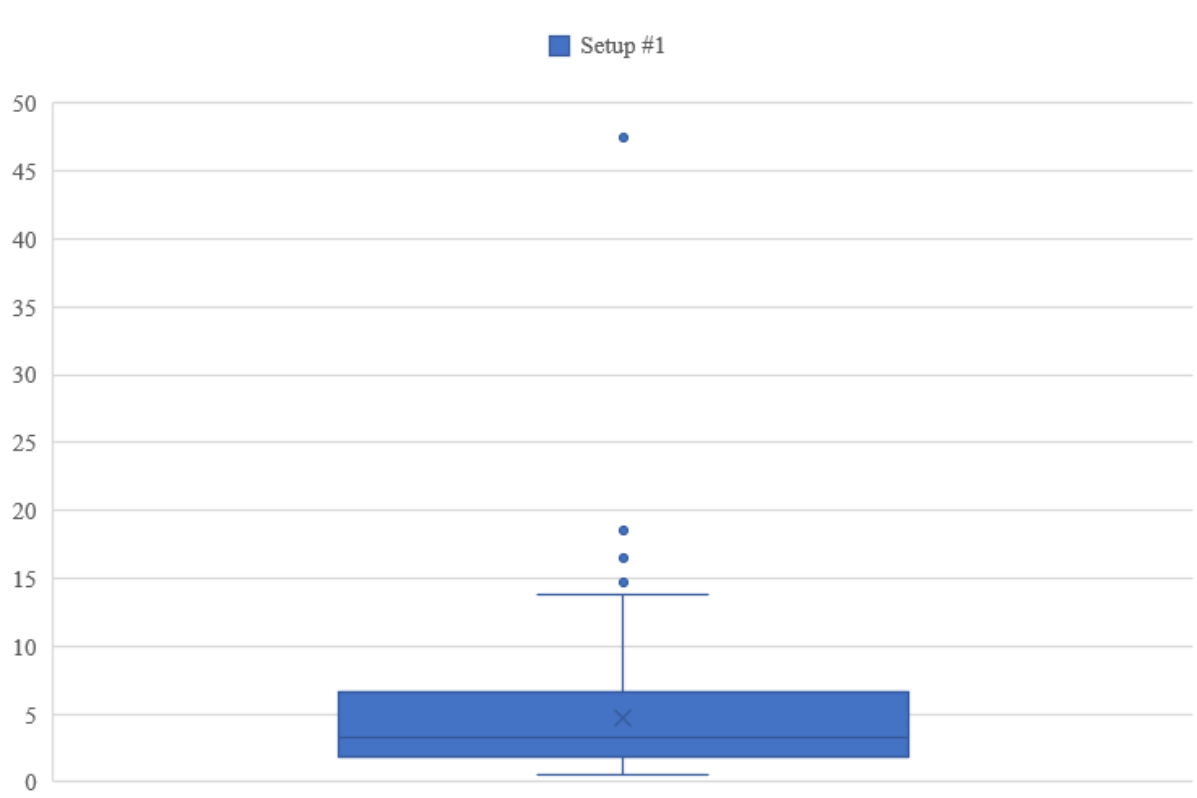
Figure 4 A flowchart of the movement system

4. Results

All reinforcement learning setups were trained for 500,000 steps. Each setup takes about 2 hours. The agent can be evaluated by how many seconds the agent takes for killing another player. If agents can't accomplish the goal within 5,000 steps, the game will be ended and counted as a failed game. If both players die at the same time (draw), the game will be also counted as a failed game. The game area is set up as the same as agent trains. Each couple fights in 200 games. The game was locked at 60 frames per second.

4.1 Setup #1 vs Setup #1

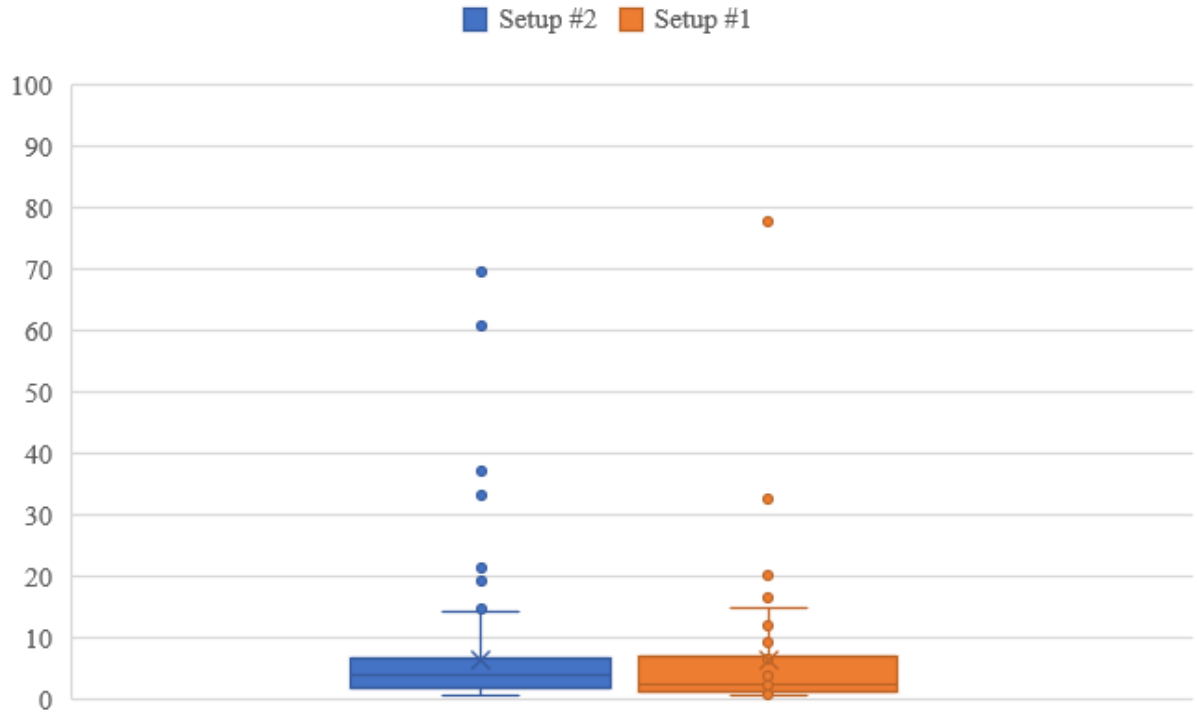
Number of success games: 188 (94%)
Number of failed games: 12 (6%)
Average killing time: 4.69 seconds
Min killing time: 0.50 seconds
Max killing time: 47.5 seconds



4.2 Setup #1 vs Setup #2

Number of success games: 183 (91.5%)
Number of failed games: 17 (8.5%)

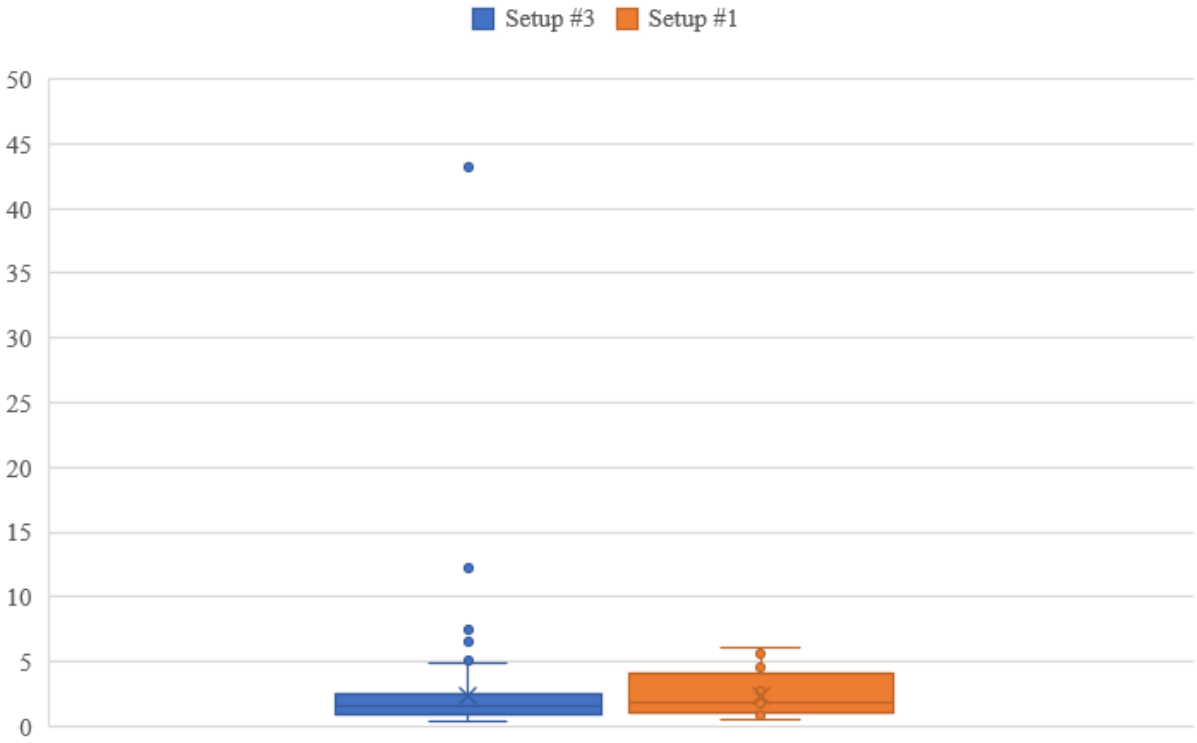
Setup #2	Setup #1
Kills: 138 Average killing time: 6.13 seconds Min killing time: 0.44 second Max killing time: 69.68 seconds	Kills: 50 Average killing time: 6.40 seconds Min killing time: 0.4 second Max killing time: 77.80 seconds



4.3 Setup #1 vs Setup #3

Number of success games: 195 (97.5%)
Number of failed games: 5 (2.5%)

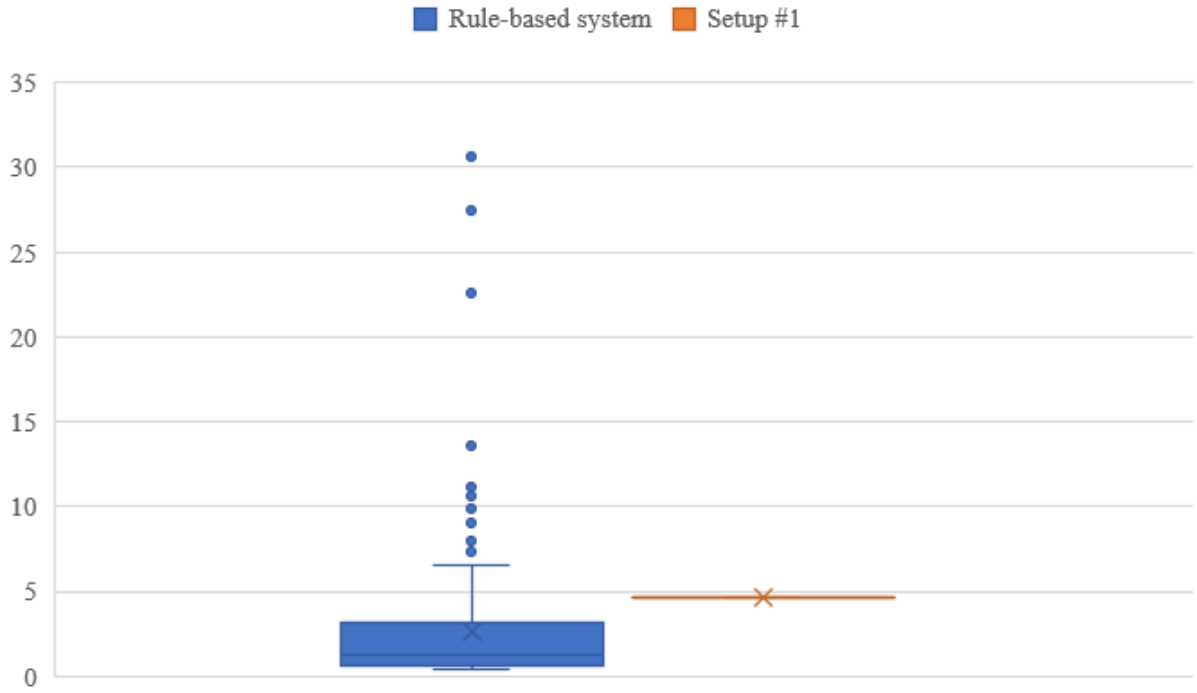
Setup #3	Setup #1
Kills: 186 Average killing time: 2.30 seconds Min killing time: 0.42 seconds Max killing time: 43.26 seconds	Kills: 12 Average killing time: 2.40 seconds Min killing time: 0.48 seconds Max killing time: 6.10 seconds



4.4 Setup #1 vs Rule-based system

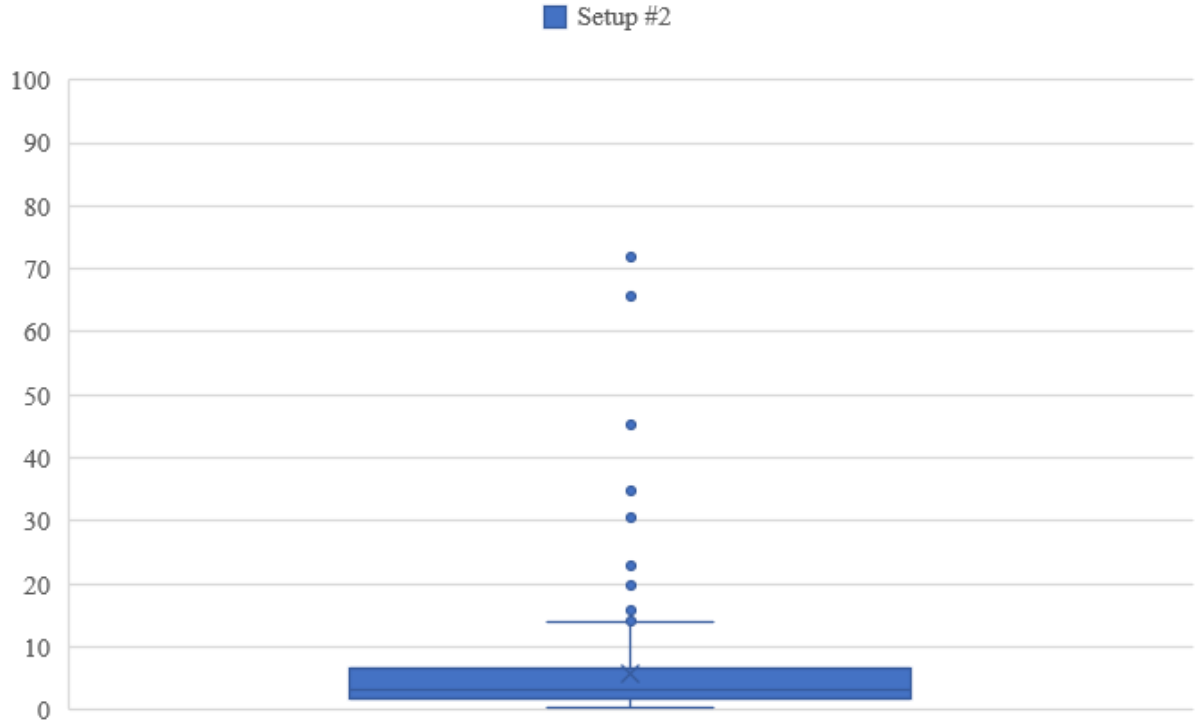
Number of success games: 193 (%)
Number of failed games: 7 (%)

Rule-based system	Setup #1
Kills: 192 Average killing time: 2.66 seconds Min killing time: 0.38 seconds Max killing time: 30.64 seconds	Kills: 1 Average killing time: 4.64 seconds Min killing time: 4.64 seconds Max killing time: 4.64 seconds



4.5 Setup #2 vs Setup #2

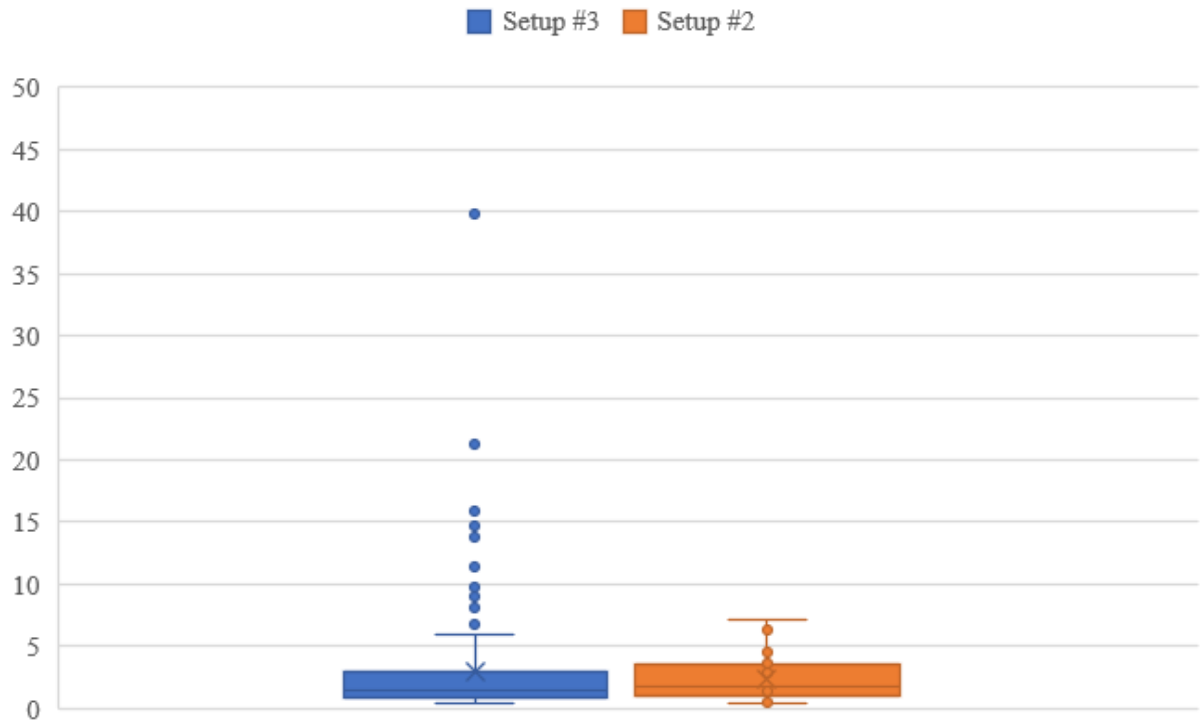
Number of success games: 184 (92%)
Number of failed games: 16 (8%)
Average killing time: 5.85 seconds
Min killing time: 0.44 seconds
Max killing time: 71.88 seconds



4.6 Setup #2 vs Setup #3

Number of success games: 193 (96.5%)
Number of failed games: 7 (3.5%)

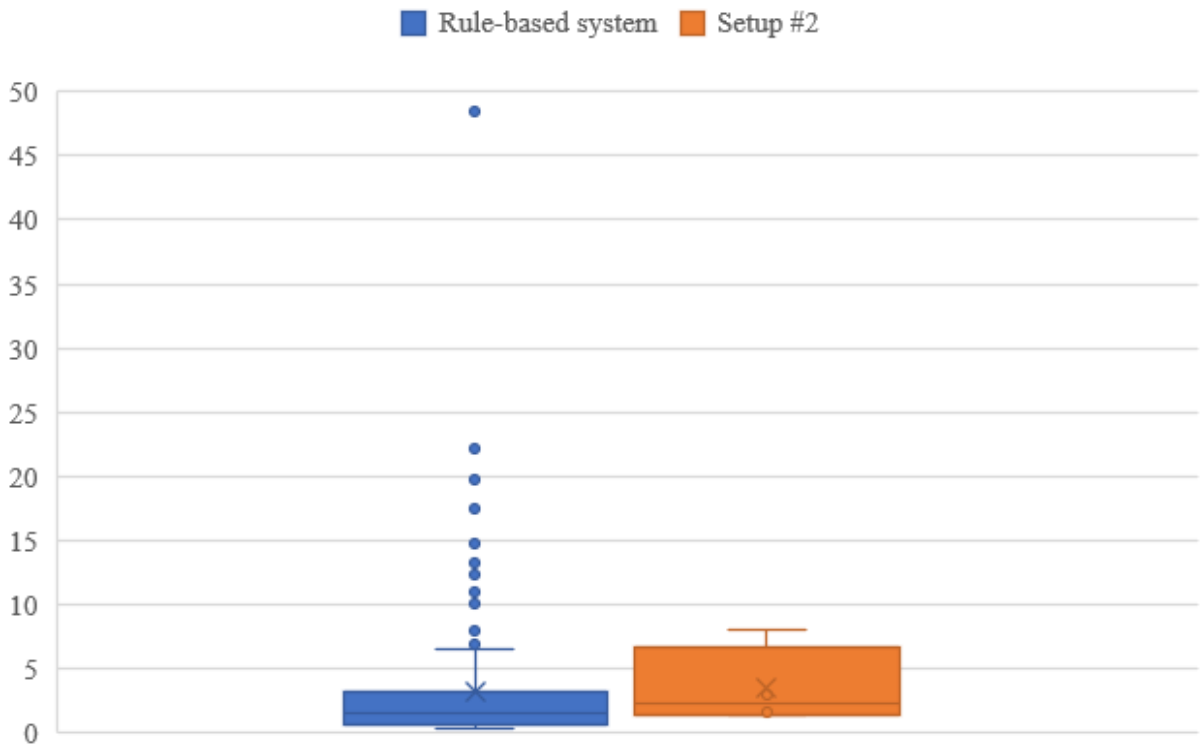
Setup #3	Setup #2
Kills: 168 Average killing time: 2.98 seconds Min killing time: 0.40 second Max killing time: 39.74 seconds	Kills: 31 Average killing time: 2.31 seconds Min killing time: 0.44 second Max killing time: 7.18 seconds



4.7 Setup #2 vs Rule-based system

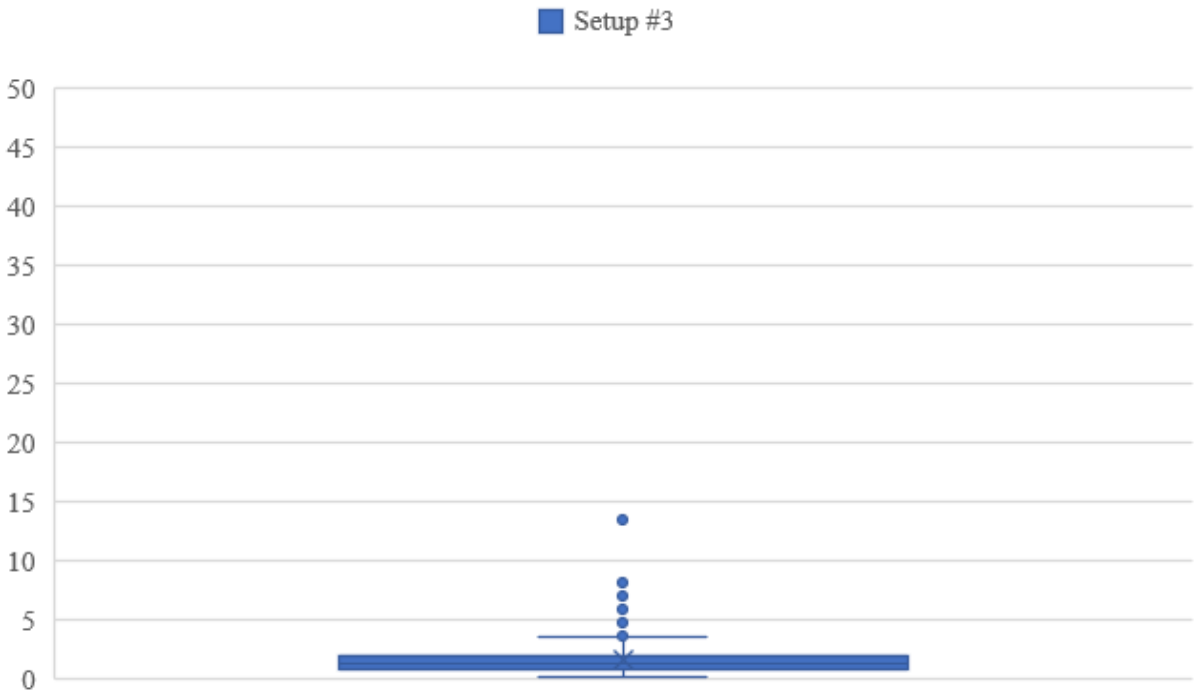
Number of success games: 192 (96%)
Number of failed games: 8 (4%)

Rule-based system	Setup #2
Kills: 191 Average killing time: 3.30 seconds Min killing time: 0.34 seconds Max killing time: 48.44 seconds	Kills: 4 Average killing time: 3.50 seconds Min killing time: 1.38 seconds Max killing time: 8.00 seconds



4.8 Setup #3 vs Setup #3

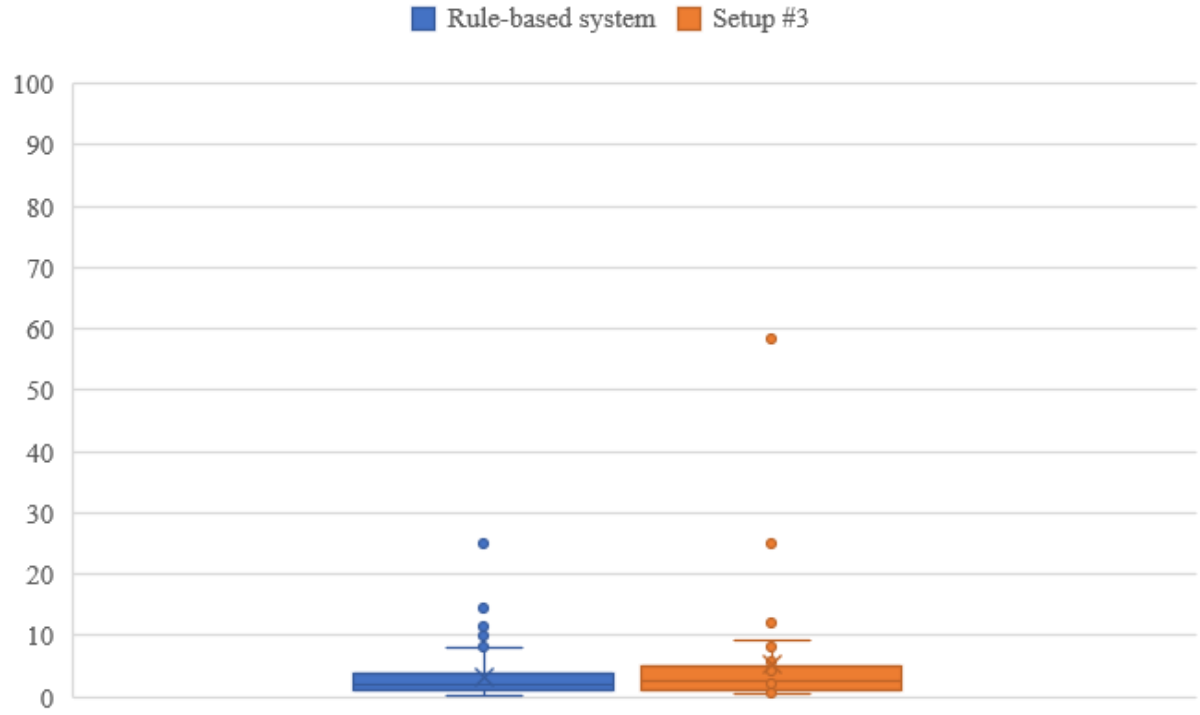
Number of success games: 190 (95%)
Number of failed games: 10 (5%)
Average killing time: 1.67 seconds
Min killing time: 0.20 seconds
Max killing time: 13.52 seconds



4.9 Setup #3 vs Rule-based system

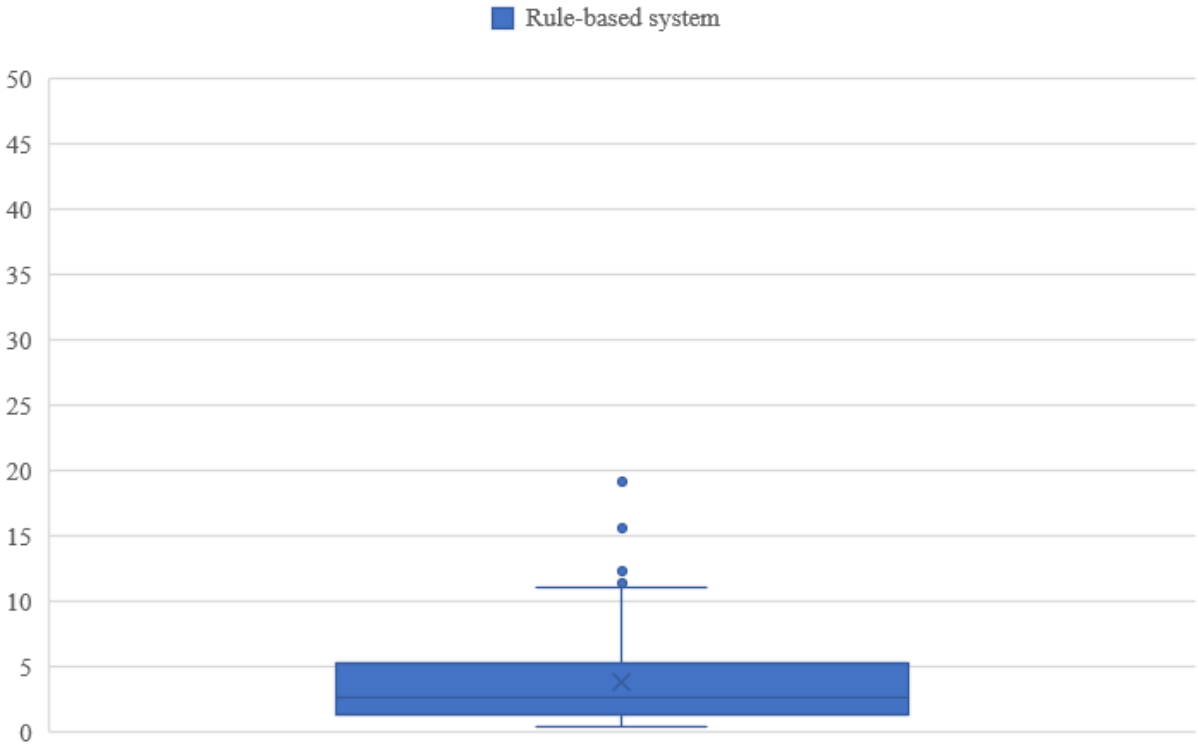
Number of success games: 191 (95.5%)
Number of failed games: 9 (4.5%)

Rule-based system	Setup #3
Kills: 159 Average killing time: 3.18 seconds Min killing time: 0.12 seconds Max killing time: 25.10 seconds	Kills: 34 Average killing time: 5.43 seconds Min killing time: 0.60 seconds Max killing time: 58.44 seconds

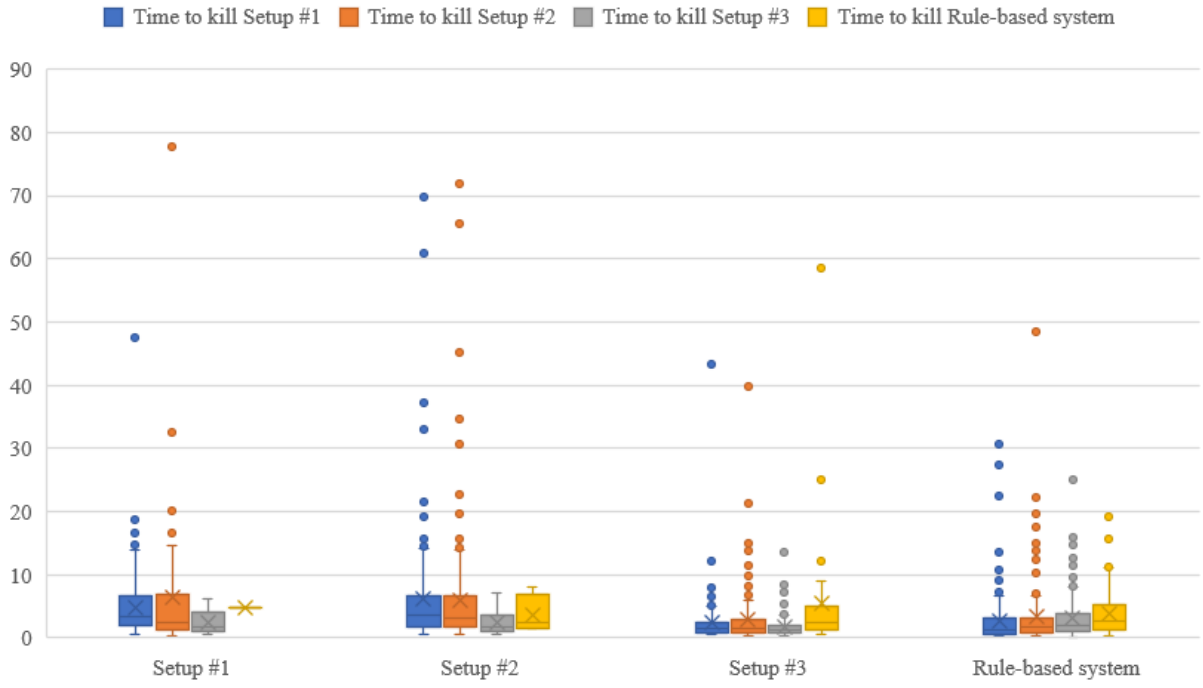


4.10 Rule-based system vs Rule-based system

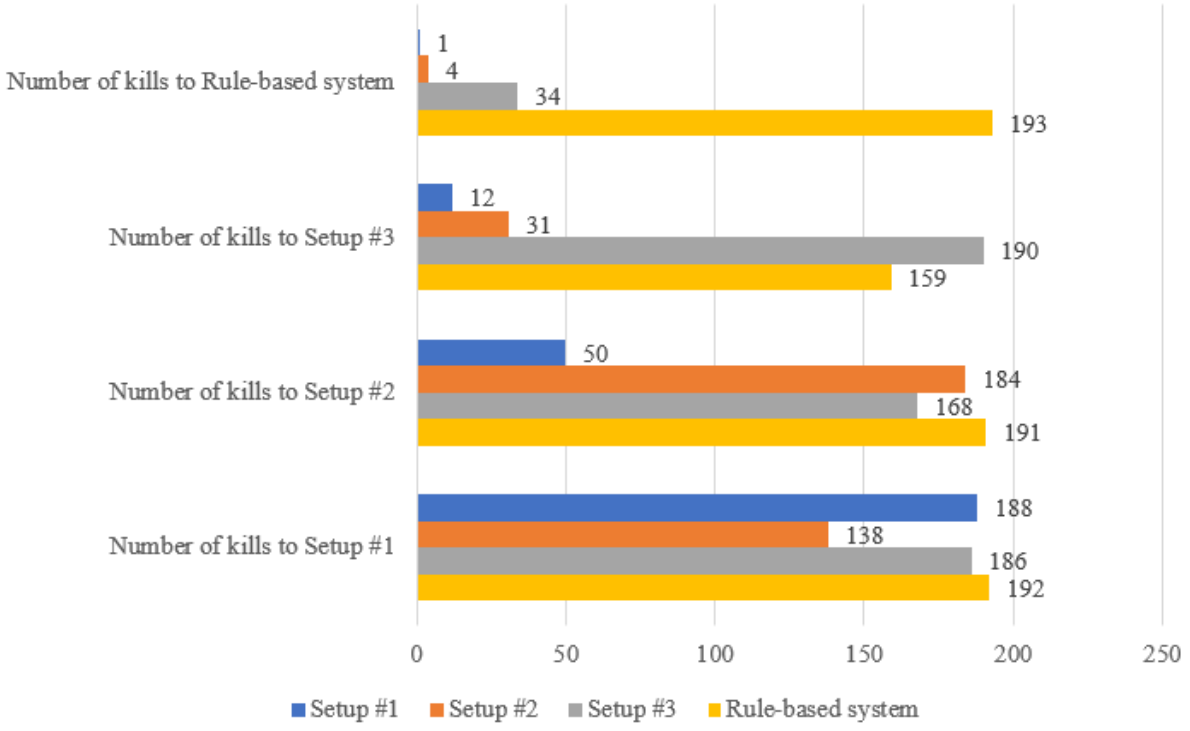
Number of success games: 193 (96.5%)
Number of failed games: 7 (3.5%)
Average killing time: 3.72 seconds
Min killing time: 0.38 seconds
Max killing time: 19.14 seconds



4.11 All agents killing time graph



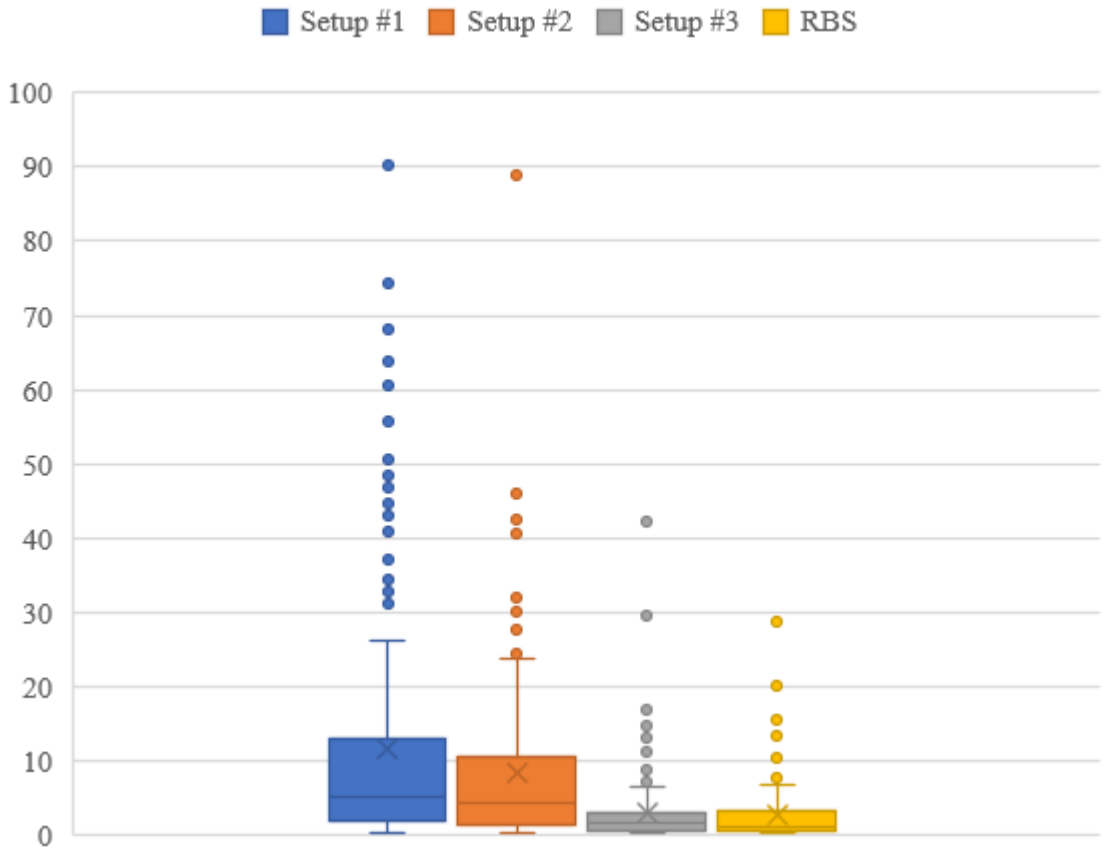
4.12 All agents' kill graph



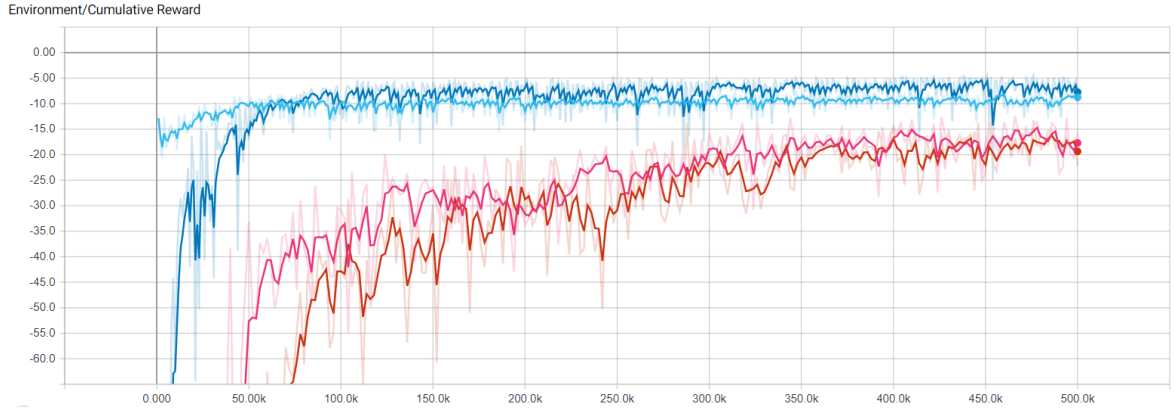
4.13 All agents vs Stationary player

The agent in each setup would fight against an idle player. This can evaluate how the agent can kill another player who is staying behind a random wall.

Agent Setup	**Setup #1**	**Setup #2**	**Setup #3**	**Rule-based system**
Number of success games	160	173	191	188
Number of failed games	40	27	9	12
Average killing time	11.84	8.41	3.21	2.85
Min killing time	0.38	0.36	0.38	0.34
Max killing time	90.32	88.77	42.30	28.82

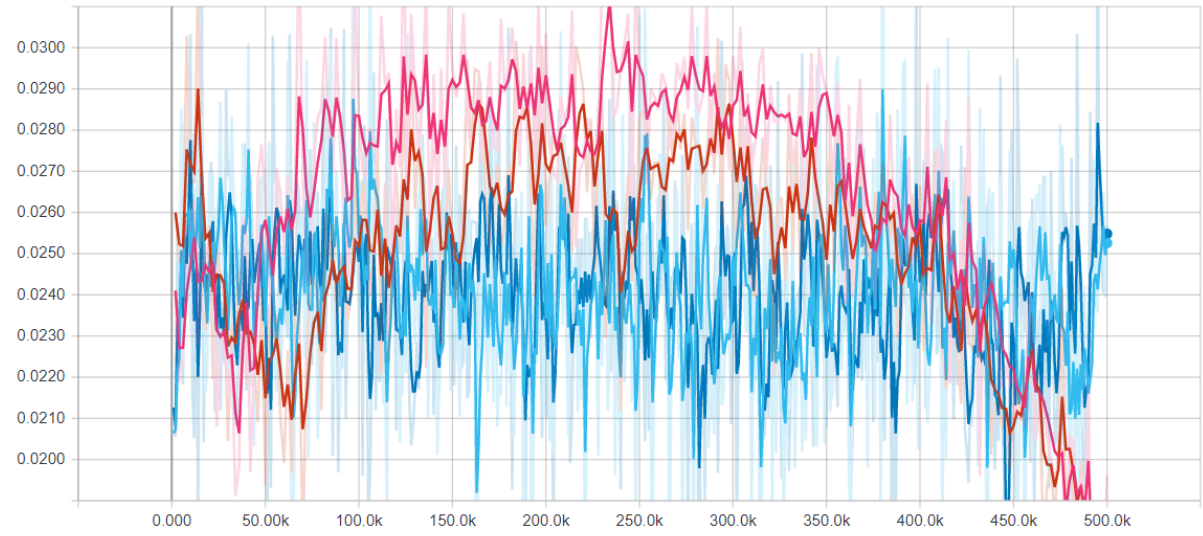


4.14 Cumulative Reward Graph



- Setup #1 vs Setup #1
- Setup #2 vs Setup #2
- Setup #3 vs Setup #3
- Setup #3 vs Rule Based system

4.15 Policy Loss Graph



- Setup #1 vs Setup #1
- Setup #2 vs Setup #2
- Setup #3 vs Setup #3
- Setup #3 vs Rule Based system

5. Discussion

From the results, a rule-based system agent has the highest kill in each couple and both an agent from setup #3 and a rule-based system have the least killing time compared to the others. For a rule-based system agent, a kill is guaranteed because it keeps locking on the target all the time. This would be unfair if a human player fought against this agent (like an aimbot tool in first-person shooting game). An agent from setup #3 is more suitable even though it might be difficult to kill but it's not unbeatable.

Comparison

Reinforcement agent	Rule-based system agent
- The behavior of the agent might be unexpected. It can open up more possibilities. - The training process consumes time. - Code change or trying something with the agent needs retraining which consumes a long period of time to see results and make more change. - The person who implements doesn't know how it works.	- The behavior of the agent is hardcoded so it can be expected. - It's easier to debug. - Use less time to implement and get the agent in a usable state. - Its effectiveness relies heavily on the agent's logic design.

5.1 Reinforcement agent's vector action design changes

In setup #1, due to how vector action design, the agent can aim by specifying the angle it wants and after some observations, it looked like an agent aim randomly and unnaturally. This led to setup #2 which vector action design about aiming was changed to make an agent rotate itself instead. The result from this change was satisfying as the agent aims naturally and the number of kills is improved.

Changing vector action from a series of floats to integers (Setup #2 to setup #3) can limit how an agent takes action which is more suitable to a simple problem that doesn't require many moves in a set of actions. An agent took less time and solved the problem more effective (more kills) according to the result.

5.2 Limitation of reinforcement agent

5.2.1 Walls

In some numbers of games, the reinforcement learning agent stuck on a corner of walls like in this scenario.

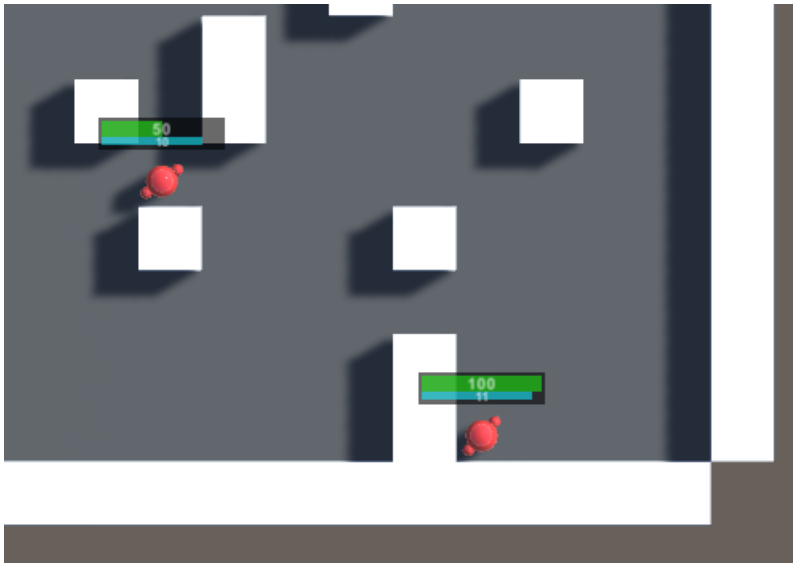


Figure 5 two agents didn't know how to deal with corners.

As in the figure, two agents tried to move against walls and the game ended as a failure.

5.2.2 Setup #3 training against rule-based system agent

An experiment was conducted which took the agent in setup #3 to train against the rule-based system agent. From the cumulative reward graph, the reward starts at -12 and fluctuates between -9 to -10. After training sessions, it could be observed that the trained agent didn't fire any projectiles and tried to die as fast as possible (because there's a penalty every action) to reduce penalties in one game. The agent died faster than it can learn that it can maximize its reward by killing another player. So, this agent is a failure.

5.3 Further improvement

- Add surviving as a new goal to the agent
  Changing from adding a penalty to a reward for every action that the agent takes while surviving can lead the agent not to do anything. Because the best way to keep getting a reward is being idle.
- Make a reinforcement learning agent learns how to deal with walls
  Adding a penalty when the agent gets stuck in the corner might help.

6. Conclusions

In terms of development, we can implement an agent performs slightly better than a reinforcement learning agent using the rule-based system with less development time. According to the result, the rule-based system is the best for always winning agent because of its highest kill chance. While setup #1 reinforcement learning agent performs the worst because its vector action is too complex for the game and it needs more training time.

7. References

Juliani, A., Berges, V., Vckay, E., Gao, Y., Henry, H., Mattar, M., Lange, D. (2018). *Unity: A General Platform for Intelligent Agents. arXiv preprint arXiv:1809.02627.* Available at: https://github.com/Unity-Technologies/ml-agents (Accessed: 24 October 2019).