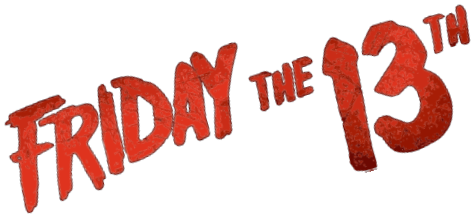



# AI Tag powered by Object Detection and Reinforcement Learning

*Fracassi Riccardo*
*1956794*

*Perera Lachitha*
*2042904*

*Maiorano Mattia*
*2009650*

*Principe Enrico*
*2085060*

***Abstract***

*We present the development of a game environment inspired by the horror theme of "Friday the 13th", in which two intelligent agents, Jason and Victim, confront each other in a tag-type context.*

*The environment was entirely developed by us in Java language and populated with interactive elements, recognizable by agents through object detection. Each animated entity is managed by a separate artificial intelligence, trained through reinforcement learning to achieve its goal: capture (Jason) or escape and survive (the Victim).*

*The adoption of a multi-agent approach in a playful and competitive context has allowed the analysis of complex dynamics of both emerging behavior and strategic adaptation, offering a useful test bench to evaluate the generalization and reactivity skills of agents. In addition, the integration of object detection as a preponderant formula for environmental perception realistically simulates sensory limitations, forcing agents to make decisions in conditions of partial visibility.*

*However, the development of such a system has involved several challenges: synchronization between perception and action, computational latency management for visual processing and optimal distribution of rewards, to ensure effective and non-polarized learning.*

*Added to these are the difficulties related to training in complex and dynamic environments, which require advanced exploration and optimization strategies.*

*The results obtained demonstrate the feasibility and effectiveness of the approach.*

# Environment

The developed game environment reproduces an extended campsite, characterized by natural and artificial elements distributed in order to encourage exploration, tactical planning and strategic interaction between the two opposing agents: Jason, with an offensive objective, and the Victim, with a defensive and exploratory objective.

### 1. Environment properties

The game environment has been designed to offer a balance between exploration, interaction and strategy, through a series of interactive elements.

The chests ( image 1.1 ) play the role of temporary hiding places for the Victim, but can be permanently destroyed by Jason, introducing an element of progressive mutation of the map. Unlike these, wells offer a safe and indestructible refuge, although more limited in number.

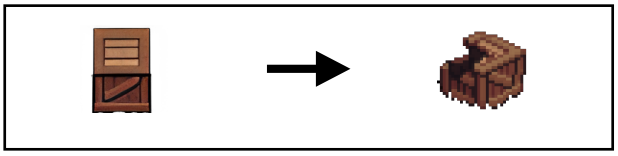


*Image 1.1*

The areas of tall grass and lakes ( image 1.2 ) represent areas capable of slowing down the movement of both agents and partially obscuring the view, encouraging the use of elusive strategies and a more conscious use of the paths. Added to this are physical obstacles such as trees that impose movement constraints, increasing the tactical complexity of the scenario.

*Image 1.2*

Central to the gaming experience is the presence of two identical houses, accessible by both agents, inside which there are furnishings and hiding places ( such as beds, tables and trunks ) and, above all, a winning object for each: a battery and a phone ( image 1.3 ). These objects are indispensable for the Victim in order to complete his goal and request rescue, thus activating the final phase of the game.

### 2. Victory Objects

The designed space is based on an asymmetrical setting, in which each agent has a specific and mutually exclusive objective.

Jason, the stalking agent, has as its sole purpose to locate and eliminate the Victim.

*Image 1.3*

The Victim, on the contrary, must pursue a sequence of chained objectives: collect two key objects, a battery and a phone, located inside the two houses, start an emergency call and finally survive for sixty seconds waiting for the arrival of help.

This multiple mission structure for the Victim introduces a temporal and sequential component that increases the tactical depth of the environment. In fact, it is not a simple chase, but a game composed of several phases, each of which requires specific behavior, alternating moments of exploration with more furtive ones. On the other hand, Jason must dynamically adapt to the evolution of the scenario, trying to anticipate the movements of the Victim and limit his progression towards the objectives.

## 3. Functional Analysis of the Environment

The game design favors a structured system capable of stimulating intelligent behaviors, avoiding excessive linearity and easily optimized situations.

First, the asymmetry between the two agents encourages the specialization of the policies learned. Differences in terms of objectives, available actions and methods of environmental interaction push agents to adopt complementary strategies, generating a competitive and non-cooperative learning environment.

A second relevant aspect concerns the limitness of the observations. Visual obstacles, and darkening areas (such as lakes or tall grass) make the environment partially observable. In this context, agents must develop more generic models of the environment and adversaries, relying on incomplete and often ambiguous information. This favors the emergence of cautious, anticipatory behaviors and, in some cases, the use of internal memory or temporary heuristics.

The great variety of possible situations, moreover, derived from the dynamic arrangement and the interaction of environmental elements, favors a great diversity of episodes.

Various hiding places, alternative paths, mobile obstacles and destructible objects are able to deeply modify the game scenario, giving the models a wide range of stimuli to interpret. This entails a higher need for generalization, limiting the possibility of overfitting on recurring patterns.

A further element of interest lies in the need, for at least one of the agents (the Victim), to balance short- and long-term objectives. The search for a hiding place must be mediated by the progression towards the collection of objects, introducing an implicit form of sequential planning that represents an excellent test bed for AI training in complex and multi-phase environments.

## 4. Learning Difficulties and Challenges

Despite the numerous advantages the environment brings, it also presents some relevant challenges in terms of learning and policy stability.

The first critical issue concerns the **variety of states** due to the combination of environmental variables, agent location, dynamic interactions and contextual conditions (visibility, collected objects, obstacles overcome). This makes the training slower and dependent on how the network and the learning algorithm are structured.

A second obstacle is represented by the **partiality of the observation**. Agents, without a global map or complete vision, must infer the state of the surrounding world through a local perceptive flow, which can easily mislead in the absence of memory or deduction capacity. In this context, simple reactive policies are often insufficient.

In addition, the presence of **non-stationary dynamics** introduces an additional level of complexity. The boxes, for example, once destroyed by Jason permanently modify the map, altering the strategic possibilities of the Victim. This environmental variability requires the continuous adaptation of strategies, both locally and globally.

Finally, the **competitive and asymmetrical nature** of the context makes the effectiveness of a policy linked to the opponent's behavior. By imposing learning, it takes into account the co-evolution of strategies, implicitly addressing problems of dynamic balance and not behavioral determinism.

# Object Detection

### 1. Implementation in the Game Engine

From an implementation point of view, the game engine provides each agent with a partial snapshot of the surrounding world, represented as a 5x5 grid.

This grid contains the actual image that at that moment is shown on the screen in the visual box of the character ( Image 2.1 ).

The frequency of snapshots has been calibrated based on the effectiveness of Object Detection and the limited time of game computing.

Within our environment, the interaction between agents and the world around them, in fact, is not based on a structural or complete knowledge of the map, but on a visual object detection system, inspired by the perceptual paradigm of real agents.

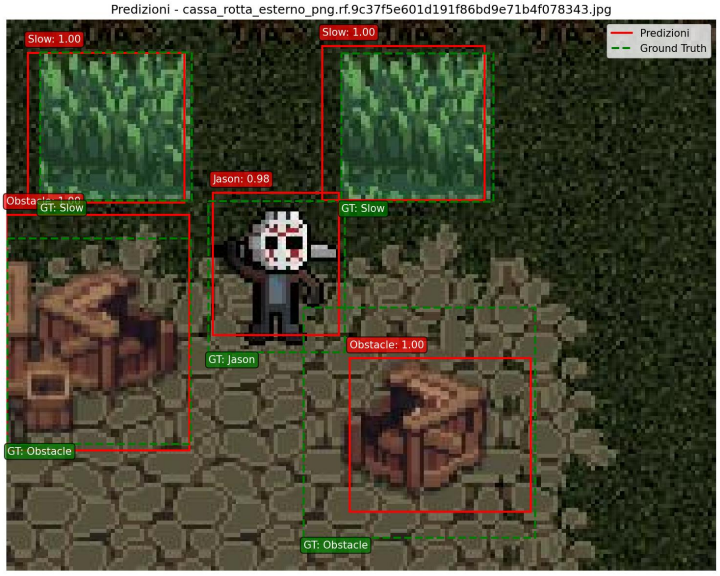



*Image 2.1*

### 2. The Technical Choices

For the object detection activity the **Single Shot Multibox Detector** ( **SSD** ) model was used, chosen for its efficiency and speed.

Unlike other networks, such as R-CNN, SSD allows to simultaneously perform both the generation of region proposals and the classification of bounding boxes in a single step.

This approach is able to guarantee an adequate compromise between accuracy and speed, being particularly effective in detecting multiple objects in a single image.

In particular, the **SSD300** version has been adopted, which accepts in input images of 300x300 pixels. This choice is justified by the fact that the two agents have a field of view limited to images of 100x100 pixels.

The SSD model is based on a pre-trained **VGG16** network, to which additional convolutional layers are added.

These allow you to generate feature maps at different scales, useful for detecting objects of various sizes. For each feature map, the model provides the bounding boxes and the relative class, also performing a regression to refine the position of the boxes.

Once all the predictions are obtained, the boxes with a score below a threshold are eliminated. Next, the **Non-Maximum Suppression** ( **NMS** ) technique is applied to remove duplicate boxes, keeping only those with greater confidence.

#### *2.1. Dataset Creation*

The final model was trained on a dataset composed of 658 images, on which different **data augmentation techniques** were applied, including:

- Horizontal and vertical flip
- 90° rotation ( clockwise and counterclockwise )
- Changing the hue
- Blurring

These transformations led to an expansion of the dataset, which reached a total of **1844 images**. All images are in 100x100 pixel format, representing the field of view of the players.

The objects present have been labeled in various classes:

- Jason
- Victim
- Hide ( hiding places like chests, trunks, beds )
- Obstacle ( tree, bricks, tables )
- Win ( win objects )
- Slow ( obstructing elements such as water and tall grass )

#### *2.2 Training*

The training environment has been configured as follows:

- Epochs number: 500
- Batch size: 8 (per prevenire errori di memoria e ridurre il rumore)
- Learning rate: 0.001
- Optimizer: SGD

For each era, the model calculates the **total loss**, composed of:

- **Localization loss**: measures the difference between the predicted coordinates and the ground truth of the bounding boxes, considering only the positive boxes ( i.e. those that correspond to real objects ).
- **Confidence loss**: evaluates the model's ability to classify each box as an object of a certain class or as a background.

Every 5 epochs, the **validation loss** is calculated, using the same criteria applied to the training phase but on a set of images reserved for validation. Every 10 epochs, the **mean Average Precision** ( **mAP** ) is also calculated, which represents the average of accuracies for each class, obtained by comparing the predictions with the ground truths through the cumulative calculation of **precision** and **recall**.

Summary schemes are provided at the bottom of the page.

### 3. Advantages of the Simulation

The use of Object Detection has proven to be stimulating from a conceptual point of view in effectively simulating a sensory limitation.

First of all, it forces agents to base their decisions on local perceptions, encouraging the emergence of adaptive strategies, such as active exploration or the use of territory to evade the opponent.

Secondly, an implicit form of **noise** is introduced, making training more robust and less subject to overfitting on static maps or perfect observations.

In the images below ( table 2.2 and 2.3 ) is it possible to analyze the training progress.

*Loss*

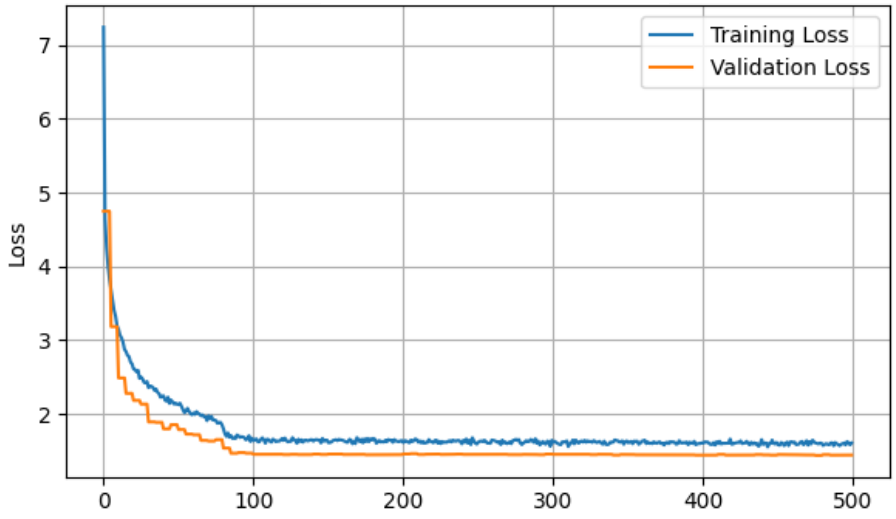



*Table 2.2*

*mAP*

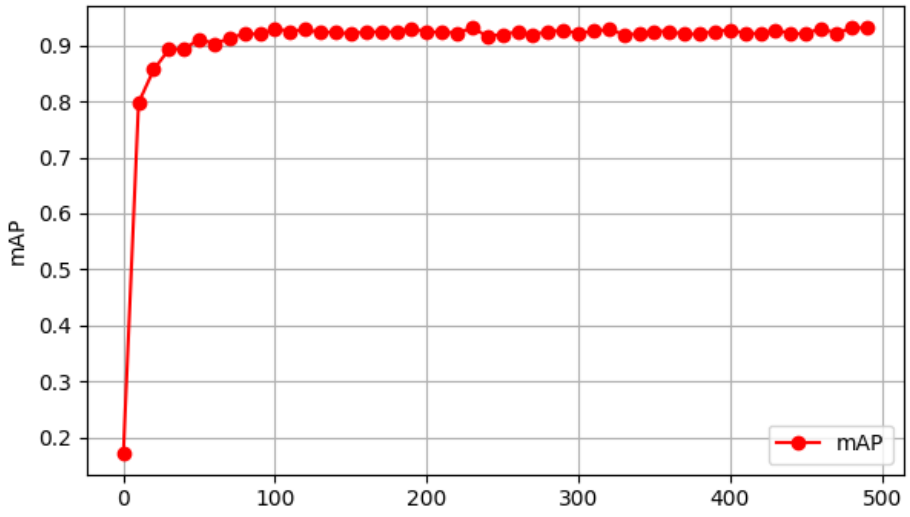



*Table 2.3*

# Reinforcement Learning

### 1. Definition of the Environment and Technical Choices

The environment was modeled based on the **Env interface** of the **Gymnasium library**, and is represented by the **KillerVictimEnv** custom class. Four distinct models have been developed for our two agents, specifically one related to the pursuer and three to the pursued.

At the beginning of the experimentation, only one model was created for the victim. However, this single model converged quickly, focusing only on certain behaviours, and thereforefailing to develop a sufficient generalization.

The structure of the environment provides the following fundamental functions:

- **__init__**: initializes the state variables and configures the interaction mode based on the agent's role.
- **reset**: reinitializes the scenario and returns the agent to the initial state at the beginning of each episode.
- **step**: performs the action decided by the agent, updates the environmental state through object detection, calculates the rewards, and checks termination conditions.
- **close**: releases the resources at the end of the simulation.

The returned observation is structured as an array that represents the objects perceived in the agent's local field of vision, while the available actions correspond to movements and interactions which can be defined as escape or attack behaviours.

Below is a summary table of the state.

| | Values | Description | | Values | Description |
|---|---|---|---|---|---|
| State | “hide”, “interaction”, “visible” | The state the agent is in, if hidden, if he is interacting with an object etc... | Distance_2_y | Float | The y distance from the house with the phone, from the phone or the exit of the house |
| End-game | Boolean | If the agent is in the final game phase | Battery | Boolean | If the agent got the battery ( only victim ) |
| Map | Integer [0..3] | In which map the agent is ( main map or two houses ) | Phone | Boolean | If the agent got the phone ( only victim ) |
| Slow | Boolean | If the agent is in an area that slows down the speed | Killer_visible | Boolean | If the agent sees the killer ( only victim ) |
| Win | Boolean | If the agent has won | Victim_visible | Boolean | If the agent sees the victim ( only killer ) |
| Finished | Boolean | If the game ended with a draw | | | |
| Dead | Boolean | If the victim has been killed | Num_obstacle | Integer [0..10] | Num of obstacles it sees |
| Sub_map | Integer [0..5] | Which area of the map the agent is ( only killer ) | Num_hide | Integer [0..10] | Num of hide spot it sees |
| Distance_1_x | Float | The x distance from the house with the battery, from the battery or the exit of the house | Num_win | Integer [0..1] | Num of win object it sees |
| Distance_1_y | Float | The y distance from the house with the battery, from the battery or the exit of the house | Num_interaction | Integer [0..10] | Num of interactions it sees |
| | | | Num_slow | Integer [0..10] | Num of slow zones it sees |
| Distance_2_x | Float | The x distance from the house with the phone, from the phone or the exit of the house | Agent_x | Float | X value of agent |
| | | | Agent_y | Float | Y value of agent |

As for the architectural choices, our architecture reflects the philosophy of the **Actor-Critic**, it was thought in fact that this could implement more moderate and less noisy updates.

In combination with this structure, **PPO** was then used, with the goal of balanced, and above all, efficient updates.

Below is a more extensive explanation.

### *1.1 Actor-Critic*

The Actor-Critic architecture is one of the fundamental structures in modern reinforcement learning. It is based on two main modules:

- **Actor**, which learns a parameterized policy to decide what action to take in a given state.
- **Critic**, which evaluates the goodness of decisions through a value function, estimating the cumulative reward expected from that state.

In our case, the neural network is initialized with the size of the **game state** and that of the **actions**. Discrete actions favor the transition from the exploration phase, in which the agent tries different options, to the exploitation phase, where he prefers strategies with known performance.

| Number | 0 | 1 | 2 | 3 | 4 | 5 |
|---|---|---|---|---|---|---|
| Relative action | Up | Down | Right | Left | Interact | Dash ( victim ) |

The actor network, defined with PyTorch, does the mapping **game state → action**:

```
self.actor = nn.Sequential (
    nn.Linear(state_dim, 64),          # Input layer
    nn.Tanh(),                         # Activation function
    nn.Linear(64, 64),                 # Hidden layer
    nn.Tanh(),                         # Activation function
    nn.Linear(64, action_dim),         # Output layer
    nn.Tanh()                          # Output in range [-1, 1]
    nn.Softmax(dim=-1)                 # Output in range [0, 1]
)
```

The **Softmax** layer allows probabilistic sampling: for example, with probability [ 0.6, 0.1, 0.3, 0.0 ], the agent will perform the first action 6 times out of 10 and so on. This approach maintains an adequate level of exploration during training, avoiding immediately converging on a deterministic choice.

| Training phase | Actor Behaviour | Example Output |
|---|---|---|
| Start | Uniform probabilities | [0.33, 0.33, 0.33] |
| While | Preferences begin to emerge | [0.1, 0.7, 0.2] |
| End | Dominant actions | [0.05, 0.9, 0.05] |

The **Critic** network, on the other hand, takes the state as input and returns an estimated scalar value ( il valore *V(s)* ):

```
self.critic = nn.Sequential (
    nn.Linear(state_dim, 64),          # Input layer
    nn.Tanh(),                         # Activation function
    nn.Linear(64, 64),                 # Hidden layer
    nn.Tanh(),                         # Activation function
    nn.Linear(64, 1)                   # Output layer —> value
)
```

The Critic makes training more efficient, allowing progressive updates that reduce fluctuations in policy parameters.

Both networks are trained using the **Adam** optimizer, with learning rates respectively of 0.0003 for the Actor and 0.001 for the Critic.

#### *1.2 PPO*

To update the weights the **PPO algorithm** was adopted, composed of two main phases:

- **Collection of experience**, through the RolloutBuffer class, which stores actions, statuses, stock probabilities, rewards, estimated values and end-of-episode reports.
- **Weights update**, carried out by batch size 4096 for the role of "victim" and 256 for the "killer".

Once the experience is collected, it calculates:

- **Return**: cumulative reward expected by each state.
- **Advantage**: difference between return and estimated value; indicates actions better or worse than average.
  - Advantage > 0: action over the average → increases the probability
  - Advantage = 0: no changes
  - Advantage < 0: action below the average → decreases the probability

During each era, the policy ratio is calculated as the ratio between the new and the old probability of the policy. Thanks to the **clipping technique**, the extent of the modification is limited, avoiding overly aggressive updates.

| Ratio value | Meaning | Consequence |
|---|---|---|
| **Ratio = 1.0** | Same policy | No changes |
| **Ratio > 1.0** | More likely action | The policy will encourage this action |
| **Ratio < 1.0** | Less likely action | The policy will discourage this action |
| **Ratio = 2.0** | Doubled probability | Meaningful changes |
| **Ratio = 0.5** | Halved probability | Meaningful changes |

This is the **loss function** used to optimize the policy:

$$LPPO = -min(r_t(\theta) * A_t, clip(r_t(\theta), 1-\varepsilon, 1+\varepsilon) * A_t) + c_1 * L_V F + c_2 * S[\pi_\theta](s_t)$$

| | |
|---|---|
| $r_t(\theta) = \pi_\theta(a_t \mid s_t) / \pi_{\theta old}(a_t \mid s_t)$ | *Policy ratio* |
| $A_t$ | *Advantages* |
| ε | *Clipping parameter ( default value 0.2 )* |
| $L_{VF}$ | *Value function loss* |
| S[π_θ] | *Entropy bonus* |

This formula can be divided into three components:

- **Clipped surrogate loss**, prevents too drastic changes

$$L_{CLIP} = -min(r_t * A_t, clip(r_t, 1-\varepsilon, 1+\varepsilon) * A_t)$$

- **Value function loss**, improves the prediction of the critic

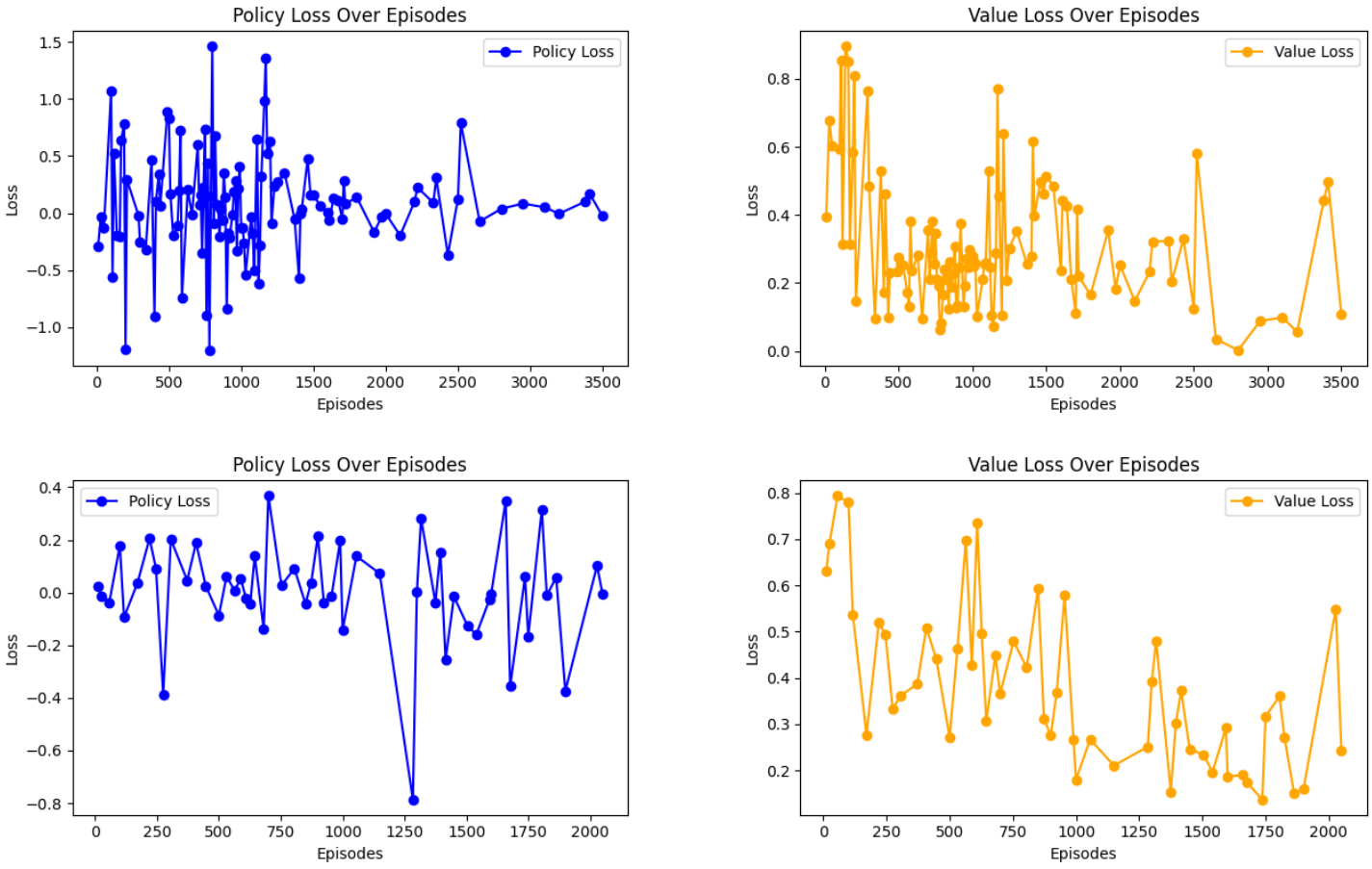
$$L_{VF} = 0.5 * (V_\theta(s_t) - V_{target})^2$$

- **Entropy bonus**, encourages exploration by maintaining the diversified policy

$$S[\pi_\theta] = -\sum \pi_\theta(a \mid s) * log(\pi_\theta(a \mid s))$$

The training parameters include **80 epochs**, the batches described above and an entropy coefficient of **0.01**, balanced between exploration and determinism.

**Value loss** charts show spikes due to variations in the spawn point, but converge stably towards the end of training. This indicates that the agent ( victim or killer ) learns a coherent and well-defined strategy.




## 2. Reward Strategy

The definition of the reward function represents one of the most delicate and central aspects in any reinforcement learning system, especially in complex and multi-agent environments such as the one developed.

In this context, it was decided to assign intermediate and specific rewards for the various roles, explicitly guiding, albeit in part, the learning of the distinct behaviors of each individual agent in line with the philosophy of **reward shaping**.

The Victim agent in particular saw several phases of training each specific to the model that was intended to be developed, with training sub-steps based on the **time-based curriculum learning technique**.

As a result, the reward system has rewarded exploration, prudence, strategic use of racing and effective interaction with the environment from time to time.

On the contrary, any static, risky or ineffective behavior, such as prolonged stasis or direct exposure to the killer, has been penalized.

For the Killer agent, on the other hand, the reward function is aimed at maximizing efficiency in the search and capture of the victim, rewarding timely identification and pursuit, interaction with useful objects, and penalizing slow decision-making or improper use of interactions, these in fact required a very expensive execution time if used inappropriately.

In both cases, the system was built in such a way as to encourage, in addition to achieving the final goal, also the maintenance of effective and partly realistic behavior throughout the episode.

### 3. Object Detection Difficulties

Some of the problems encountered were the limited field of vision, which often caused a delay in information in which the agents found themselves acting without having the full data on the environment or position of the opponent ( making it more difficult to learn effective reactive policies ).

Or again, the perceptual grid representation, despite being structurally simple, does not provide direct indications on the global topology of the map, nor on the spatial relationship between visible and off-screen elements. This requires agents to infer the entire environmental structure indirectly, often through the accumulation of episodic experience.

Moreover the dynamism of the observations, related also to the behaviour of the other agent, generates a highly non-stationary distribution of the observed states, complicating the convergence of the learned strategies. To which is added the need to manage perceptual ambiguities, such as the presence of partially obscured obstacles.

Despite these complexities, the object detection-based approach represents an effective compromise between perceptive fidelity and computational accessibility, constituting a generalized basis for learning intelligent behaviors in realistic and poorly structured environments.

### 4. Conclusions

The use of Reinforcement Learning has made it possible to generate non-trivial and adaptive strategies, which reflect the role and sensory capacities of the agents.

In particular, the combination of PPO, partial vision via object detection and a targeted reward shaping has proven effective in promoting realistic behaviors such as chase, circumvention, escape and hiding.

However, the presence of partial and dynamic observations led to greater instability during the learning phase, making it necessary to accurately tune the parameters and define progressive rewards.

Overall, the implemented pipeline forms a solid foundation for future developments, such as the integration of episodic memory, shared policy models or the introduction of communication mechanisms between agents.