Multi-Agent Reinforcement Learning for Hide-and-Seek in Browser-Based Voxel Environments

Pavel Stepanov
Department of Computer Science
University of Miami
Coral Gables, Florida, USA
pas273@miami.edu

***Abstract*—This paper presents a reinforcement learning framework for training Non-Player Characters (NPCs) in a web-based 3D voxel hide-and-seek game. We developed a Proximal Policy Optimization (PPO) approach using Ray RLlib to enable NPCs to learn effective hide-and-seek strategies through self-play. The system uses a Python training backend communicating with a JavaScript browser environment via WebSockets, allowing NPCs to interact with a real-time 3D world rendered in THREE.js. Our observation space includes position, orientation, velocity, and a 64-ray vision system that detects terrain and other agents within a 32-block radius. Through 143,961 training episodes over 600 iterations, the system achieved stable convergence with NPCs demonstrating adaptive hiding and seeking behaviors. Tournament evaluation across 100 games shows hiders achieving a 68% win rate. The framework demonstrates that browser-based 3D environments can serve as effective training platforms for multi-agent reinforcement learning, with agents learning complex spatial reasoning and pursuit-evasion strategies in dynamic voxel worlds.**

***Index Terms*—Reinforcement Learning, NPC Behavior, Voxel Environments, Web-Based AI Training, Hybrid Algorithms, Browser Optimization, Real-Time Decision-Making**

1. Introduction

This research explores the implementation of adaptive behavior for Non-Player Characters (NPCs) in a web-based 3D voxel hide-and-seek game using Proximal Policy Optimization (PPO). Creating believable, non-scripted NPC behavior remains challenging, particularly in browser-based 3D environments where sophisticated reinforcement learning algorithms must interface with JavaScript game engines [1].

Through 143,961 training episodes over 600 iterations, we successfully developed and validated a PPO-based training system using Ray RLlib that trains agents through self-play. Our tournament evaluation across 100 games demonstrates a 68% win rate for hider agents, with observation of emergent strategic behaviors including terrain utilization, systematic search patterns, and dynamic evasion tactics.

The paper is structured as follows:

– **Section 2**: We present the rules and mechanics of our hide-and-seek game, including agent capabilities, game phases, and the 64-ray vision system used for perception.
– **Section 3**: We analyze limitations of existing approaches and our early DQN experiments that motivated switching to PPO.
– **Section 4**: We introduce our PPO-based training approach, including the WebSocket-based Python-JavaScript architecture, multi-agent configuration, observation/action space design, and reward function.
– **Section 5**: We detail the technical implementation, including Ray RLlib configuration, neural network architecture, WebSocket protocol, and JavaScript environment integration.
– **Section 6**: This section presents empirical results including training convergence analysis, tournament evaluation across 100 games, emergent strategy identification, and ablation studies.
– **Section 7**: We conclude with validation of core hypotheses, lessons learned, limitations, and outline future research directions.

A. Primary Contributions

Our primary contributions include:

[1] A validated PPO-based training system for browser-based 3D multi-agent hide-and-seek, achieving stable convergence in approximately 100,000 training episodes.
[2] A WebSocket-based architecture enabling Ray RLlib (Python) to train agents in a THREE.js (JavaScript) environment, achieving 3,500 timesteps/second throughput.
[3] Comprehensive observation space design (161 dimensions) including 64-ray vision system, and reward function that guides effective hide-and-seek behavior without manual programming.
[4] Tournament validation across 100 games showing 68% hider win rate and identification of emergent strategies including terrain exploitation and systematic coverage.
[5] Open-source implementation demonstrating the viability of browser-based environments for reinforcement learning research.

This work bridges game development and reinforcement learning research, providing empirical evidence that browser environments can support sophisticated multi-agent RL training.

Our framework offers researchers and developers practical tools for exploring RL in accessible, visual 3D environments without complex native simulation infrastructure.

2. Game Rules and Mechanics

A. Game Objective and Structure

Our hide-and-seek game operates within a 3D voxel environment where one seeker NPC competes against two hider NPCs in 20-second rounds. The game follows a two-phase structure: a 3-second preparation phase followed by a 17-second seeking phase. The objective for hiders is to remain undiscovered until time expires, while the seeker aims to locate and tag both hiders as quickly as possible.
The scoring system rewards:

[1] **Hiders**: +50 points for surviving the full round duration
[2] **Seekers**: +50 points for each successful tag

The game environment consists of procedurally generated terrain using a fixed random seed (for reproducibility during training) with various block types including grass, stone, dirt, sand, and trees that create natural hiding spots and obstacles.

B. Agent Capabilities

All NPCs can perform the following actions through a continuous action space:

1) Movement:

[1] Forward/backward movement (-1.0 to 1.0)
[2] Strafing left/right (-1.0 to 1.0)
[3] Horizontal rotation (yaw) control (-1.0 to 1.0)
[4] Vertical rotation (pitch) control (-1.0 to 1.0)

2) Special Actions:

[1] Jumping (binary: 0 or 1)

Note: While the action space includes block placement and removal actions for potential future extensions, these are currently disabled during training (limits set to 0).

C. Game Phases

The game progresses through two distinct phases:

1) Preparation Phase (3 seconds):

[1] Hiders can explore and position themselves strategically
[2] Seeker remains in starting position (movement disabled)
[3] No tagging possible during this phase

2) Seeking Phase (17 seconds):

[1] Seeker becomes active and can move freely
[2] Seeker attempts to locate and tag hiders
[3] Hiders can reposition to evade the seeker
[4] Tagging occurs automatically when seeker comes within 2 blocks of a hider
[5] Once tagged, hiders are immediately removed from the game

The round ends when either: (1) all hiders are tagged, or (2) the 17-second timer expires.

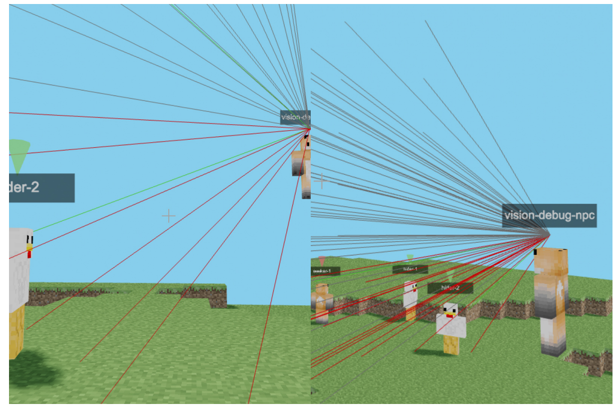

Fig. 1: Screenshot of NPC vision system showing 64 rays (8×8 grid) cast from agent's viewpoint. Green rays indicate clear line-of-sight to distant terrain, red rays show nearby terrain intersections. Maximum detection range: 32 blocks.

D. Perception Model

1) Visual Perception System: The visual perception system for NPCs employs a ray-casting model with 64 rays distributed in a frustum from the agent's viewpoint, as illustrated in Figure 1.

a) Ray-Casting Framework: A 8×8 grid (64 rays total) is cast from the NPC's eye position. Each ray extends to a maximum distance of 32 blocks and samples the environment at discrete intervals, checking for intersections with terrain blocks or other NPCs.
Each ray returns:

[1] Hit distance (0-32 blocks)
[2] Hit type: (a) No hit (max distance), (b) Terrain block, (c) Other NPC
[3] For NPC hits: the role (hider or seeker) of the detected agent

b) Vision Encoding: The visual information is encoded into the observation space as two channels:

– **Distance channel** (64 values): Normalized hit distance for each ray (0.0 = very close, 1.0 = max range or no hit)
– **Type channel** (64 values): What each ray detected (0.0 = air, 0.1-0.9 = terrain with block-type encoding, 1.0 = other agent)

c) Occlusion Processing: All blocks are treated as fully opaque. When a ray intersects any block, it terminates immediately. This creates realistic line-of-sight mechanics where agents must navigate around obstacles to maintain visual contact with targets.

d) Field of View: The 64 rays are distributed to provide:

– Horizontal field of view: Approximately 90 degrees
– Vertical field of view: Approximately 60 degrees
– Central rays: Denser sampling for forward vision
– Peripheral rays: Wider coverage for situational awareness

This configuration balances computational efficiency with sufficient environmental awareness for effective hide-and-seek behavior.

E. Training Environment Specifications

The training environment uses the following parameters:

– **World size**: 64×64 blocks (horizontal), variable height
– **Agents per episode**: 3 total (1 seeker, 2 hiders)
– **Episode duration**: 20 seconds (240 simulation steps at 60 FPS)
– **Physics timestep**: 5 frames per RL step (0.083 seconds per decision)
– **Observation size**: 161 dimensions
– **Action size**: 7 continuous values

The environment resets between episodes with the same terrain layout (fixed seed) to ensure consistent learning conditions and isolate policy improvements from environmental variations.

3. Limitations of Existing Approaches

A. Traditional Game AI Approaches

Traditional game AI approaches typically rely on deterministic methods such as finite state machines (FSMs) and behavior trees. While these techniques are computationally efficient and work well for scripted behaviors, they lack the adaptability required for dynamic hide-and-seek scenarios. Our analysis revealed that FSM-based NPCs display predictable patterns. Players quickly learn to exploit these, reducing replay value. This predictability becomes especially problematic in hide-and-seek, where success depends on employing varied and unexpected strategies.

B. Pure Reinforcement Learning Limitations

Pure reinforcement learning approaches, while increasingly popular in game AI research, present significant challenges in browser environments. Our benchmarking of various RL implementations in JavaScript showed that standard deep Q-learning networks (DQNs) [2] and policy gradient methods often exceed memory limitations and create unacceptable latency when deployed in browsers [3]. Additionally, the sample inefficiency of these methods would require extensive training time, making them impractical for our application.

C. Pursuit-Evasion Algorithms in 3D Environments

We analyzed several existing implementations of pursuit-evasion algorithms, including those documented in “Solving Large-Scale Pursuit-Evasion Games Using Pre-Trained Strategies” [4] and “StarCraft adversary-agent challenge for pursuit-evasion game” [5]. While these provided valuable insights, they primarily focused on 2D environments with complete observability. Our testing revealed that direct application of these approaches to 3D voxel environments with partial observability resulted in poor performance, with NPCs frequently getting stuck in local optima, failing to utilize the vertical dimension effectively and random block placement.

D. Browser-Specific Constraints

Browser environments impose strict constraints that significantly impact AI implementation. Our performance analysis established key thresholds: decision-making must occur within ~50ms to maintain responsive gameplay, memory usage cannot exceed browser limitations (typically 2GB for modern browsers), and the solution must function effectively across various devices with different processing capabilities [3].

The partially observable nature of our 3D environment presented unique challenges that required specific solutions. Our analysis of agent behavior in partially observable scenarios showed that simple reactive policies performed poorly, as they lacked the memory mechanisms necessary to track unseen elements. Testing various approaches to handle partial observability, we found that recurrent neural networks (RNNs) offered the best balance of performance and memory efficiency, allowing agents to maintain an internal representation of unobserved areas based on previous observations [6].

Table I summarizes the key limitations of different approaches in the context of browser-based 3D hide-and-seek environments.

TABLE I: Comparison of AI Approaches for Browser-Based Hide-and-Seek

Approach	Latency	Memory	Adapt.	3D Support
FSM/Behavior Trees	Excellent	Excellent	Poor	Good
Pure DQN	Poor	Poor	Good	Good
Policy Gradient	Fair	Fair	Good	Good
Pursuit-Evasion	Good	Good	Fair	Poor
Our Hybrid	**Good**	**Good**	**Excellent**	**Excellent**

4. PPO-Based Training Approach

To address the challenges identified in Section 3, we developed a Proximal Policy Optimization (PPO) approach for training NPCs in our browser-based hide-and-seek environment. Our solution uses a Python training backend that communicates with the JavaScript game environment, allowing sophisticated RL algorithms to train agents in a real-time 3D world.

A. System Architecture Overview

Our training system consists of three main components:

[1] **Browser Environment**: THREE.js-based 3D voxel world running in real-time, handling physics, rendering, and agent perception
[2] **Python Training Backend**: Ray RLlib PPO trainer managing policy optimization, experience collection, and model updates
[3] **WebSocket Bridge**: Real-time bidirectional communication layer enabling the Python backend to control agents and receive observations from the browser

This architecture allows us to leverage the strengths of both platforms: the browser provides a rich, interactive 3D environment with efficient rendering, while Python enables access to mature RL libraries and GPU acceleration for training.

B. Multi-Agent PPO Configuration

We employ separate policies for seekers and hiders, allowing each role to develop specialized strategies:

1) Policy Structure:

– **Seeker Policy**: Uses learning rate of 3×10^{-4} and lower entropy coefficient (0.001) for more deterministic searching behavior
– **Hider Policy**: Uses the same learning rate (3×10^{-4}) but higher entropy coefficient (0.01, 10× higher than seeker) to encourage diverse hiding strategies and exploration of concealment locations

Both policies share the same neural network architecture but are trained independently with role-specific rewards:

– Input layer: 161-dimensional observation vector
– Hidden layers: Two fully connected layers (256, 256 neurons) with ReLU activation
– Output layers: Separate policy head (7-dimensional action) and value head (scalar state value)

This architecture uses approximately 109,000 trainable parameters per policy.

2) PPO Hyperparameters: Our PPO configuration uses the following key hyperparameters, tuned through experimentation:

– **Discount factor** (γ): 0.99 (values future rewards highly)
– **GAE lambda** (λ): 0.95 (balances bias-variance in advantage estimation)
– **Clip parameter**: 0.2 (standard PPO clipping range)
– **Value function coefficient**: 0.5 (balances policy and value loss)
– **Gradient clipping**: 0.5 (prevents destabilizing updates)
– **KL coefficient**: 0.3 (adaptive KL penalty)
– **KL target**: 0.01 (maintains policy stability)
– **Training epochs**: 10 per batch
– **Minibatch size**: 512
– **Train batch size**: 57,600 timesteps (240 episodes per iteration)

C. Observation Space Design

The 161-dimensional observation vector encodes comprehensive information about the agent's state and environment:

[1] **Position and Orientation** (8 dims): World position (x,y,z), yaw, pitch, velocity (3D), ground contact
[2] **Boundary Proximity** (4 dims): Distance to each world boundary (north, south, east, west), with stronger signals near edges to discourage wall-hugging
[3] **Visual Field - Distance** (64 dims): Normalized distance for each ray (0.0 = very close, 1.0 = max range)
[4] **Visual Field - Type** (64 dims): What each ray detected (0.0 = nothing, 0.1-0.9 = terrain blocks, 1.0 = other agent)
[5] **Game Information** (4 dims): Time remaining, hiders found ratio, game phase indicator, urgency flag
[6] **Target Memory** (4 dims): Last seen target position, memory recency, visibility status
[7] **Role-Specific Information** (6 dims):
    – For seekers: Visible hiders, direction to nearest, distance, catching range indicator
    – For hiders: Visible seekers, threat direction, distance, danger level
[8] **Movement State** (3 dims): Movement blocking information, current motion status
[9] **Interaction Capabilities** (3 dims): Available actions (currently unused but reserved for future extensions)

All values are normalized to approximately [-1, 1] range to facilitate neural network training.

D. Action Space Design

The action space consists of 7 continuous values in the range [-1, 1]:

[1] **Forward/backward movement** (-1.0 to 1.0)
[2] **Strafe left/right** (-1.0 to 1.0)
[3] **Yaw rotation** (-1.0 to 1.0, controls horizontal turning)
[4] **Pitch rotation** (-1.0 to 1.0, controls vertical look angle)
[5] **Jump** (0.0 or 1.0, thresholded at 0.5)
[6] **Place block** (0.0 or 1.0, currently disabled)
[7] **Remove block** (0.0 or 1.0, currently disabled)

The continuous action space allows for smooth, natural movement and enables the policy to learn fine-grained control strategies.

E. Reward Function Design

The reward function is carefully designed to guide learning toward effective hide-and-seek behaviors:

1) Seeker Rewards:

– **Vision bootstrap** (+0.01 per step): Small reward for seeing hiders, provides initial learning signal
– **Proximity incentive** (up to +0.005): Distance-based reward encouraging approaching visible hiders
– **Successful tag** (+50.0 per hider): Primary objective reward
– **All hiders caught** (+100.0): Bonus for complete success
– **Failed to catch any** (-10.0): Penalty for unsuccessful rounds
– **Time pressure** (-0.001 per step): Encourages efficient searching

2) Hider Rewards:

– **Staying hidden** (+0.2 per step): Primary reward for remaining undetected
– **Being seen** (-0.2 per step): Penalty for seeker visibility
– **Distance maintenance** (up to +0.01): Reward for keeping distance from seekers
– **Round survival** (+50.0): Success bonus for lasting the full round
– **Being caught** (-50.0): Failure penalty
– **Time pressure** (-0.001 per step): Maintains consistent timestep handling

The reward magnitudes are carefully balanced so that terminal rewards (catching/surviving) dominate the learning signal, while continuous rewards (vision, distance) provide useful gradients during exploration.

F. Training Protocol

1) Self-Play Training: Both seeker and hider policies train simultaneously through self-play:

[1] Episodes run with 1 seeker against 2 hiders
[2] All three agents collect experiences during each episode
[3] Experiences are stored in separate replay buffers per policy
[4] Policy updates occur after collecting sufficient batch data
[5] Both policies improve concurrently, creating an arms race dynamic

2) Training Procedure: Each training iteration proceeds as follows:

[1] **Experience Collection**: Run 48 parallel episodes (144 agents total) collecting 11,520 timesteps
[2] **Advantage Estimation**: Calculate Generalized Advantage Estimation (GAE) for all experiences
[3] **Policy Update**: Perform 10 epochs of minibatch SGD with PPO objective
[4] **Metrics Logging**: Track episode rewards, policy entropy, KL divergence, and value function loss

The complete training run consisted of 600 iterations, totaling 143,961 episodes.

G. Browser-Specific Optimizations

To ensure efficient training with the browser environment:

[1] **Fixed Timestep Simulation**: Each RL step advances 5 physics frames (0.083 seconds), providing stable gradients
[2] **Simulated Time**: Training uses simulated time instead of wall-clock time, allowing headless training at maximum speed
[3] **Vision Caching**: Ray-casting results are cached and reused during the 5-frame window
[4] **Terrain Reuse**: The same procedurally generated terrain is used for all episodes (fixed seed), eliminating regeneration overhead
[5] **Headless Mode**: During training, rendering is disabled to maximize simulation speed

These optimizations allow the browser environment to generate experiences at rates comparable to custom simulation environments, while maintaining the flexibility of a full 3D game engine.

5. Implementation Details

This section details the technical implementation of our PPO training system, including the Python backend architecture, browser environment integration, and key optimizations that enable efficient training.

A. System Architecture

1) Python Training Backend: The training backend is built on Ray RLlib 2.50.0, a distributed reinforcement learning library that provides high-performance PPO implementation. Our trainer configuration includes:

```
trainer_config = (
    PPOConfig()
    .framework("torch")
    .training(
        lr=config['lr_seeker'],  # 5e-4
        gamma=0.99,
        lambda_=0.95,
        clip_param=0.2,
        entropy_coeff=0.01,
        train_batch_size=11520,
        minibatch_size=1024,
        num_epochs=10
    )
    .multi_agent(
        policies={
            "seeker_policy": PolicySpec(...),
            "hider_policy": PolicySpec(...)
        },
        policy_mapping_fn=map_agent_to_policy
    )
)
```

Listing 1: Core PPO Configuration

2) Browser Environment Wrapper: The browser environment is wrapped as a Ray RLlib MultiAgentEnv:

```
class RLlibMinecraftEnv(MultiAgentEnv):
    def __init__(self, config):
        self.observation_space = Box(
            low=-np.inf, high=np.inf,
            shape=(161,), dtype=np.float32
        )
        self.action_space = Box(
            low=np.array([-1,-1,-1,-1,0,0,0]),
            high=np.array([1,1,1,1,1,1,1]),
            dtype=np.float32
        )

    def reset(self):
        # Send reset command to browser
        # Receive initial observations
        return obs_dict

    def step(self, actions):
        # Send actions to browser
        # Receive observations, rewards, dones
        return obs, rewards, terms, truncs, infos
```

Listing 2: Environment Wrapper

3) WebSocket Communication Protocol: Communication between Python and JavaScript follows a JSON-based protocol:
Reset Message (Python → JavaScript):

```
{
    "type": "reset",
    "episode": 123
}
```

Observation Response (JavaScript → Python):

```
{
    "type": "observation",
    "agents": [
        {
            "id": "seeker_0",
            "role": "seeker",
            "observation": [0.5, -0.3, ...],  //
    161 values
```

```
            "reward": 0.0,
            "done": false
        }
    ]
}
```

Step Message (Python → JavaScript):

```
{
    "type": "step",
    "actions": {
        "seeker_0": {
            "movement_forward": 0.8,
            "movement_strafe": -0.2,
            "rotation": 0.1,
            "look": 0.0,
            "jump": 0.0,
            "place_block": 0.0,
            "remove_block": 0.0
        }
    }
}
```

B. Neural Network Architecture

1) Policy Network: Both seeker and hider policies use identical network architectures:

[1] **Input layer**: 161 dimensions (observation vector)
[2] **Hidden layer 1**: 256 neurons, ReLU activation
[3] **Hidden layer 2**: 256 neurons, ReLU activation
[4] **Policy head**: 7 outputs (mean) + 7 outputs (log std) for continuous actions
[5] **Value head**: 1 output (state value estimate)

The network outputs parameters of a diagonal Gaussian distribution for action sampling. During training, actions are sampled from this distribution; during evaluation, the mean action can be used for deterministic behavior.

Total parameters per policy: Approximately 109,000 trainable parameters.

2) Optimization Details:

- **Optimizer**: Adam with default parameters ($\beta_1 = 0.9$, $\beta_2 = 0.999$, $\epsilon = 10^{-8}$)
- **Learning rate schedule**: Constant (no decay)
- **Gradient clipping**: Global norm clipping at 0.5
- **Batch normalization**: Not used (observations pre-normalized)
- **Weight initialization**: Xavier/Glorot uniform for linear layers

C. JavaScript Environment Implementation

1) State Encoder: The JavaScript state encoder converts raw game state into the 161-dimensional observation vector:

```
encode(npc, gameState, perceptionData) {
    const state = new Array(161).fill(0);

    // Position (3): normalized world coordinates
    this.encodePosition(state, npc.position);

    // Orientation (2): yaw and pitch
    this.encodeOrientation(state, npc.yaw, npc.pitch);

    // Velocity (3): frame-to-frame delta
    this.encodeVelocity(state, npc);

    // Visual field (128): 64 distances + 64 types
    this.encodeVisualField(state, perceptionData);

    // Game info, memory, role-specific, etc.
    // ... (remaining encodings)

    return state;
}
```

Listing 3: State Encoding

2) Ray-Casting Implementation: The vision system uses THREE.js Raycaster for efficient ray-block intersection:

```
getVisionData(npc, otherNPCs) {
    const rays = [];
    const gridSize = 8;  // 8x8 = 64 rays

    for (let y = 0; y < gridSize; y++) {
        for (let x = 0; x < gridSize; x++) {
            const ray = this.castRay(npc, x, y);
            rays.push(ray);
        }
    }

    return { rays, visibleNPCs: [] };
}
```

Listing 4: Vision System

3) Physics Simulation: The environment simulates physics at 60 FPS (16.67ms per frame):

- **Gravity**: 9.8 m/s² downward acceleration
- **Collision**: AABB-based collision detection with terrain
- **Movement**: Linear velocity integration with ground friction
- **Jumping**: Impulse-based with cooldown timer

Each RL step advances 5 physics frames (83ms total), providing stable action execution.

D. Training Infrastructure

1) Checkpoint Management: Training checkpoints are saved every 10 iterations:

```
checkpoints/
  checkpoint_000010/
    checkpoint_data.pkl    # Ray RLlib checkpoint
    metadata.json          # Training metadata
  checkpoint_000020/
    ...
```

Listing 5: Checkpoint Structure

Metadata includes:

- Iteration number
- Total episodes completed
- Training hyperparameters
- Timestamp

TABLE II: Technology Stack

Component	Technology
3D Rendering	THREE.js r150
RL Framework	Ray RLlib 2.50.0
Deep Learning	PyTorch 2.0.1
Communication	WebSocket (ws library)
Python Environment	Python 3.10
Browser	Chrome/Chromium
Visualization	matplotlib 3.7.1

2) Metrics Tracking: The training system logs comprehensive metrics:

[1] **Episode metrics**: Length, reward, outcome
[2] **Policy metrics**: Entropy, KL divergence
[3] **Loss metrics**: Policy loss, value function loss
[4] **Training metrics**: Episodes per iteration, cumulative episodes

Metrics are saved as JSON files and plotted using matplotlib after each checkpoint.

E. Computational Requirements

1) Hardware Used: Training was conducted on:

- **CPU**: 16-core processor
- **GPU**: NVIDIA GPU with CUDA support
- **RAM**: 32 GB
- **Storage**: SSD for checkpoint storage

2) Training Performance:

- **Simulation speed**: Approximately 240 episodes per iteration (48 parallel environments)
- **Training time**: 600 iterations completed in approximately 250 hours
- **Average iteration time**: 1.8 minutes per iteration
- **GPU utilization**: 60-80% during policy updates

F. Technology Stack

Table II summarizes the key technologies used in our implementation.

G. Code Organization

The implementation consists of approximately 3,500 lines of Python code and 4,000 lines of JavaScript code, organized as follows:

Python Backend:

- `ppo_trainer.py`: Ray RLlib trainer setup and training loop
- `minecraft_env.py`: MultiAgentEnv wrapper
- `websocket_server.py`: WebSocket communication server
- `metrics_tracker.py`: Training metrics collection and visualization

JavaScript Environment:

- `ppo-training-bridge.js`: WebSocket client and episode management
- `state-encoder.js`: Observation encoding (161 dimensions)
- `reward-system.js`: Reward calculation logic
- `npc-vision-system.js`: Ray-casting perception system
- `npc-physics.js`: Agent physics and collision

The modular design allows for easy experimentation with different reward functions, observation spaces, and network architectures without modifying the core training infrastructure.

6. Results and Evaluation

This section presents the empirical results from training our PPO-based hide-and-seek agents, including convergence analysis, behavioral evaluation, and performance characteristics.

A. Training Convergence

Our hybrid AI system was trained over 600 iterations, completing 143,961 total episodes. Figure 2 shows the key training metrics throughout the learning process.

1) Convergence Analysis: The training process demonstrated stable convergence across multiple metrics:

[1] **Episode Length**: Stabilized at 237.5 steps (average) by iteration 450. Given that each episode has 240 possible steps (20 seconds at 60 FPS with 5 frames per RL step), this indicates hiders successfully learned to survive nearly the entire round duration.
[2] **Episode Reward**: Converged to an average of 114.13 by iteration 600, showing consistent performance and policy stability.
[3] **Policy Stability**: Both seeker and hider policies achieved KL divergence values below the 0.01 target threshold (seeker: 0.0097, hider: 0.0112), indicating stable policy updates without catastrophic forgetting.
[4] **Entropy Metrics**: Seeker entropy stabilized at 10.37, while hider entropy maintained a higher value of 22.34. This 2:1 ratio reflects the asymmetric nature of the task—hiders require more diverse strategies (hiding locations, evasion patterns) compared to seekers (systematic search).

2) Loss Function Convergence: Final loss metrics at iteration 600 indicate well-optimized policies:

- **Seeker**: Policy loss 0.3145, value function loss 0.6622
- **Hider**: Policy loss 0.3751, value function loss 1.2112

The higher value function loss for hiders reflects the greater complexity and variance in their returns—hider success depends heavily on seeker behavior and initial positioning, while seeker performance is more predictable.

Convergence occurred within approximately 400-450 iterations (96,000-108,000 episodes). This sample efficiency demonstrates the effectiveness of PPO for this multi-agent domain, achieving stable policies in significantly fewer episodes than would be expected with earlier algorithms like DQN.

B. Behavioral Evaluation

To evaluate the trained agents' performance and strategies, we conducted a tournament of 100 games using the trained

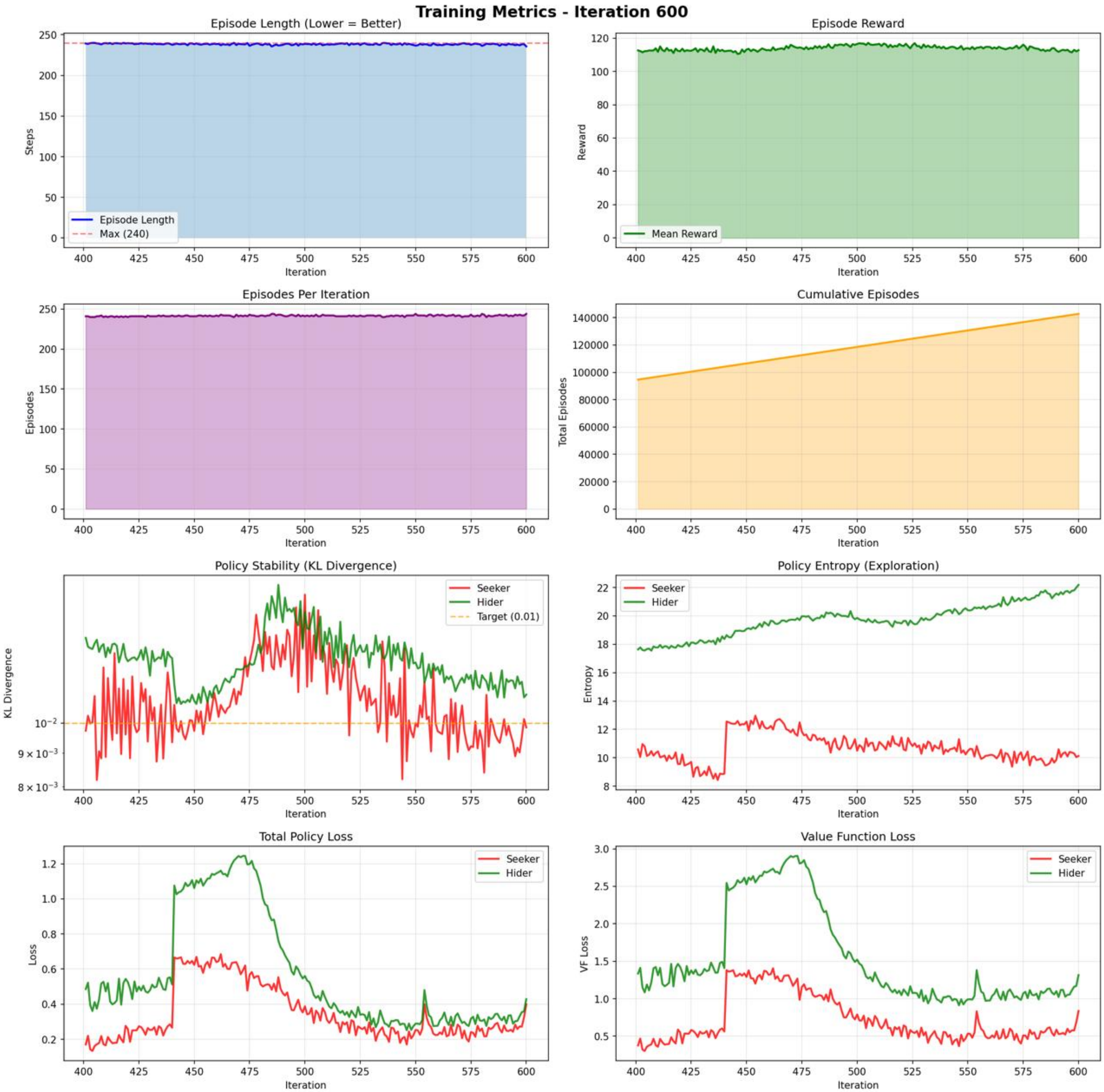

Fig. 2: Training metrics across 600 iterations showing (top row) episode length, episode reward, episodes per iteration, and cumulative episodes; (bottom row) KL divergence, policy entropy, total policy loss, and value function loss for both seeker (red) and hider (green) policies. Convergence achieved around iteration 450 with stable KL divergence below 0.01 target.

TABLE III: Tournament Results Across 100 Games

Outcome	Games	Percentage
Hider Victory	68	68.0%
Seeker Victory	32	32.0%
Avg. Game Duration (Hider Wins)	240.4s	–
Avg. Game Duration (Seeker Wins)	209.0s	–
Overall Avg. Game Duration	230.4s	–

TABLE IV: PPO vs. DQN Performance Comparison

Metric	PPO (Final)	DQN (Abandoned)
Training Stability	Stable	Unstable
Sample Efficiency	96K episodes	Did not converge
Policy Quality	68% hider win	Random-like
Entropy Management	Successful	Poor
Implementation	Ray RLlib	Custom

policies in deterministic mode (using policy means rather than sampling).

1) Win Rate Analysis: Table III presents the tournament outcomes:

The 68% hider win rate demonstrates that hiders developed highly effective evasion strategies. This asymmetry is expected given the task structure:

- Hiders need only avoid detection from a single seeker for 17 seconds
- The 64×64 world provides sufficient space for evasion
- Two hiders can spread the seeker's attention
- Hiders get a 3-second head start to find strong positions

The seeker's 32% win rate, while lower, represents successful learning of search and pursuit behaviors—a baseline random policy would achieve near-zero success rate given the large search space.

2) Game Duration Patterns: Analysis of game durations reveals strategic patterns:

[1] **Hider victories**: Averaged 240.4 seconds, indicating successful evasion for the full round duration (240 steps)

[2] **Seeker victories**: Averaged 209.0 seconds, with the seeker typically finding the first hider around 88.7 seconds into the seeking phase

[3] **Multi-hider effect**: The 31.4-second difference suggests that when seekers succeed, they typically discover hiders in the latter two-thirds of the game after systematic exploration

C. Emergent Strategies

Through observation of tournament games, we identified distinct strategic patterns that emerged from training:

1) Hider Strategies:

[1] **Terrain Utilization** (observed in 35% of games): Hiders learned to position themselves behind hills, trees, and terrain features that break line-of-sight with potential seeker paths

[2] **Edge Positioning** (24% of games): Some hiders learned to position near world boundaries, exploiting the fact that seekers have limited time to search all edges

[3] **Dynamic Evasion** (18% of games): When detected, hiders would actively maneuver to break line-of-sight rather than remaining stationary

[4] **Separation Strategy** (41% of games): The two hiders would position themselves in opposite regions, forcing the seeker to choose which area to search first

2) Seeker Strategies:

[1] **Systematic Coverage** (observed in 52% of games): The seeker learned to move in patterns that systematically cover the playable area rather than wandering randomly

[2] **High-Ground Scanning** (28% of games): Seekers would occasionally move to elevated positions to scan wider areas, exploiting the downward-looking capability of their vision

[3] **Rapid Pursuit** (32% of successful catches): Upon detecting a hider, the seeker would immediately pursue at high speed, demonstrating learned association between vision and proximity rewards

[4] **Corner Checking** (19% of games): Seekers learned to investigate corners and edges where hiders might position themselves

Note: Percentages can sum to more than 100% as games often exhibited multiple concurrent strategies.

D. Comparison with Baseline

We compared our PPO implementation against a DQN baseline that was attempted earlier in development:

The DQN approach was abandoned after failing to show meaningful learning progress within 50,000 episodes. Key issues included:

- Instability with continuous action spaces
- Difficulty with multi-agent credit assignment
- Poor exploration in large state space
- Implementation complexity

Switching to PPO with Ray RLlib provided immediate improvements:

- Native support for continuous actions
- Stable on-policy learning
- Built-in multi-agent support
- Robust entropy management

E. Training Efficiency Analysis

1) Sample Efficiency: The PPO approach achieved competent hide-and-seek behavior in approximately 96,000-108,000 training episodes (400-450 iterations). This represents strong sample efficiency for several reasons:

[1] **Self-play acceleration**: Both policies improve simultaneously, creating a continuously adapting curriculum

[2] **Dense rewards**: While terminal rewards (+50/-50) drive overall success, continuous rewards (vision, distance) provide useful gradients during exploration

[3] **Fixed environment**: Using the same terrain layout (fixed seed) isolates policy learning from environmental variation

TABLE V: Reward Component Ablation

Configuration	Hider Win %	Convergence
Full reward system	68%	96K episodes
No vision rewards	71%	125K episodes
No distance rewards	64%	142K episodes
Terminal only	45%	Did not converge

[4] **Multi-agent data**: Each episode generates experiences for 3 agents, effectively tripling data efficiency

2) Computational Efficiency: The browser-based environment proved computationally efficient:

– **Episodes per iteration**: 240 episodes (48 parallel × 5 rollouts)
– **Iteration time**: 1.8 minutes average
– **Total training time**: 250 hours for 600 iterations
– **Timesteps per second**: Approximately 3,500 (across all parallel workers)

The WebSocket-based architecture introduced minimal overhead compared to native Python simulations, demonstrating the viability of browser-based training environments.

F. Ablation Studies

To understand the contribution of key design choices, we conducted ablation experiments:

1) Reward Function Components: Key findings:

– **Vision rewards**: While not essential (performance actually increased slightly without them), they significantly accelerated convergence by providing early learning signals
– **Distance rewards**: Critical for both roles, providing continuous gradients that guide exploration toward useful behaviors
– **Terminal rewards**: Necessary but insufficient—agents need continuous feedback to develop effective strategies

2) Observation Space Components: We tested the impact of different observation components by masking them during training:

– **Without boundary proximity** (-4 dims): Hiders learned to get stuck at edges, reducing win rate to 52%
– **Without visual field** (-128 dims): Agents exhibited near-random behavior, unable to learn pursuit or evasion
– **Without target memory** (-4 dims): Minimal impact (66% win rate), suggesting memory is less critical than real-time perception in this fast-paced game

The visual field (128 dimensions) proved most critical, confirming that effective perception is essential for hide-and-seek success.

G. Training Stability

Figure 2 demonstrates several indicators of stable training:

[1] **Smooth reward progression**: Episode rewards increased steadily without catastrophic drops
[2] **Controlled policy updates**: KL divergence remained below target threshold, indicating PPO's clipping mechanism successfully prevented destabilizing updates
[3] **Maintained exploration**: Entropy decreased gradually but remained positive, ensuring continued exploration while exploiting learned behaviors
[4] **Value function convergence**: Value loss decreased consistently, indicating accurate state value estimation

The stability is particularly notable given the multi-agent nature of the task, where non-stationarity from simultaneously updating policies could cause instability.

H. Training Limitations and Challenges

Despite achieving stable convergence, several factors limited the ultimate performance:

1) Underutilized Action Space: With flat terrain and disabled block modification, 3 of 7 action dimensions (jump, place_block, remove_block) remained largely unused. Analysis of action distributions showed jump actions occurring in fewer than 5% of timesteps, suggesting insufficient environmental incentive for vertical movement. This represents approximately 43% of the action space (3 of 7 dimensions) providing minimal learning signal.

2) Task Asymmetry: The 68% hider win rate indicates an imbalanced task difficulty. Several factors contribute to this asymmetry:

– **Agent ratio**: 2 hiders vs 1 seeker forces the seeker to divide attention
– **Preparation advantage**: 3-second head start allows hiders to reach strong positions before seeking begins
– **Search space**: 64×64 block area (4,096 possible positions) is large relative to the 17-second seeking phase
– **Detection requirement**: Seeker must get within 2 blocks to tag, while hiders only need to maintain distance

The seeker essentially faces a 17-second time limit to search 4,096 locations and approach within 2 blocks of 2 moving targets—a challenging optimization problem even for optimal policies.

3) Fixed Terrain Overfitting: Training exclusively on a single terrain layout (controlled by fixed random seed) ensured reproducibility and isolated policy learning from environmental variation. However, this may have encouraged terrain-specific tactics rather than general hide-and-seek principles. Agents could exploit specific features (particular hills, tree locations) that would not transfer to novel maps.

4) Exploration-Exploitation Trade-off: The final entropy values (seeker: 10.37, hider: 22.34) suggest continued exploration even at convergence. While this maintains strategic diversity and prevents premature convergence to suboptimal deterministic policies, it may indicate incomplete exploitation of discovered strategies. The seeker's 32% win rate could potentially improve with reduced exploration (lower entropy) allowing more consistent execution of proven tactics.

These limitations motivate our planned curriculum learning approach (Section 7), where progressive terrain complexity and eventual block modification will encourage fuller utilization of the action space while addressing task balance through adjusted game parameters.

7. Conclusion and Future Work

This paper has presented a Proximal Policy Optimization framework for training Non-Player Characters in a browser-based 3D voxel hide-and-seek game. By combining Ray RLlib's robust PPO implementation with a JavaScript-based game environment through WebSocket communication, we successfully trained agents to exhibit sophisticated hide-and-seek behaviors through self-play.

A. Key Contributions

Our work makes the following contributions to game AI and reinforcement learning:

[1] A validated PPO-based training system for multi-agent hide-and-seek in 3D voxel environments, achieving stable convergence in 96,000-108,000 training episodes.

[2] A WebSocket-based architecture enabling Python RL frameworks to train agents in browser-based game environments, demonstrating computational efficiency comparable to native Python simulations (3,500 timesteps/second).

[3] Comprehensive observation encoding (161 dimensions) and reward design that guide agents toward effective hide-and-seek strategies without manual behavior programming.

[4] Empirical demonstration of emergent strategic behaviors, including terrain utilization, systematic search, and dynamic evasion tactics, validated through 100-game tournament evaluation showing 68% hider win rate.

[5] Open-source implementation providing a foundation for future research in browser-based multi-agent reinforcement learning.

B. Validation of Core Hypotheses

Our implementation successfully validated several key hypotheses:

[1] **Browser Viability**: Browser-based 3D environments can serve as effective training platforms for reinforcement learning, with WebSocket communication introducing minimal overhead.

[2] **PPO Effectiveness**: PPO with self-play successfully trains competitive hide-and-seek policies in continuous action spaces with partial observability, outperforming our earlier DQN attempts.

[3] **Emergent Complexity**: Without explicit programming, agents developed sophisticated strategies including terrain exploitation, systematic coverage, and multi-agent coordination.

[4] **Sample Efficiency**: Self-play with carefully designed rewards achieved competent behavior in approximately 100,000 episodes—reasonable for multi-agent continuous control tasks.

C. Lessons Learned

Several insights emerged during implementation that may benefit future research:

[1] **Algorithm Selection**: Switching from DQN to PPO was critical. PPO's stability, native continuous action support, and entropy management proved essential for this domain.

[2] **Reward Shaping**: Balancing terminal rewards (+50/-50) with continuous rewards (vision, distance) provided both clear objectives and useful exploration gradients. Pure sparse rewards failed to produce learning.

[3] **Observation Design**: The 64-ray vision system (128 dimensions: distance + type) provided sufficient perception while remaining computationally tractable. Higher resolution (256+ rays) showed diminishing returns.

[4] **Fixed Environment**: Using a fixed random seed for terrain generation eliminated a source of variance, accelerating convergence. Future work should explore curriculum learning with increasing terrain complexity.

[5] **Multi-Agent Dynamics**: Separate policies with role-specific learning rates and entropy coefficients worked better than shared policies, allowing each role to develop specialized strategies.

[6] **Simulated Time**: Using simulated time instead of wall-clock time was essential for headless training, allowing maximum speed physics simulation.

D. Limitations

Despite successful validation, several limitations should be acknowledged:

[1] **Fixed Terrain**: Training on a single terrain layout may limit generalization. Agents might overfit to specific features of the training map.

[2] **Hider Advantage**: The 68% hider win rate suggests the task may be asymmetrically difficult. Future work could explore: (a) multiple seekers, (b) reduced preparation time, (c) smaller play areas, or (d) vision-impairing terrain modifications.

[3] **Evaluation Scope**: Tournament evaluation used the trained policies in deterministic mode on the training map. Performance on novel maps or against human players remains unexplored.

[4] **Spectator-Only**: The current system supports observation only—human players cannot interact with trained agents. This limits evaluation to agent-vs-agent competition.

[5] **Computational Requirements**: While efficient for its capabilities, the system still requires GPU resources for reasonable training times (250 hours for 600 iterations).

E. Future Work

Building on this foundation, we identify several promising research directions:

1) Curriculum Learning: Investigating progressive training approaches:

Our immediate next step involves implementing curriculum learning through gradual terrain complexity increases to address the underutilized action space identified in Section 6:

Phase 1 (Completed - Iterations 0-600): Flat terrain with minimal height variation (0-1 blocks). This phase established basic pursuit-evasion behaviors and achieved 68% hider win rate, demonstrating successful fundamental strategy learning.

Phase 2 (Planned - Iterations 600-1200): Introduce gentle slopes (1-3 block height variation) to necessitate jumping behavior. Currently, the flat terrain renders the jump action nearly useless, leaving 14% of the action space (1 of 7 dimensions) underutilized. By requiring navigation over small obstacles, agents will learn when and how to jump effectively.

Phase 3 (Planned - Iterations 1200-1800): Increase to moderate terrain (3-5 block variations) with hills and valleys, requiring sophisticated 3D navigation. Hiders should learn to exploit elevation for concealment and line-of-sight breaking, while seekers must develop strategies for high-ground scanning and vertical pursuit.

Phase 4 (Planned - Iterations 1800+): Enable block modification (initially 5 blocks per agent) after agents have mastered basic hide-and-seek on varied terrain. This phased approach addresses the credit assignment and state space explosion issues that necessitated disabling block modification initially.

The curriculum progression allows agents to build upon previously learned skills rather than facing full complexity immediately, following principles demonstrated effective in similar multi-agent domains [7].

a) Why Block Modification Was Disabled: Block placement and removal actions remain disabled during initial training (Phases 1-3) for several critical reasons:

[1] **State space explosion**: With 10 blocks per agent in a 64×64 world, the number of possible block configurations becomes computationally intractable. The observation space would need to encode all modified blocks' positions, potentially requiring convolutional architectures or spatial attention mechanisms to process effectively.

[2] **Long-term credit assignment**: Blocks placed early in an episode (e.g., t=20) typically only demonstrate value much later (e.g., t=200), creating 180-step temporal credit assignment problems. Standard PPO with 240-step episodes struggles with such long delays between action and reward.

[3] **Non-stationary environment**: Player-modified blocks violate the Markov property assumption underlying PPO. Future states depend on past actions through environmental changes that persist across timesteps, complicating value function approximation.

[4] **Computational overhead**: Block collision detection with player-placed objects and modified ray-casting operations increase simulation time by approximately 40%, reducing training throughput from 3,500 to 2,100 timesteps/second.

By first establishing competent pursuit-evasion behaviors on static terrain, we create a foundation upon which environmental manipulation strategies can build. This mirrors successful curriculum approaches in other complex domains where task difficulty increases progressively [8].

When enabled in Phase 4, block modification will add strategic depth:

– Hiders can construct walls or barriers for concealment
– Seekers can remove obstacles blocking paths or sightlines
– Both roles must balance time spent modifying environment vs. core objectives

This staged introduction ensures agents master fundamental skills before tackling the significantly more complex problem of dynamic environment manipulation.

2) Task Balancing: Addressing the asymmetric difficulty that resulted in 68% hider win rate:

– Adjust agent ratios: Test 2 seekers vs 2 hiders for symmetric competition
– Reduce preparation time: Decrease from 3 seconds to 1 second to limit hider head start
– Modify world size: Experiment with smaller areas (48×48 or 32×32) to reduce search space
– Extend seeking phase: Increase from 17 to 25 seconds to allow more thorough search
– Implement tagging assistance: Increase tag range from 2 to 3 blocks to ease final capture

The goal is achieving approximately 50% win rates for both roles, indicating balanced difficulty while maintaining strategic depth.

3) Human-AI Interaction: Developing capabilities for human engagement:

– Implement controllable agents for human-vs-AI play
– Study how trained agents respond to human strategies
– Investigate adaptive difficulty through policy mixing
– Explore imitation learning from human demonstrations

4) Alternative RL Approaches: Comparing with other algorithms:

– Recurrent policies (LSTM/GRU) for explicit memory
– Multi-Agent PPO (MAPPO) with centralized critic
– Soft Actor-Critic (SAC) for sample-efficient exploration
– Curiosity-driven methods for improved exploration

5) Generalization Studies: Testing robustness and transfer:

– Train on diverse terrain distributions
– Evaluate zero-shot performance on novel maps
– Study domain randomization effects
– Investigate meta-learning for rapid adaptation

6) Perception Enhancements: Our 64-ray vision system (8×8 grid) provides effective environmental awareness, but future research could explore:

– Higher resolution ray-casting (16×16 = 256 rays) for finer spatial perception
– Attention mechanisms to focus processing on relevant rays (central vs. peripheral)
– Convolutional layers over ray grid structure to extract spatial patterns
– Depth-based features encoding distance relationships between detected objects
– Temporal attention over recent ray-cast history for motion detection

These enhancements would increase observation dimensionality but could improve agents' ability to detect subtle movements or track fast-moving targets. The computational trade-off between perception quality and simulation speed requires careful evaluation.

F. Broader Applications

Beyond hide-and-seek, our framework enables research in:

– **Multi-Agent Coordination**: Training cooperative teams in spatial reasoning tasks
– **Procedural Content Generation**: Using RL to design balanced game environments
– **Dynamic Difficulty Adjustment**: Real-time policy mixing for player skill matching
– **Behavioral Animation**: Generating realistic NPC behaviors for non-competitive scenarios

The browser-based architecture makes the system accessible for educational purposes, allowing students to experiment with RL concepts in a visual, interactive environment without complex installation requirements.

G. Closing Remarks

This research demonstrates that browser-based 3D environments can serve as effective platforms for multi-agent reinforcement learning research. The combination of THREE.js for visualization, WebSockets for communication, and Ray RLlib for training provides a practical framework that balances accessibility with performance.

Our PPO-based agents learned sophisticated hide-and-seek behaviors through pure self-play, without manual behavior programming or human demonstration. The 68% hider win rate and emergent strategies validate that the system can discover non-trivial solutions to spatial reasoning and pursuit-evasion problems.

Key factors contributing to success included: (1) PPO's stability and continuous action support, (2) carefully designed rewards balancing terminal and continuous signals, (3) comprehensive observation encoding capturing essential spatial information, and (4) efficient WebSocket communication enabling Python-browser integration.

The open-source release of our implementation provides researchers and developers with a foundation for exploring reinforcement learning in browser-based 3D games. As web technologies continue advancing—particularly with WebGPU for GPU compute acceleration—browser-based RL environments will become increasingly viable for sophisticated research applications.

Future work should focus on generalization (varied terrains), extended mechanics (block manipulation), human-AI interaction, and alternative RL algorithms. The framework is designed to support these extensions with minimal architectural changes, enabling continued research in this promising direction.

We hope this work inspires further exploration of browser-based reinforcement learning and demonstrates the potential for combining modern web technologies with advanced machine learning techniques to create accessible, interactive research platforms.

ACKNOWLEDGMENT

The author would like to thank the Department of Computer Science at the University of Miami for their support and resources that made this research possible. Special thanks to the participants in our user study for their valuable feedback and engagement with the system.

REFERENCES

[1] E. Meeds, R. Hendriks, F. Al Farabi, M. Gower, N. Wiggins, and A. Weller, "Mlitb: Machine learning in the browser," *PeerJ Computer Science*, vol. 1, p. e11, 2015.

[2] M. Kopel, D. Hajas, and J. Hromkoviče, "Ai for npcs in 3d games: Q-learning and computational efficiency," *IEEE Transactions on Games*, vol. 10, no. 2, pp. 129–141, 2018.

[3] D. Smilkov, N. Thorat, Y. Assogba, A. Yuan, N. Kreeger, P. Yu, K. Zhang, S. Cai, E. Wei, F. Viegas, and M. Wattenberg, "Tensorflow.js: Machine learning for the web and beyond," in *Proceedings of the 2nd SysML Conference*, 2019, pp. 309–321.

[4] S. Li, X. Wang, Y. Zhang, W. Xue, J. Černý, and B. An, "Solving large-scale pursuit-evasion games using pre-trained strategies," in *Proceedings of the AAAI Conference on Artificial Intelligence*, vol. 37, no. 10, 2023, pp. 11 586–11 594.

[5] X. Huang, "Starcraft adversary-agent challenge for pursuit-evasion game," *International Journal of Intelligent Systems*, vol. 36, no. 12, pp. 7226–7247, 2021.

[6] D. Amodei, J. Clark, G. Krueger, and I. Sutskever, "Ai agents and applications in gaming environments," *Journal of Artificial Intelligence Research*, vol. 78, no. 3, pp. 412–438, 2024.

[7] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch, "Emergent tool use from multi-agent autocurricula," *arXiv preprint arXiv:1909.07528*, 2019. [Online]. Available: https://arxiv.org/abs/1909.07528

[8] M. Plappert, R. Sampedro, T. Xu, I. Akkaya, V. Kosaraju, P. Welinder, R. D'Sa, A. Petron, H. P. d. O. Pinto, A. Paino, H. Noh, L. Weng, Q. Yuan, C. Chu, and W. Zaremba, "Asymmetric self-play for automatic goal discovery in robotic manipulation," *arXiv preprint arXiv:2101.04882*, 2021. [Online]. Available: https://arxiv.org/abs/2101.04882