Tufts University
Department of Computer Science
COMP 138 - Reinforcement Learning

Deep Q-Learning for Tree Chopping in Craftium with Reward Shaping and Visual Augmentation

By: Ash Sze, Cheng Xi Tsou and Haijun Si
Supervisor: Jivko Sinapov
May 9, 2025

1 Introduction

In real-world environments, agents rarely have access to fully observable or discrete state spaces – they must learn based on sensory inputs such as vision, noise, or feel. Classical RL benchmarks such as gridworlds, cartpole, and Atari games assume full observability and access to low-dimensional state representations which limits their abilities to mimic real-world tasks such as autonomous navigation, tasks that involve precise control of complex robotic components, or human-environment interactions. Previous works such as Deep Q-Learning (Mnih et al., 2015) have demonstrated human-level proficiency in Atari games using 64x64 pixel images where the agent has a full observation of the environment with a simple rule set. However even with the simple environment, solving reinforcement learning problems with pure visual inputs and a sparse reward function is a hard task that requires long training times, expensive computations, and is slow to converge. In order for robust agents to be deployed and effective in real world tasks, they must be trained in a physics-driven environment with enough entity complexity to support a diverse set of tasks.

In real-life, human perception is often noisy and our brains choose to dynamically focus on important visual signals to prioritize important information. In this paper, we explore how visual augmentations mimicking a human's perception such as a limited FOV (field-of-view), blurred edges, or only perceiving edges and outlines can solve the task of tree-chopping in Minecraft, a highly complex 3D environment where the agent only has a partially observable pixel input of the environment. Our results show that our agent with visual augmentations that significantly reduce the state space along with domain specific reward shaping is able to perform just as well as unmodified visual inputs. This work highlights the importance of input augmentation in vision based reinforcement learning and how visual augmentations that significantly reduce the state space still retain enough information for an agent to complete a task.

2 Background

Minecraft is an open-world sandbox game with a procedurally generated voxel-based 3D environment. Discrete blocks in the game represent different materials, such as wood, stone, and dirt, which can be mined, collected, and used by players for crafting, and building. Players typically navigate in first-person perspective. Although there is no explicit endgame, players can pursue achievements such as acquiring wood, crafting tools, and defeating endgame bosses. Minecraft's physics system includes entity movement, gravity, and fluid mechanics, making it a moderately complex domain for reinforcement learning (RL) research.

RL in visually rich environments typically involves high-dimensional state spaces. Deep Q-Networks (DQNs) (Mnih et al., 2015) have demonstrated success in such scenarios by approximating the Q-value function using a convolutional neural network (CNN). This architecture processes raw pixel observations to estimate action values, enabling agents to learn directly from visual inputs without needing manual feature engineering. The standard DQN pipeline includes experience replay, which stores past transitions in a memory buffer, and target network freezing, a Q-network which is updated less frequently (Deepmind, 2017). Both of these strategies help stabilize training in high-variance, sparse-reward domains like Minecraft.

In this project, we use the Craftium framework, an environment built on top of Minetest, to simulate chopping a tree in Minecraft (Craftium, 2024). The agent receives first person visual observations of the environment and selects from the discrete actions of moving forward, mouse movement, and digging. Our baseline uses a standard DQN to learn the tree-chopping task. To encourage efficient learning and reduce policy collapse into irrelevant behaviors (e.g., digging at dirt or air), we apply shaped rewards including sparse task completion bonuses and denser intermediate feedback (e.g., sustained attention on wood blocks, proximity, and penalizing partial digs).

Recent literature suggests that learning in visually augmented spaces may not only accelerate convergence but also improve generalization (Kostrikov et al., 2021). Inspired by these findings and also by human vision, we explore how varying levels of visual augmentation can affect DQN learning. Specifically, we consider three common forms of visual alteration: Gaussian blurring, center cropping, and Canny edge detection to extract high-contrast boundaries while removing color and intensity gradients.

Our modifications are inspired by biological systems, where peripheral vision and color precision dynamically change to prioritize essential information. Studying these augmentations allows us to test our hypothesis, and see if constrained visual inputs can actually guide the learning process away from distractors. In a broader context, these experiments help us better understand how RL agents might learn more robust and transferable policies in environments that resemble the imperfections of real-world sensing.

Our work aims to investigate the effects of reward shaping, and whether lower-fidelity visual representations can improve model learning in complex visual environments like Craftium. Our findings have the potential to influence the design of future RL agents that operate under real-world visual constraints, where image corruption, occlusion, and partial observability are the norm rather than the exception.

3 Environment Set-up

3.1 Minecraft

Minecraft is a complex open-world and partially observable 3D sandbox environment that attempts to simulate the real world and has emerged as a powerful testbed for reinforcement learning research where agents require long horizon planning, spatial reasoning, and exploration to solve tasks. The world is made up of discrete blocks that represent different materials that a player can collect and a variety of static and autonomous entities such as animals, trees, caves, and enemies that a player can interact with. Minecraft also features a consistent set of physics rules that constrains how a player can interact with the world. The player cannot move through blocks, must navigate through different terrains, and are subject to movement dynamics such as velocity, falling, momentum, and jumping. This makes spatial reasoning an important aspect in any task in Minecraft.

Unlike classical control tasks such as CartPole or MountainCar, the Minecraft environment is more similar to the real world where agents can only operate on raw visual inputs with delayed and sparse rewards. Due to its highly stochastic environment and open sandbox world, there are many kinds of tasks that can simulate problems in the real-world. In this paper, we will be conducting experiments using Craftium, a Gymnasium-compatible wrapper for the game Minecraft that provides RL interfaces and an easily customizable framework for training agents (Craftium, 2024).

3.2 Task: Chopping a Tree

The task our agent is aiming to solve is chopping a tree, made up of a one block wide vertical column of wood blocks topped with a cube of leaf blocks. The agent spawns in a dense forest filled with many trees during the day time with an empty inventory. The agent can chop a tree block by repeatedly digging at a tree block for a fixed duration until the tree block is removed from the environment. Every time the agent chops the tree block a positive reward of 1.0 is given, otherwise the reward is 0.0 so the goal is to chop as many tree blocks as you can. This setup requires the agent to have vertical camera control, to identify and navigate to a tree, and make sequential actions to successfully chop a tree block.

The observation space is a 64x64 pixel image with 3 color channels with values [0, 255]. As the agent's perception is given in a first-person point of view, an observation at any time is a partially observable snapshot of the state of the world. The action space is limited to a set of 8 discrete actions: no action, forward, jump, dig, and mouse x+, mouse x-, mouse y+, mouse y- that simulate an incremental change in the camera's horizontal and vertical angles. The dig action interacts with the block at the center of the agent's screen indicated by a cross which requires precise control and aim to execute on the desired block. To fully chop a block, the agent must execute the dig action consecutively for a fixed duration so any other action will reset the timer.

Figure 1: First-person view of the Craftium observation space which mimics the first-person Minecraft gaming experience.

4 Reinforcement Learning methods

4.1 Deep Q-Learning

In reinforcement learning, one of the most popular methods for learning is an off-policy temporal difference control algorithm called Q-learning, defined by:

$$Q(S_t, A_t) \leftarrow Q(S_t, A_t) + \alpha[R_{t+1} + \gamma max_a Q(S_{t+1}, a) - Q(S_t, A_t)]$$

Similar to TD learning, Q-learning allows us to approximate the optimal action-value function q_* directly from sampled experience where the agent is only exposed to a sample of the environment's dynamics of states, actions, and rewards. This is a very important characteristic as it allows the agent to learn in an environment through interaction alone without the full knowledge of an environment's model. Q-learning also approximates the optimal action-value function q_* independent of the policy being followed which allows us to learn action-values that follow subsequent optimal behavior while sampling from a trajectory following suboptimal behavior such as ε-greedy which encourages exploration.

The environment we are using is based on the video game Minecraft, which has a very complex and continuous state space. Tabular Q-learning can be very computationally expensive, slow to converge, and require a large amount of experience in complex and sparse state spaces so nonlinear function approximation methods are neccesary to capture the complex patterns in the optimal policy. In this project, we will be using Deep Q-learning (Mnih et al 2013) which uses a set of weights θ to estimate the optimal action-value function via a neural network and updates with stochastic gradient descent. The Q-network minimizes the following loss:

$$L_i(\theta_i) = \mathbb{E}_{s,a}[(y_i - Q(s, a; \theta_i))^2]$$
$$y_i = \mathbb{E}'_s[r + \gamma max_a Q(s', a'; \theta_{i-1})|s, a]$$

where instead of expectations, we update the Q-network at every time-step with a single sample sampled from the behavior distribution following an ε-greedy policy that selects a random action a with probability ε. The algorithm also keeps a memory buffer of past experiences and retrains the model with a random sample at every timestep to smooth out learning and avoid divergence in the parameter. In our project, our agent will be using Deep Q-learning to approximate an optimal policy. As our environment's observation space is a 64 by 64 image with 3 color channels and values [0,255], we will be using a convolutional layer in our neural network.

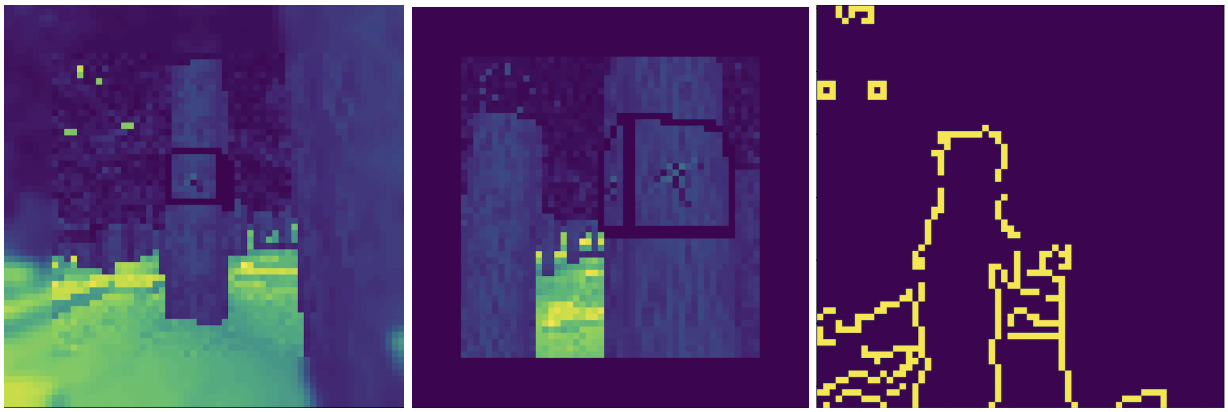
4.2 Visual alterations

We applied three types of visual alteration to our baseline model: blurred input, reduced field-of-view (FOV) input, and edge detection. In the blurred input condition, we applied a Gaussian blur to the outer 8-pixel border of each 64×64 image frame before feeding it into our Deep Q-Network (DQN). Gaussian blur is a smoothing technique that reduces image detail by

averaging pixel values with their neighbors, weighted by a Gaussian function. This results in a soft, gradual blur that mimics how visual information might alter in peripheral vision.

For the reduced FOV input, we limited the agent's visible area by effectively blackening the outer 8-pixel ring of the 64×64 image. Since our DQN expects a 64×64 input, we preserved the image dimensions but removed visual information from the edges, simulating a narrower field of view.

Lastly, edge detection was applied by using the Canny edge detection algorithm where an image's complexity is reduced by only keeping the outlines and edges of objects. We preprocess the image by converting it into grayscale and apply a Gaussian blur. Then, we calculate the gradient magnitude and orientation of the pixels to determine which edges to keep in the image. We hypothesize that for this specific task, the outlines of the trees along with the digging action indicators are enough for the agent to learn how to chop trees. This visual augmentation should speed up convergence as there is less noise to learn at the cost of performance as we have less information.



Figures 2-4: Blurred observation space (left), reduced FOV of the observation space (middle), and edge map detection of the observation space (right).

5 Experimental Set-up

5.1 Technology Used

We used 3.12 Python as the main programming language for setting up our environment and training our agent. The core libraries and packages we used were Pytorch, numpy, Matplotlib and Gymnasium. For our reinforcement learning framework, we used Craftium and Minetest for Lua modding.

5.2 Training Protocol

We detail the training protocol for a single DQN agent to learn the task of chopping a wood block using the Craftium environment. Each episode terminates either after a fixed number of steps (1000). We decided against ending the episode earlier if the agent completes the task

(chop a single wood block) in order to allow more opportunities for the agent to be rewarded for chopping wood. Our training protocol follows the DQN training algorithm from the class textbook, but incorporates domain-specific reward shaping in init.lua.

Algorithm 1 Deep Q-Learning Algorithm (Mnih et. al 2013)

Initialize replay memory $\mathcal{D}$ with capacity N
Initialize action-value function $Q(s, a; \theta)$ with random weights θ
for each episode **do**
  Initialize state s_0
  for each step in the episode **do**
    Select action a_t using ϵ-greedy policy:

$$a_t = \begin{cases} \text{random action} & \text{with probability } \epsilon \\ \arg\max_a Q(s_t, a; \theta) & \text{otherwise} \end{cases}$$

    Execute action a_t and observe reward r_t and next state s_{t+1}
    Store transition (s_t, a_t, r_t, s_{t+1}) in $\mathcal{D}$
    Sample random minibatch of transitions (s_j, a_j, r_j, s_{j+1}) from $\mathcal{D}$
    Compute target y_j:

$$y_j = \begin{cases} r_j & \text{if terminal state} \\ r_j + \gamma \max_{a'} Q'(s_{j+1}, a'; \theta^-) & \text{otherwise} \end{cases}$$

    Perform gradient descent step on loss:

$$L(\theta) = \mathbb{E}[(y_j - Q(s_j, a_j; \theta))^2]$$

  end for
end for

Figure 5: Pseudo-code from the Barto and Sutton textbook for the deep Q-Learning algorithm from the Mnih et. al 2013 paper on playing Atari with deep reinforcement learning.

5.1.1 Training Episodes

Baseline models with no visual augmentation were trained for 300 episodes, while models with visual augmentations were trained for 200 episodes. Training was stopped earlier in select models due to early convergence. Training episodes were capped at 1000 steps.

5.1.2 Evaluation Metrics

Training performance was measured with the following metrics:

- Reward per episode: Total reward accumulated across each episode.
- Training Loss: The Huber loss computed during Q-network updates.
- Time to Convergence: Measured in episodes.

Loss and reward are visualized live during training using Matplotlib, with loss tracked over gradient updates and reward over episodes.

5.1.3 Model Trained

We train the following four models (baseline, blurred, cropped, edgemap) using identical DQN architecture.

Baseline

Unmodified grayscale visual input with standard reward shaping. The baseline model was subject to the most development in order to ensure the tree chopping task was learnable before

we moved on to training with visual augmentations. We trained the baseline with both simple reward shaping and complex reward shaping. We decided to use the architecture of simple reward shaping for training the visually augmented models because it was more robust and consistent in tree chopping after training.

Blurred Gaussian blur applied to each frame, using the training architecture with simple reward shaping.

Cropped Center crop applied to restrict field of view, using the training architecture with simple reward shaping.

Edgemap Input converted to binary edge maps (Canny operators), using the training architecture with simple reward shaping.

5.2 Implementation Detail

5.2.1 Reinforcement Learning Algorithm

We use DQN as described by Mnih et al. (2015) and the textbook algorithm for DQN as a coding reference, with several modern enhancements. The target Q-network is updated via Polyak averaging (0.005), a buffer of 10,000 transitions stores (state, action, reward, next_state) tuples and batches of 128 are sampled for training.

We use epsilon-greedy exploration with an initial epsilon of 0.9, final epsilon of 0.05 and an epsilon decay rate of 3000 steps (exponential).

5.2.2 Optimization and Hyperparameters

Hyperparameter	Value
Optimizer	adamW
Learning rate	1e-4
Batch size	128
Discount factor	0.99
Frame skip	4
Frame stack	4
Mouse movement	+/- 0.2
Render mode	“human”
Loss function	Huber (smoothL1loss)

Table 1: Table detailing the hyperparameter specifications used in training the simple and complex baseline model, and the three simple augmentation models.

The environment is configured with a 64x64 pixel resolution in grayscale, with maximum 1000 timesteps for each training episode. Our discrete action wrapper contains forward, dig, mouse x+, mouse x-, mouse y+, mouse y-. We elected to exclude “jump” in order to simplify the action space and because 1000 timesteps makes jumping unnecessary as the agent would not need to navigate large distances to find a tree.

5.2.3 Code Flow

Our overall code flow is detailed as follows:

1. The agent receives an observation and selects an action using epsilon-greedy policy.
2. The environment transitions to a new state and returns a reward.
3. The transition is stored in the replay buffer.
4. A minibatch is sampled to update the Q-network.
5. The target network is softly updated.
6. The agent’s epsilon is decayed.

5.3 Reward Shaping

In the simple baseline, and visual augmentation models we used three reward functions.

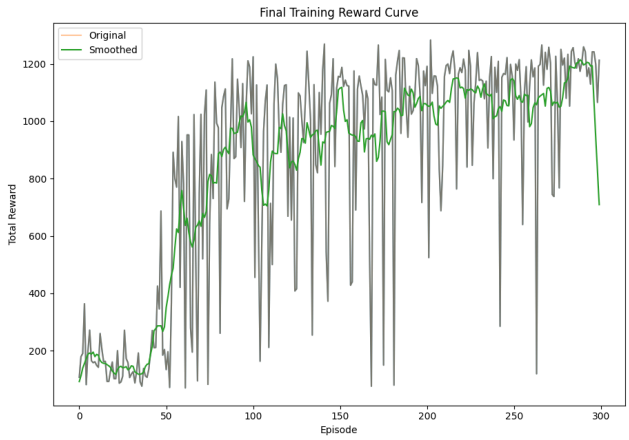
1. +15.0 reward for successfully digging a tree block through minetest.register_on_dignode with "tree" detection.
2. −5.0 penalty for digging any non-tree block through minetest.register_on_dignode catching not string.find(node.name, "tree").
3. +0.5 reward every time the player is holding the dig button (LMB) through general sustained digging reward with no raycast or node checks.

In the complex baseline, we explored the following functions as well.

4. +3.0 reward for punching a tree block that is currently in focus which requires current_look_pos to match the punched block.
5. +1.0 reward for punching a tree block that is being looked at via raycast
6. −0.2 penalty for punching any block that is not a tree
7. +0.5 reward for sustained digging at the same tree block for >1.5s
8. −2.0 penalty if the player aborts digging a tree block too quickly (<1s)
9. +0.5 reward for maintaining gaze (look direction) on a tree block for >2.0s
10. −4.0 penalty for partially completed dig with timeout
11. Proximity-based reward for being near the closest tree calculated from distance, with reward increasing as distance decreases.

6 Results

Figures 6 and 7 show the agent exploring in episodes 0-50, and gradually learning to maximize rewards in episodes 50-300. Both reward shaping models begin to converge towards episode 50. The convergence in episode 50 suggests overfitting due to too many episodes.



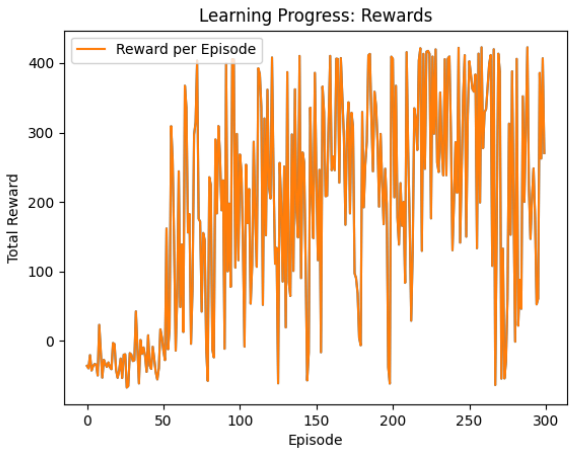


Figure 6 and 7: Final training reward curve for total reward per episode with the complex reward shaping baseline model (left) and the simple reward shaping baseline model (right).

The loss over time looks to be decreasing in figure 8, with spikes around 100,000 and 300,000 training steps. The maximum loss is around 0.12, and minimum loss is 0.04.

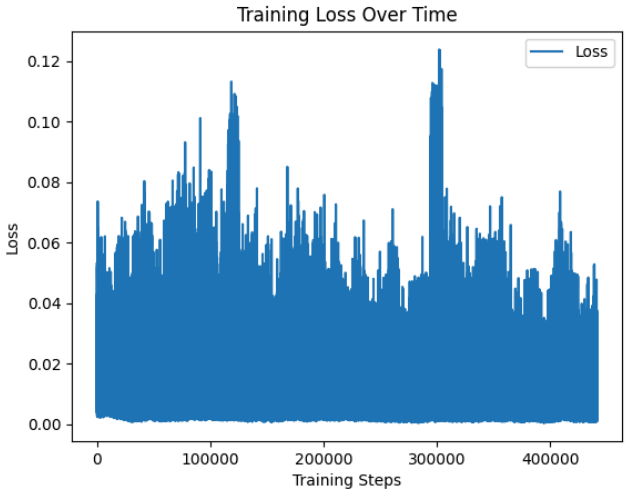


Figure 8: training loss trends towards decreases very slightly but does not show clear convergence compared to total reward.

Figures 9-11 show the results of the effect of visual alterations on learning. Blurred edges and reduced FOV seem to converge relatively quickly around episode 0-10, while edge map takes considerably longer around 175-200 episodes.

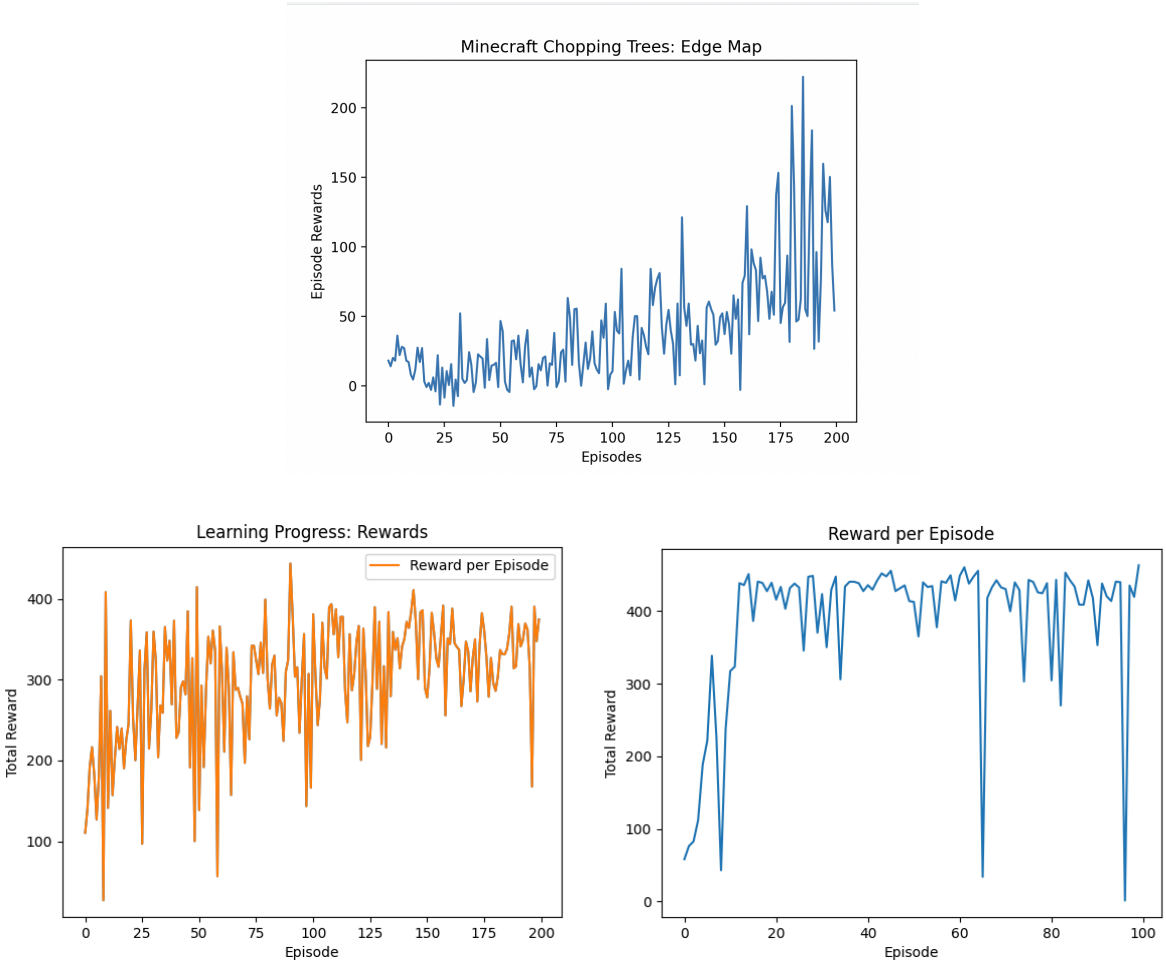


Figure 9-11: Final training reward curve for total reward per episode with the simple reward shaping model for reduced FOV (bottom-left), blurred (bottom-right) and edgemap (top).

7 Discussion

In this project, we investigated how reward shaping and visual alteration affected learning performance of reinforcement learning (RL) agents in a pixel-based environment. Specifically, we used a tree-chopping task in Craftium to compare baseline Deep Q-Learning (DQN) agents against agents trained with blurred vision and restricted field-of-view (FOV) inputs. Alongside this, we explored various methods in reward shaping to help the agent learn its task better in such a complex environment.

From our results, we found that reward shaping played an important role in stabilizing learning and improving sample efficiency across all visual conditions. Models without reward shaping struggled to make much progress, highlighting the necessity of reward functions in sparse-reward environments.

Additionally, we found that both blurred and reduced FOV agents performed comparably to the baseline in terms of total episode rewards and training stability. Both agents showed improved

convergence rates early in training, potentially helping the agent focus on central, task-relevant information. This is consistent with prior work suggesting that reducing irrelevant visual detail can regularize pixel-based RL models and improve stability (Kostrikov, 2021). However, we found that edge mapping hindered the agent's ability to learn more efficiently.

Limitations of our study include a relatively small training budget (due to hardware constraints each training session took ~9 hours) and a limited range of task complexity. Additionally, our visual alterations were applied uniformly to every frame, while human vision dynamically adapts focus and attention. This static approach may have oversimplified perceptual alteration in ways that don't fully capture biologically-inspired mechanisms. Furthermore, our results are limited to a single task (tree-chopping), and findings may not generalize to more dynamic or multi-objective environments.

8 Conclusion and Future Directions

The project was successful. We gained valuable experience in conducting research, integrating with open-source projects and external APIs, and implementing our own Deep Q-Network (DQN) algorithm with customized reward shaping. We learned that reward shaping is very delicate in complex environments like this and needs to be balanced properly with the RL algorithm. These skills not only deepened our understanding of deep reinforcement learning but also taught us practical lessons about working within larger codebases and adapting existing methods to new problems.

We explored how visual alteration and reward shaping affects reinforcement learning agents' ability to learn a basic manipulation task in a pixel-based environment. Our findings show that moderate visual simplification like blurring and reduced FOV can stabilize learning, while severe constraints like edge mapping impairs the agent's ability to learn efficiently.

This suggests that certain types of biologically-inspired perceptual constraints may benefit reinforcement learning in pixel-based tasks by filtering irrelevant visual noise. It also supports previous research that regularizing visual input can improve stability and sample efficiency in deep RL models.

Future work includes extending this framework to more complex tasks (e.g., killing spiders), developing more generalizable reward shaping strategies for multi-task learning, and exploring additional forms of visual alteration. This could involve adjusting blur levels based on the agent's attention or combining multiple visual alteration techniques on the agent's perception.

References

1. *Craftium*. (2024). https://craftium.readthedocs.io/en/latest/
2. Deepmind. (2017, Oct 06). Rainbow: Combining Improvements in Deep Reinforcement Learning. https://arxiv.org/pdf/1710.02298
3. Hasselt, H. v. (2015, Sep 22). Deep Reinforcement Learning with Double Q-learning. https://arxiv.org/abs/1509.06461
4. Kostrikov, I. (2021, March 7). Image Augmentation Is All You Need: Regularizing Deep Reinforcement Learning from Pixels. https://arxiv.org/pdf/2004.13649
5. Malagón, M. (2024, July 2). Craftium: An Extensible Framework for Creating Reinforcement Learning Environments. https://arxiv.org/pdf/2407.03969
6. Mnih, V. (2013, Dec 19). https://arxiv.org/pdf/1312.5602
7. Mnih, V. (2015, February 25). Human-level control through deep reinforcement learning. https://www.nature.com/articles/nature14236
8. Shukla, Y. (2022). https://sim2real.github.io/assets/papers/2022/shukla.pdf