AI Agent For Catan
Research Report

School of Computer Science & Applied Mathematics

University of the Witwatersrand

Prepared by Liam Brady
Supervised by Prof. Clint van Alten

November 10, 2025

Abstract

Catan is a complex multiplayer board game with a large, partially observable state-action space, requiring strategic decisions in the face of stochastic game-play elements. Player actions are determined by the placement of pieces on the board and cards obtained through dice rolling. This creates two main elements in the game: the spatial features of the board, and the non-spatial features of the various cards and points. This research report describes an implementation of the Proximal Policy Optimisation algorithm using a combined network architecture - Convolutional Neural Network and Multi-Layer Perceptron (MLP)- to improve feature extraction and performance of an RL Catan playing agent. Reward shaping and network ablation are used to evaluate the effects of dense rewards on agent performance and to determine which feature type contributes more heavily to the agent's winning capabilities. A combination of the different network types were tested using both sparse and dense rewards. Results show a 6.32% improvement from the original sparse reward MLP feature extractor when using the proposed combined/mixed model with dense rewards. The results also show that there is an 8.05% difference in performance in favour of a board only (spatial features) extractor with sparse rewards when compared to a numeric only (non-spatial features) extractor with dense rewards.

1 Introduction

One of the biggest difficulties in Reinforcement Learning is for an agent to make strategic decisions when navigating a large state-action space with opposing agents. Not only must it explore the world in which it has been placed, but also win while attempting to manage stochastic elements, incomplete information, and enemy tactics - all of which change what actions are available at each point in the game. These complexities create many challenges for developing such an agent. The drastic change in each state of the game makes correct feature extraction invaluable, as it provides the agent with as much information as possible for good decision making.
In Catan, there are two such features:

1. Spatial Features - These are the physical aspects of the game, including city, settlement, and road placements of each player, the port locations, and where the robber is at each point in the game.

2. Non-spatial Features - These are the different non-physical aspects of the game, such as the resource cards used for trading, development cards used to give the players bonuses at the expense of resources, and victory points which dictate who the winning player is.

A game such as chess has only spatial features - the pieces on the board influence which action should be taken next, meaning that the entirety of each state can be interpreted by a single network, such as a CNN [1]. If a network such as an MLP [2] were used to extract these features, the information relating to physical patterns of the board would be lost or not fully captured for an agent to make good decisions. Catan has both spatial and non-spatial features which need different network types to effectively extract the information of each state during the game.

This research report introduces an agent which uses a combined feature extractor - CNN and MLP - paired with a Maskable Proximal Policy Optimisation algorithm to improve upon the PPO[9] [3] agent available in the Catanatron environment. This improvement also demonstrates the effectiveness of dense reward shaping on agents exploring large 4-player state-action spaces and determines which feature type, spatial or non-spatial, contributes more towards the winning capability of a Catan playing agent.

[1] Convolutional Neural Network
[2] Multi-Layer Perceptron
[3] Proximal Policy Optimisation

2 Background

There have been multiple approaches to producing a successful Catan-playing agent with a variety of algorithms and architectures. Due to the large state space of Catan, searching algorithms can be very useful in evaluating possible actions.

The Monte-Carlo Tree Search

MCTS is a search algorithm, developed by [6], which analyses the winning probabilities of different actions from a particular state. It does so by simulating every possible game from each available action all the way through until a chosen depth or terminal state, such as winning or losing the game, is reached. A modified MCTS has been incredibly successful in AlphaGo Zero [10], one of the most advanced board game playing agents to date. The first Catan adaptation was used by [11], though it can be noted that the model used perfect information and the algorithm performed better when given more domain knowledge, such as the values of particular outcomes. [4] adapted it further including an evaluation term to work with their particular neural network. Although it showed some success, the model was not compared to the JSettlers bot [7], which other models have used as a standard benchmark. MCTS is a relevant and useful search tool for simulating games, but adds complexity to the model and can be very computationally intensive. For these reasons, the model proposed will not include it.

Neural Networks

Network selection determines how the information of the board will be processed by an RL algorithm, and so finding the right network to represent each state of the board is crucial.

LSTM [14] used an LSTM [4] network, a sub-type of Recurrent Neural Network capable of capturing information from previous states. This provides the agent with information for better decision making when it comes to trading. This is useful in games such as Catan because previous actions can have an effect on current and future actions. The information stored by the network is controlled by a series of gating mechanisms which decide how much information is saved at each time step. This addition to a regular RNN

[4] LSTM - Long Short-Term Memory

[5] resolves the issue of vanishing gradients and works well on sequential and temporal data. Moves in Catan are described as temporal because they are played in order and depend on past actions of players. An LSTM's ability to compute this type of data makes it a good choice for approximating local Q-functions, and the action-dependent Q-values as used in Q-function decomposition, incorporated by [14]. Although an LSTM's ability to capture temporal dependencies is useful, it does not take advantage of the visual nature of the game the same way a CNN might. For this reason, it will not be used in the proposed agent.

CDNN A CDNN [5] includes a CNN to handle 2D input, and a Fully Connected network to handle scalar input. This combination of networks makes use of the spatial aspect of the game board rather than its trading characteristic. Unlike the LSTM, the CNN uses gridded images of the board in the form of "brick coordinates" to make decisions based on patterns and local interactions of paths, hexes and vertices. From these local patterns, it can then identify more complex global patterns found across the board. Features which all behave differently, such as hexes, vertices, and paths are separated into different types of channels to prevent the convolution from processing them as the same. Catan has many such features, as well as others unrelated to space on the board, including development cards. To handle separate 2D and scalar inputs, the two networks are combined into one Cross Dimensional Neural Network, where two networks are interconnected and feed information between each other. This avoids the degenerative nature of two parallel networks with fully connected layers that would process the information separately. This network processes different types of data appropriately, whilst ensuring that they still contribute to updating the network. Although this makes it a good choice for interpreting information in Catan, considering it is able to handle two distinct types of features, it is complex and computationally expensive. For these reasons, the concept of separation of feature types can be taken and used in a simplified architecture.

CNN Similarly to [14], [3] focus on trading in the game, but differ in their application. [3] use a CNN to optimise two tasks at the same time: learning to offer trades and learning to reply to offers. Unlike the LSTM implementation, which focuses on sequential temporal data to track the effects of previous actions on current ones, the CNN implementation learns from the high-dimensional state space consisting of many features describing

[5] RNN - Recurrent Neural Network

the game board and available resources. This differs considerably from how [5] used a CNN. Both employ CNNs but [5] focused on spatial relationships and general board patterns to make general gameplay decisions, whereas the CNN utilised by [3] makes use of trading interactions. [5] separate features into different channels to process the spatial aspects of hexes, vertices, and paths, but the model used by [3] processes semantic trading negotiations, for example: "I will give you brick for lumber", to maximise its trading advantage. This means that their model is designed to choose actions which maximise the results of trading instead of processing the entire action space. A CNN could make for a good feature extractor when analysing the spatial features of the board.

Comparison We can see from the three networks that the LSTM is able to capture time based dependencies more effectively than the CDNN [6] and would better understand how past moves influence the present, whereas the CDNN is more effective at managing the different spatial features of Catan's complex state space, which gives it a stronger ability to interpret how the current positions of players affect the next possible actions. As pointed out by [13], because of Catan's many non-spatial features such as resources, development cards and victory points, a CNN can struggle with noise resulting from these features. The LSTM by [14] emphasises time dependencies and sequential data in trading decisions, the CNN from [3] is designed to maximise long-term rewards through trading policies, and the CDNN from [5] uses spatial data and patterns to make strategic decisions without the use of trade. The best choice of architecture would be, similarly to the CDNN, a combination of a CNN and an MLP/FCN [7] as this mixture would be able to extract the spatial properties of the board using a CNN, and the non-spatial features with an MLP.

Model Tree In addition, [8] found that neural networks had drawbacks in complex domains, such as Catan, when used in reinforcement learning. This is because of the high number of games required to train the model. The solution utilised by [8] was to create a Model Tree approximator to calculate real valued functions (such as Q-functions), providing a faster learning rate with local models and discontinuities when compared to neural networks. Instead of using board positions as features, new high-level features are created to summarise important information about the board's current

[6] CDNN - Cross Dimensional Neural Network
[7] Fully Connected Network

state. Although it can train a model more quickly, there is no algorithm that can be used for online training, which can be achieved using a neural network such as the LSTM implementation by [5]. This shows that in some cases, although not a method of deep learning, hand crafted features can occasionally outperform neural networks in data-constrained settings.

Reinforcement Learning

RL trains an agent to select optimal actions in an environment by maximising cumulative rewards. This makes it suitable for teaching agents in complex games like Catan which have various ways of rewarding an agent. Rewards can be structured to reflect in-game achievements, such as building roads, settlements or winning the game. To accumulate these rewards, the agent assesses the set of all possible actions based on its current state. A reward function provides feedback for each state transition, with which the agent then aims to take an action to achieve the highest cumulative reward. The action choice is dependent on the algorithm's policy - a function which determines the probability of taking an action a, given a state s. On-policy algorithms learn from the policy the agent is currently using to act within the environment. Off-policy algorithms learn from a target policy that differs from their current policy. This is what is used to make their decisions.

Q-learning Q-learning is a popular off-policy value-based RL algorithm, which estimates expected cumulative reward (Q-value) for taking a particular action. Its popularity stems from its effectiveness given how simple it is to implement. To tackle the large action state space of Catan, [14] use an optimal policy of $\pi(s) = argmaxQ^*(s,a)$ and value estimation $Q^*(s,a) = R(s,a) + \gamma \max_{a'} Q^*(s',a')$ where $R(s,a)$ is the reward for taking a particular action at a given state and γ is the discount factor. Rather than calculating the Q-function directly, local Q-values are approximated to make use of the LSTM's recurrent nature (Q-decomposition). This is achieved using the Bellman Equation recursively: $\hat{Q}(s_t, a_t^i; \theta) = r(s_t, a_t^i) + \gamma \max_a \hat{Q}(s_{t+1}, a; \theta)$ where $i = 1, 2, ..., I$ such that $I = |A|$, A being the state of all possible actions. Each action has its own set of parameters θ^i, and so Q-values can be computed locally and in parallel. The maximum Q-value, $\hat{Q}$, and optimal action a^* are returned for the agent to make its next decision. This decomposition benefits the agent by avoiding the complexity of estimating a global Q-function over the entirety of the large action space. The model also approximates these Q-values in parallel, which improves scalability and

performance. This online algorithm trains itself during gameplay, and so no previous data is required for it to start learning. This approach provides efficient value estimation and supports temporal dependencies through recurrency, making it well suited for complex, sequential environments such as Catan. [3] used Q-learning in their own Catan agent. Similarly to [14], this implementation was designed with trading in mind, which found optimal trade requests and responses. Q-learning requires careful reward shaping to ensure the agent can learn from delayed rewards provided by its environment, but shows promise in its ability to find optimal actions.

Modified Advantage Actor Critic [5] employ a modified Advantage Actor Critic (A2C) algorithm. The agent collects experiences in parallel where 8 games are played simultaneously by 16 concurrent instances of the policy. This parallelism increases learning rate and enables diverse opponent modeling through self-training by using previous versions of itself. All versions are updated as the training progresses, but these updates are staggered in a round robin fashion so that the agent will always have variety in terms of its opponent's difficulty. The training is then nearly online, but uses slightly off-policy data. From the modified Advantage Actor Critic, we have a policy update rule (actor) given by $\nabla_\theta J_\pi(\theta) = \nabla_\theta \ln(\pi(a_t|s_t;\theta))A(s_t,a_t;\theta)$ and a value function (critic) of $\nabla_\theta J_v(\theta) = -\nabla_\theta v(s_t;\theta)(r_t+\gamma v(s_{t+1})-v(s_t))$ where the advantage is given as $A(s_t,a_t) = Q(s_t,a_t)-V(s_t) \approx r_t+\gamma v(s_{t+1})-v(s_t)$. This advantage lets the agent know how much better the action it has chosen is compared to the average action in the state. These are then combined into a total gradient of $\nabla_\theta J = \alpha_\pi \nabla_\theta J_\pi(\theta)+\alpha_v \nabla_\theta J_v(\theta)+\alpha_H \nabla_\theta \sum_a \tilde{\pi}(a|s)\ln\tilde{\pi}(a|s)+\alpha_p \nabla_\theta \sum_i p_i^2$ where H is the entropy bonus and $\alpha_p \sum_i p_i^2$ is the policy activity loss, which are added to provide improved exploration and regularisation. The modified Actor Critic provides a balance between learning a value function (critic) and directly optimising a policy (actor). The use of entropy regularisation improves exploration, and parallel rollouts allow for more efficient and diverse training through self-play.

Proximal Policy Optimisation In contrast to Q-learning, Proximal Policy Optimisation is an on-policy, policy-based RL algorithm, which learns policies directly. It incorporates Temporal Difference learning [1], offering potential improvements in learning efficiency, which could enhance decision making in the Catanatron environment [2]. It builds off of the Policy Gradient objective:

$L^{\text{PG}}(\theta) = \hat{\mathbb{E}}_t\Big[\log\pi_\theta(a_t|s_t)\hat{A}_t\Big]$ where $\pi_\theta(a_t|s_t)$ is a stochastic policy and $\hat{A}_t$ is an advantage estimate. PPO introduces a clipped surrogate objective: $L^{\text{CLIP}}(\theta) = \hat{\mathbb{E}}_t\Big[\min(r_t(\theta)\hat{A}_t, \text{clip}(r_t(\theta), 1-\epsilon, 1+\epsilon)\hat{A}_t)\Big]$, where $r_t(\theta) = \dfrac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)}$ is the probability ratio of the new policy and the old policy, and ϵ is a tuneable hyperparameter. This clipping objective is used to ensure the action probability ratio is penalised if it goes beyond a trust region of $[1-\epsilon, 1+\epsilon]$. This encourages improvement while making updates more stable, effectively solving the instability of the DQN. This clipped objective is combined with a value function to make the actor-critic objective: $L_t^{\text{CLIP + VF + S}}(\theta) = \hat{\mathbb{E}}_t\Big[L_t^{\text{CLIP}}(\theta) - c_1 L_t^{\text{VF}}(\theta) + c_2 S\big[\pi_\theta\big](s_t)\Big]$, where S denotes an entropy bonus, c_1 and c_2 are coefficients, and L_t^{VF} is a squared-error loss $(V_\theta(s_t) - V_t^{\text{targ}})^2$ [9].

Algorithm 1 PPO, Actor-Critic Style

for iteration = 1, 2, ... **do**
  for actor = 1, 2, ..., N **do**
    Run policy $\pi_{\theta_{\text{old}}}$ in environment for T timesteps
    Compute advantage estimates $\hat{A}_1, \ldots, \hat{A}_T$
  end for
  Optimise surrogate objective L w.r.t. θ, using K epochs and minibatch size $M \leq NT$
  Update parameters: $\theta_{\text{old}} \leftarrow \theta$
end for

This PPO algorithm [9] seems the most suitable for the proposed agent due to its simplicity and stability.

3 Research problem

Questions

• Can the Proximal Policy Optimisation agent in the Catanatron environment be improved with a different feature extractor?
• Can the proposed model convincingly beat three Catanatron Weighted Random bot opponents with a win rate higher than 25%?
• Can the proposed model surpass the performance of the original PPO agent in the Catanatron environment?
• Does dense reward shaping improve the performance of an agent in a large

state-action space?
• Can it be determined which feature type contributes more to whether the agent is more likely to win a game?

Hypothesis

• By replacing the MLP feature extractor of the current PPO agent in the Catanatron environment with a combined feature extractor (CNN and MLP), there can be an improvement in performance of the agent.
• The proposed agent will perform on par, if not better than the existing Catanatron agent, and will significantly outperform the Weighted Random baseline agent.
• The inclusion of dense reward shaping will significantly improve the performance of the existing PPO Catanatron agent.
• There will be a clear distinction between which feature type contributes more to the winning capability of the agent.

Aims

• Improve the existing PPO agent in the Catanatron environment using a combined feature extractor and dense reward function.
• Determine whether spatial or non-spatial features contribute more to the winning capabilities of an agent in Catan.

Objectives

• Create a combined feature extractor network which will take in separated spatial and non-spatial features of the board and provide a suitable representation of each state of the game for the PPO algorithm.
• To train both the original agent and the proposed agent in the Catanatron environment against 3 Weighted Random agents and then compare results.
• Train each agent with both sparse and dense reward functions and compare results.
• Train one agent using only the spatial features of the board, and train another using only the non-spatial features and then to compare results.

4 Methodology

4.1 Environment

The experiment was conducted in the Catanatron [2] gymnasium environment [12]. Many aspects of the digital board can be altered, including the size as well as how many victory points are required to win. For faster training, the "MINI" environment was used with 6 Victory Points needed to win a game. The MINI environment uses 7 tiles - wood, desert (where no resource is gained), brick, sheep, wheat, another wheat, and ore (Figure 1). This excludes any ports. Given the smaller domain, the agent can explore with fewer victory requirements, thus reducing the time taken to complete each game played. Due to resource constraints in processing power, this made the smaller map size crucial for training.

Figure 1: A visual representation of the MINI board with randomly selected hex positions.

The configuration for the environment can be set with the following parameters:

```
configuration = {
 "map_type": map_type,
 "vps_to_win": vps_to_win,
 "enemies": enemies,
 "reward_function": reward_function,
 "representation": representation,
}
```

In this case map_type = MINI, vps_to_win = 6, enemies = Weighted Random x 3, reward function = sparse/dense, representation = mixed/vector.

4.2 Agents

The features of the board are represented in two ways: vector and mixed. Both agents are named after the representation of the board they use and will be compared to one another after being trained on, and evaluated with, 3 Weighted Random agents. The vector agent can be found in the Catanatron documentation, and acts as the basis on which the mixed agent was built. The agent uses the MaskableActorCritic policy, paired with the MaskablePPO algorithm from the sb3-contrib (Stable Baselines) reinforcement learning library. This version of the PPO algorithm allows for masks to be applied to invalid actions, making learning much faster than if the agent had to learn which actions could and couldn't be taken at any particular state. This is done by multiplying all invalid action probabilities by 0, and valid action probabilities by 1.

4.2.1 Weighted Random

The Weighted Random agent, provided by the environment, randomly selects actions from observations but places a bigger emphasis on building cities, settlements and development cards. It does so by skewing the random distribution of action selection in favour of these actions by multiplying their probabilities by a constant value. This makes for a decent player with which to benchmark the vector and mixed agents because of its ability to outperform the Monte-Carlo Tree Search player also provided in the environment. This can be tested using the Catanatron CLI provided in the Catanatron documentation.

4.2.2 Vector

This is the base model used in the Catanatron gymnasium, found here. Using an observation of shape (1002,), the features are fed into two separate feature extractor networks (Figure 2). When unspecified in the model hyperparameters, the agent defaults to using two separate MLP extractors which pass their results directly to the action (policy) and value networks. Updating the weights of each network separately allows for better feature extraction as each network has targeted updates for their own specific tasks. In this case, that would be predicting the actions (action network) and predicting how good an action is (value network). Although a single backbone network is less computationally expensive, it does not have the same flexibility as two different networks. The agent uses a sparse reward function by default, but both the sparse and dense reward functions were used in training and evaluation for a full comparison.

4.2.3 Mixed

The purpose of the mixed agent is to separate the features of the board into different feature extractors which are better suited for their respective data types. That is, a CNN would preserve the spatial data of the 3D tensors of the board, whereas an MLP would be much more efficient with non-spatial data. Making use of the mixed observations of the board, the agent separates feature extraction using a CNN for the 3D board tensors with dimensions (C=20, W=21, H=11) representing the physical features, and an MLP which accepts an observation shape of (76,) for the non-physical features (Figure 3). Once the features have been processed by their respective networks, they are then concatenated and fed into the action and value networks. Unlike the vector agent, each head does not have its own extractor network, but rather one shared backbone network. The reason for this is to save computing resources. The second change made to the agent is the reward function used when training in the environment. It can be difficult for an RL agent to make successful updates to its network when it doesn't know what moves led it to success. The dense reward function aims to mend this by providing rewards for moves that lead the agent into a winning state. For training, both the sparse and dense reward functions were used.

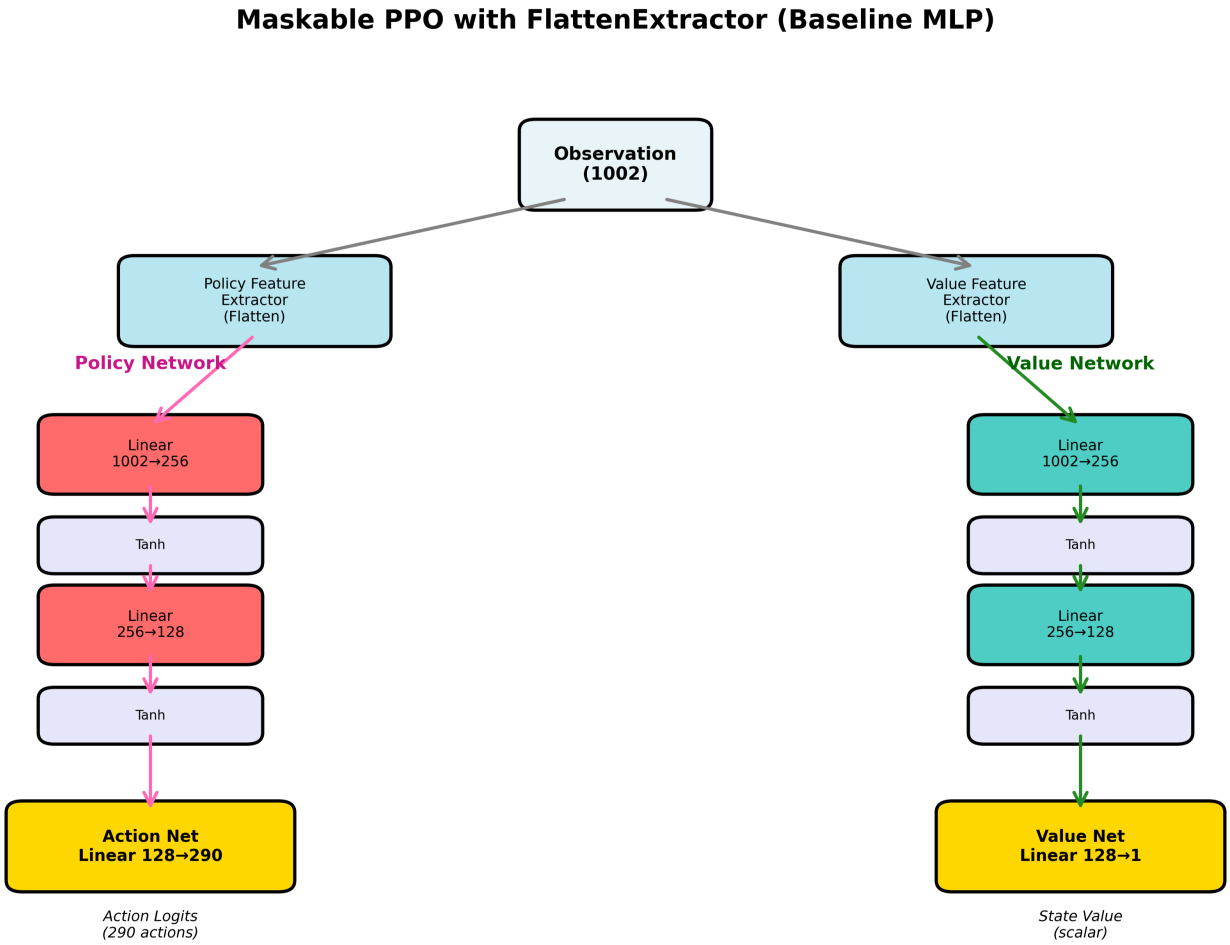

Figure 2: Visual representation of the vector agent.

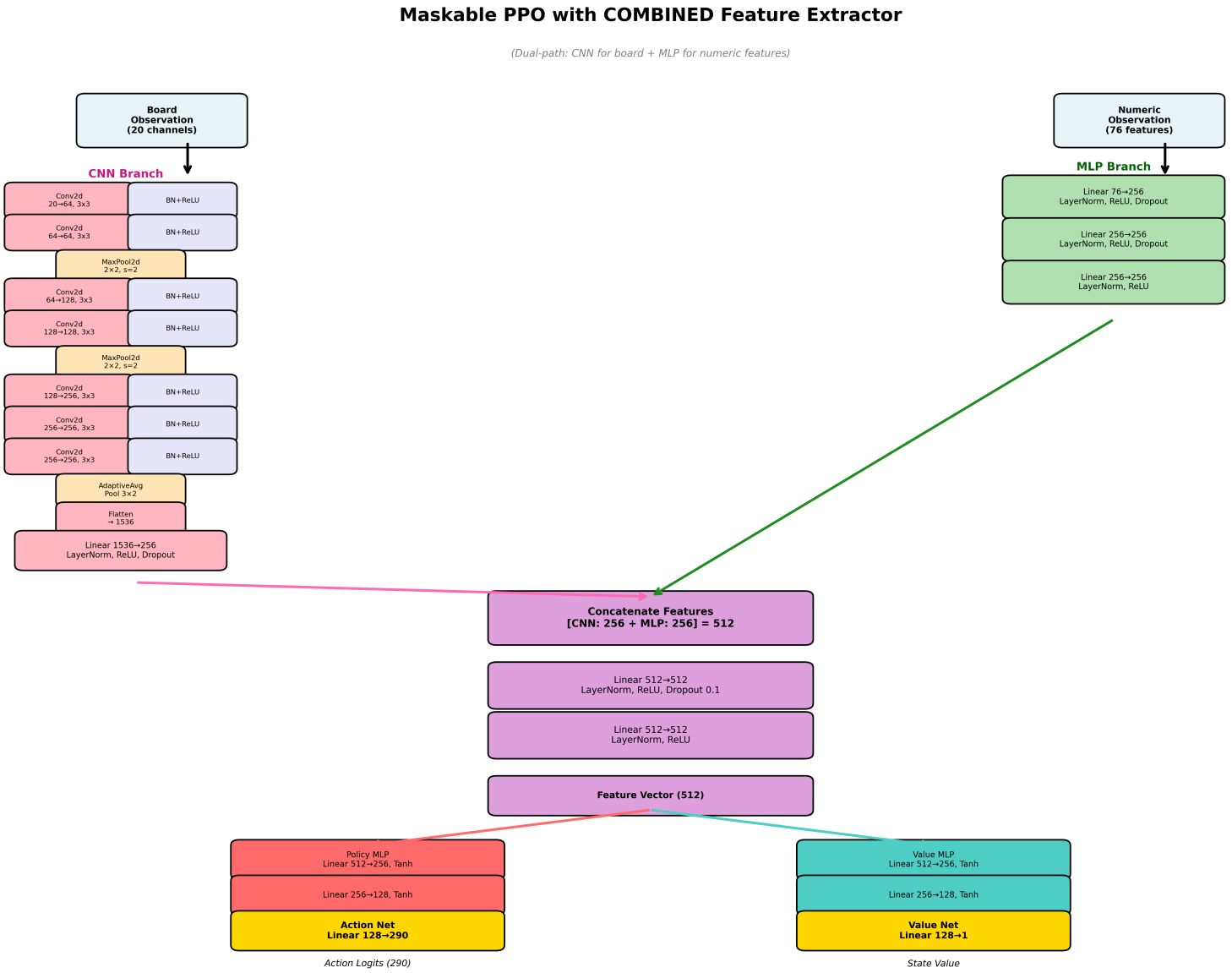

Figure 3: Visual representation of the mixed agent.

4.2.4 Numeric and Board only

The Numeric and Board only agents use a split version of the mixed agent network architecture - the Numeric only agent (Figure 4) uses the MLP feature extractor, whereas the Board only agent (Figure 5) uses the CNN feature extractor. Similarly to the mixed agent, they share one backbone extractor network. The purpose of these agents is to determine which feature makes a greater impact on the performance of a Catan playing agent and to determine whether it relies more on the spatial or non-spatial features.

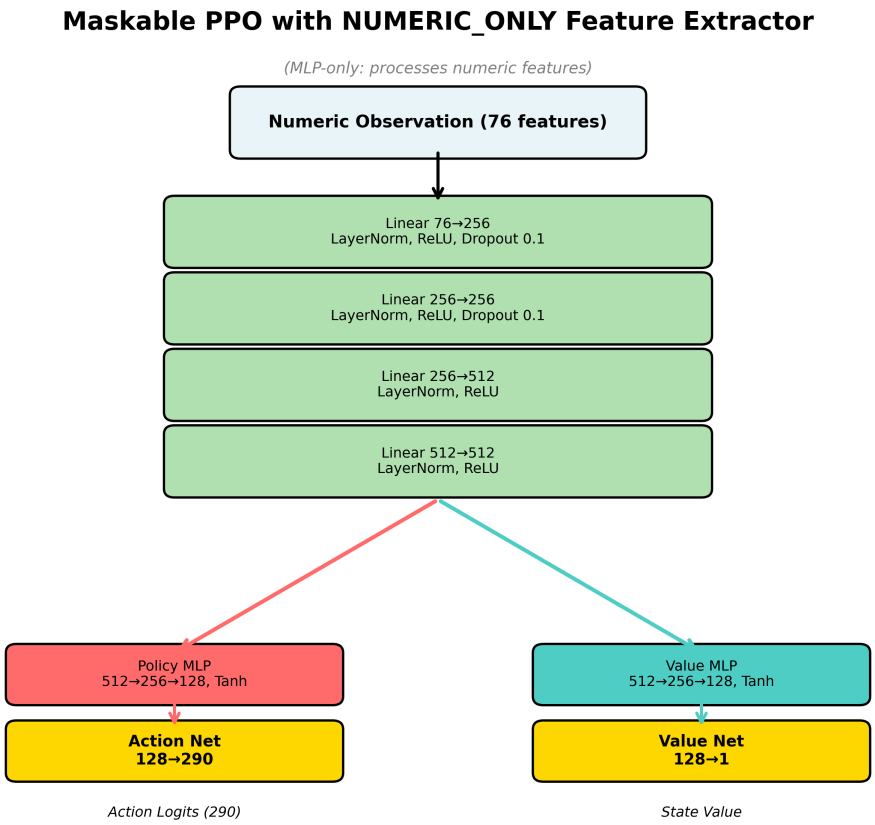

Figure 4: Visual representation of the numeric only agent.

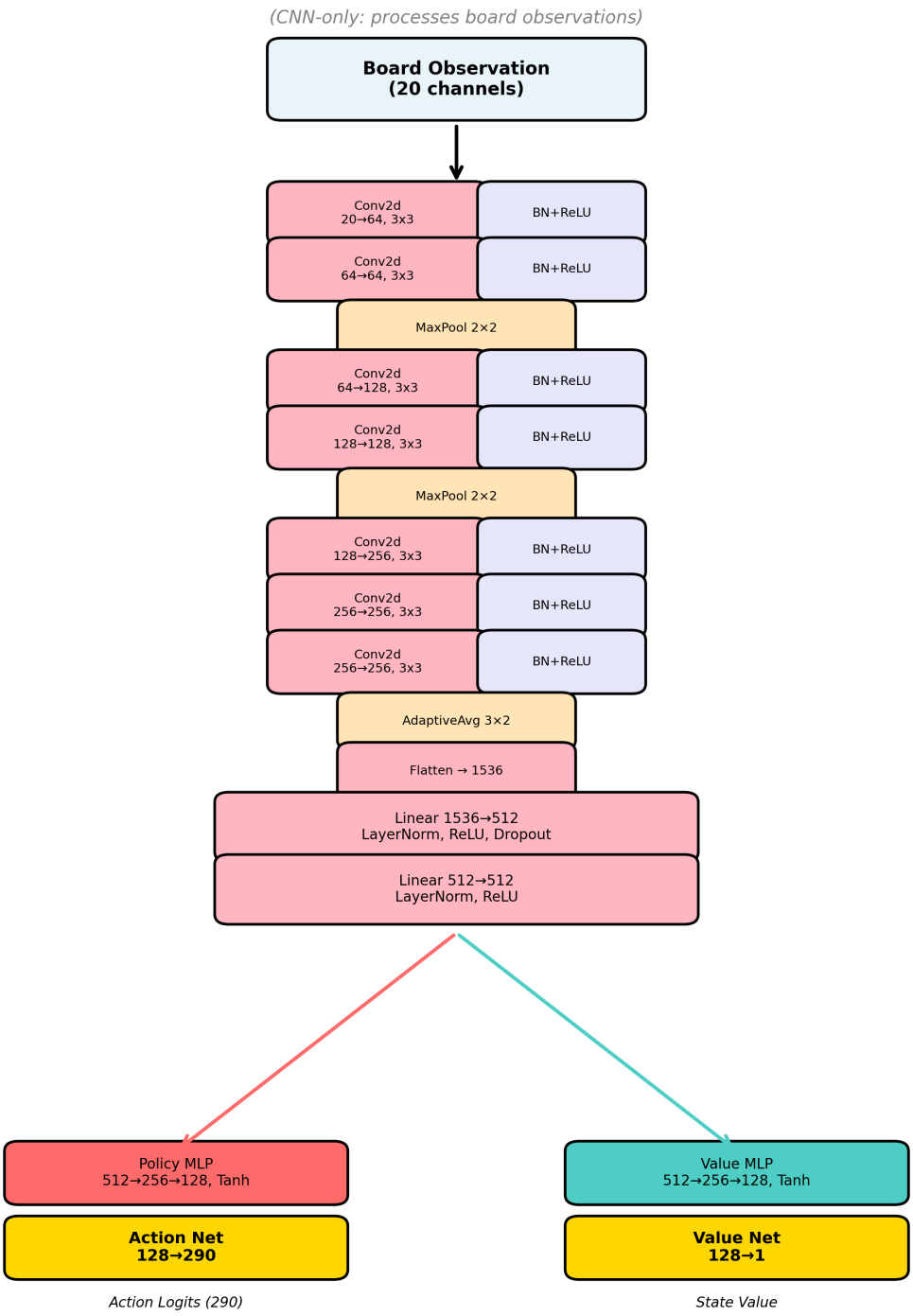

Figure 5: Visual representation of the board only agent.

4.3 Reward Shaping

The original sparse reward function gave the agent 100 for winning, -100 for losing and 0 otherwise. Catan's large state space can make sparse rewards less effective at teaching the agent which actions are more favourable during the game as it only knows how well it is doing when the game is already over. Favourable actions include whether the agent had used any development cards that round, placed a city/settlement/road, gained achievements such as longest road or largest army and whether it gained a victory point. To remedy the lack of rewards for these actions, a new dense reward function was used to further improve the base implementation of the agent. Iteration one of this reward function mainly used negative rewards which were continuously given to the agent throughout the game and were reduced as it made progress by placing settlements and cities. Performance did not improve compared to when sparse rewards were given. Iteration two was thus a more positive reward function, in case the negative rewards resulted in shorter matches so that the agent could preserve its total reward. The new function congratulated the agent for making progress, rather than penalising it for making unfavourable decisions. The problem with the second version, similarly to the first, was that it continuously gave the agent rewards long after the action it had taken to achieve them, possibly creating noise for the agent. This is because the observations were dependent on each state of the game and so if a value was updated, it would remain changed until the end. Version three fixed this by reading and writing to a file, and treating it as secondary storage. This meant that the file would be read and changed only if there was a write to the file. The agent would get the reward, update the file and then read it again in the next timestep to see if there were any changes and then update rewards accordingly.

4.4 Training and Evaluation

To see how the agents learn over time, each model was pitted against 3 Weighted Random bots and trained on increasing timesteps: 125 000, 250 000, 500 000 and 1 000 000. This was repeated 10 times to achieve a realistic average in a stochastic environment, and was conducted with dense and sparse reward functions for each model. Although a callback function [8] would have been useful to reduce the total training time, there were issues with its implementation related to the maskable environment and so it was

[8] A callback function allows a training model to intermittently be evaluated, saving the version of the model that performs the best.

not used. After training, each model was evaluated with another 3 Weighted Random agents over 1 000 epochs. The winrate in percentage and training time [9] were recorded, averaged and compared for each model. All training data collected can be found in Tables 3 - 10.

Two runs of the mixed agent with dense rewards were recorded at 2 000 000 timesteps, with winrates 54.90% and 57.50%. It took twice as long to run as 1 000 000 timesteps but showed similar results. For this reason, and limited computing resources, 1 000 000 timesteps seemed a suitable stopping point for recording performance.

4.5 Computer Specifications

The agent was trained on a computer with the following specifications:

- CPU
  - 12th Gen Intel(R) Core(TM) i5-12400
  - Base speed: 2,50 GHz
  - Cores: 6
- Graphics Card
  - NVIDIA GeForce RTX 3060
  - Dedicated GPU memory: 12,0 GB
- Memory
  - Primary: 32,0 GB DDR4-3600MHz

[9] Table 3 and Table 5 do not include all training times, as this was only implemented after these values were recorded. For average training time, the 5 recorded values for each timestep were averaged instead of 10.

5 Results and Discussion

The improved reward function outperforms the previous version at every point, showing how it significantly improved the agent's performance (Figure 6). Not only is the winrate of the agent higher, but the narrower standard deviation band is much closer to the average, showing a lower variance in the data. For this reason, it was used in place of the previous version. Collected data is show in the Appendix Table 2.

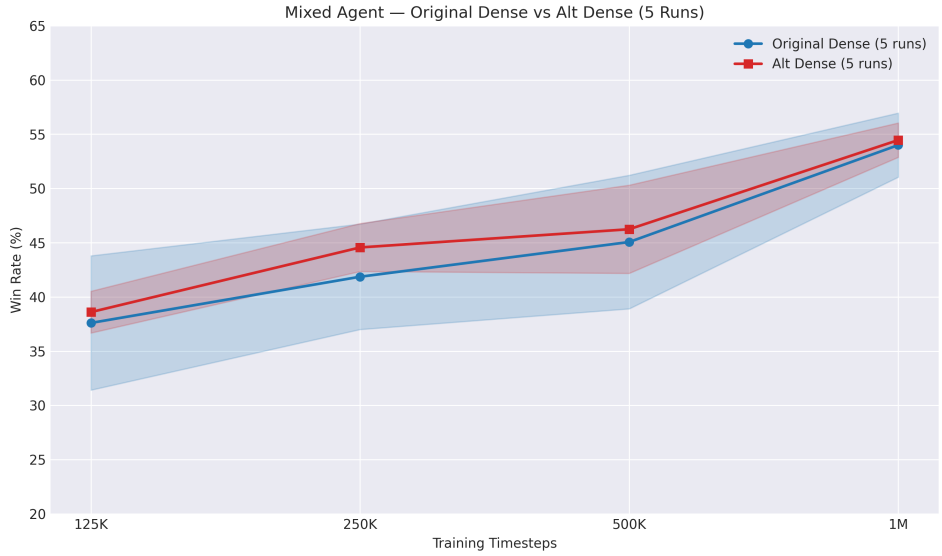

Figure 6: Comparison of the previous and improved dense reward functions.

As described above, the dense reward function reduces the variance of the mixed agent, creating a more predictable outcome when assessed against the sparse reward function. However, this is not the case with the vector agent where the standard deviation bands reveal similar results for both agents. This could suggest that the dense reward function is better suited for the architecture of the mixed agent rather than the vector agent. In both cases, the agent using the dense reward function performs the best overall and places above the sparse variants after 1 000 000 timesteps (Figure 7).

Here we also see that the mixed agent, although starting with a lower winrate at 125 000 timesteps, surpasses the vector agent before 1 000 000 timesteps in both cases, only doing so more convincingly with the dense reward function. This further illustrates the advantage of the improved reward function paired with the mixed agent over the sparse reward function. The initial lower winrate is most likely due to increased number of weights

that need to be updated in the network compared to the vector agent, which leads to slower initial learning.

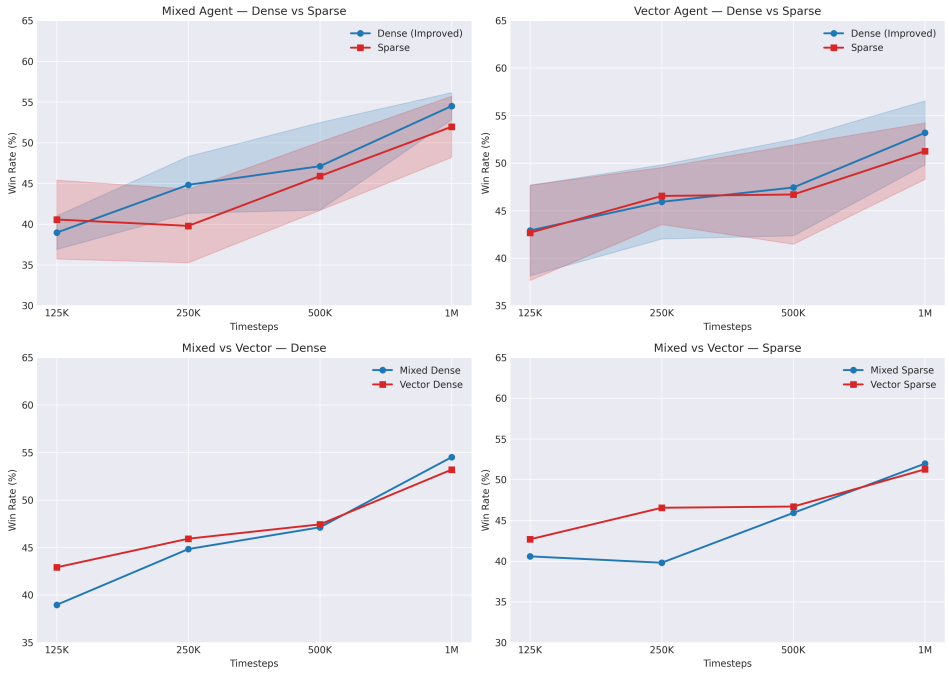

Figure 7: A collection of performance graphs, illustrating the win rates over different timesteps for the mixed and vector agents, alternating between the dense and sparse reward functions.

When comparing all models (Figure 8), the mixed dense agent has the lowest starting winrate, but eventually exceeds the performance of all other agents by the end of training. The dense rewards increased the winrate of the mixed agent significantly after 250 000 timesteps, which could imply that the dense rewards improved the agent's ability to take favourable actions much sooner. The vector agents however had very similar performance throughout training until 500 000 timesteps where the dense variant pulled ahead. The mixed agents beat both vector models in their respective reward functions, showing a similar trend from 500 000 timesteps where the mixed agents pass the vector agents - the dense variants elevated relative to their sparse counterparts.

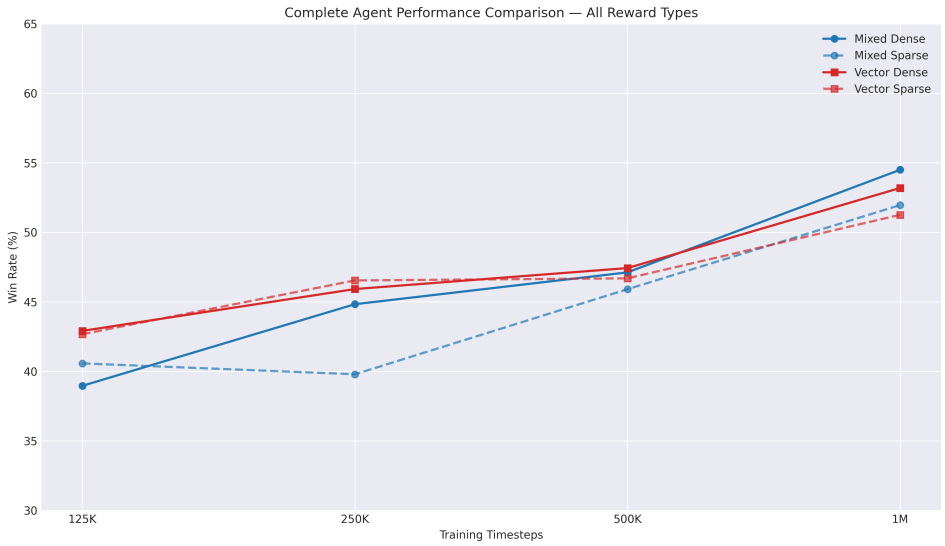

Figure 8: Figure comparing all mixed and vector agents with sparse and dense rewards.

The numeric only dense agent saw an increased variance throughout the different timesteps when compared to its sparse reward counterpart (Figure 9). This is the opposite of what was seen with the mixed dense agent, whose variance was reduced. A possible reason for these stark differences in variance could be the way the dense rewards affect the weights of the CNN and MLP extractors. One explanation could be that the spatial features are updated with small changes applied after actions affecting the board, whereas the non-spatial features are changed more often, resulting in more drastic updates and therefore a major change in performance.

Unlike the mixed and vector agents, the board and numeric only models have a very similar learning trend over time, irrelevant of which reward function they use - the board only agent starts with a higher winrate at 125 000 timesteps, is then surpassed in performance by the numeric agent at 250 000 timesteps, and then ultimately beats the numeric only agent shortly after. Similarly to the mixed dense agent, the numeric dense agent had a significant jump in performance at the 250 000 timesteps mark. This further illustrates the possibility of the dense rewards causing the agent to make more favourable moves earlier on in training.

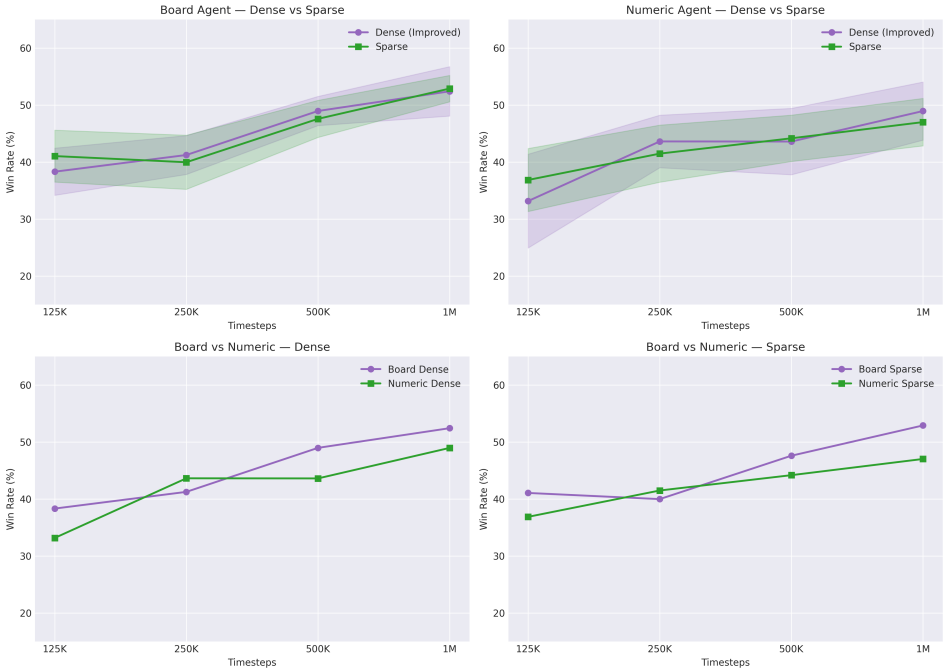

Figure 9: Comparison of the numeric and board only agents with sparse and dense rewards.

The comparison of the board and numeric models (Figure 10) shows that the board only agent performs significantly better than the numeric only agent with both the dense and sparse reward functions. The board sparse model is the first agent using the sparse reward function to outperform the dense reward variant, albeit not by much. There is a symmetry in the resulting graph with the winrate achieved at 125 000 timesteps by each model, almost matching the results obtained at 1 000 000 timesteps.

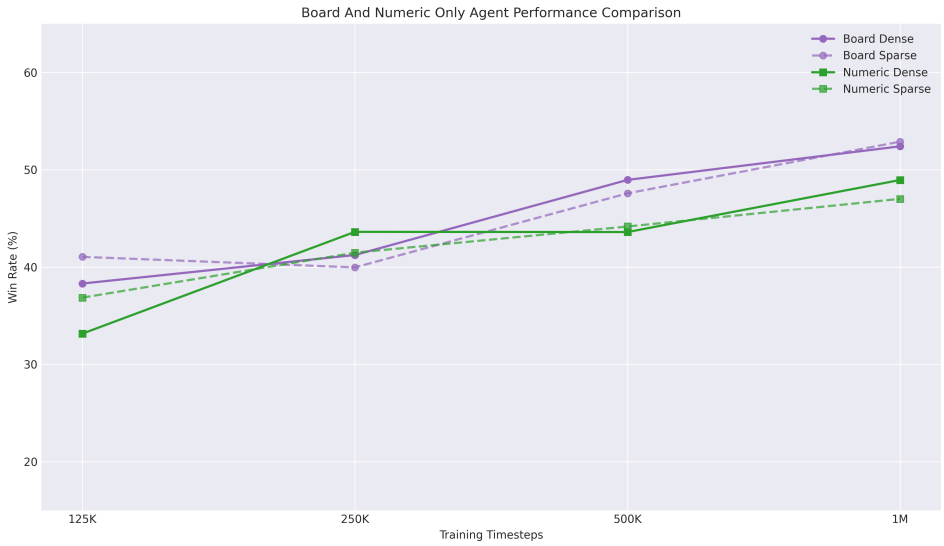

Figure 10: Full comparison of numeric and board models.

The mixed dense agent had the highest performance overall as well as the lowest variance (Table 1), surpassing the performance of the original vector sparse agent by a total of $\frac{54.5 - 51.26}{51.26} \cdot 100 = 6.32\%$, and substantially beating the aim of a 25% winrate against 3 Weighted Random bots. This is a significant improvement in performance in a stochastic 3 player environment and shows the benefits of adding both the change to the feature extractor and to the reward function when compared to the original model. For individual performance of adding the changed feature extractor, there was an improvement of $\frac{51.96 - 51.26}{51.26} \cdot 100 = 1.37\%$ and for the change in reward function, there was a improvement of $\frac{53.19 - 51.26}{51.26} \cdot 100 = 3.77\%$. This demonstrates that adding a dense reward function does significantly improve the base vector sparse agent. The results show that the majority of the models which used dense rewards improved their sparse reward counterparts (3 of 4). The dense rewards decreased the variance of the mixed agent, but increased the variance of the board and numeric only models. This is a counter-intuitive result because it would suggest that the feature extractors separated have a high variance individually, but a low variance when paired together (Figure 11). The board dense agent beat the numeric dense agent (better perform-

ing of the two numeric agents) by $\frac{52.41 - 48.95}{48.95} \cdot 100 = 7.07\%$, with the board sparse beating it by $\frac{52.89 - 48.95}{48.95} \cdot 100 = 8.05\%$. Clearly, the board features had a much bigger influence on the final result of the agent's performance. This could be due to the substantial disparity in number of features (20 x 21 x 11 = 4620 compared to 72), and so the numeric agent has far less information to utilise when deciding which actions are more favourable. The vector sparse and mixed dense models had a major difference in time taken to train at 1 000 000 timesteps, with the mixed model taking 123.91 - 79.22 = 44.69 minutes longer to train than the original vector sparse model (Table 1).

Agent	**Avg. Win Rate (1M)**	**Avg. Train Time (min) (1M)**
Mixed Dense	54.50% ± 1.65	123.91 ± 3.28
Mixed Sparse	51.96% ± 3.75	126.34 ± 4.79
Vector Dense	53.19% ± 3.36	77.13 ± 2.26
Vector Sparse	51.26% ± 2.96	79.22 ± 5.88
Board Dense	52.41% ± 4.33	117.36 ± 5.42
Board Sparse	52.89% ± 2.31	112.51 ± 6.03
Numeric Dense	48.95% ± 5.09	94.05 ± 1.18
Numeric Sparse	47.01% ± 4.16	87.30 ± 1.24

Table 1: Average win rates and training times at 1M timesteps with standard deviation.

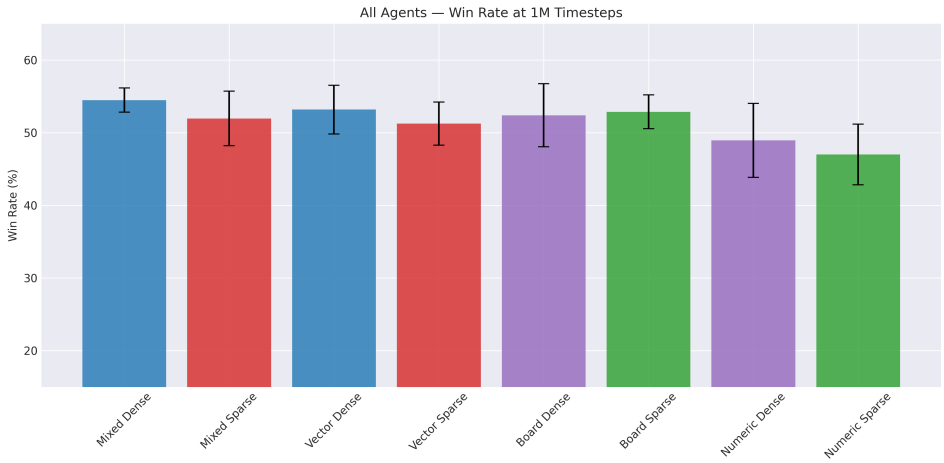

Figure 11: Bar graph of all agents with standard deviation lines.

The board dense model performed nearly as well as the vector dense agent but had a much larger interquartile range. This shows that the results were significantly more spread out in the board dense model, which demonstrates the different effects the dense rewards had on the various agents. The board sparse model performed slightly better than the board dense agent, and slightly worse than the vector dense agent, but showed an improved, narrower spread of values compared to both other models (Figure 12).

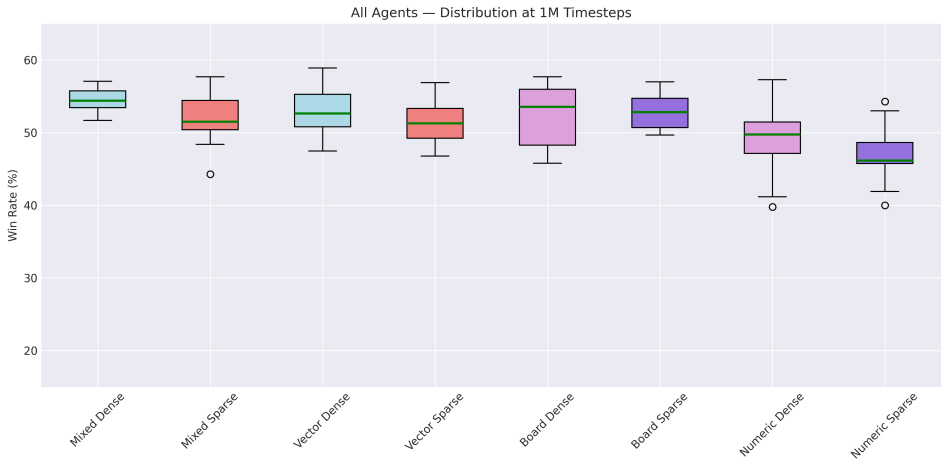

Figure 12: Box plots of all agents.

6 Conclusion

Catan's state-action space is complex, making it difficult for an RL agent to learn favourable actions effectively. The choice of feature extractor to better suit the feature data type can make a positive difference to the overall performance of a reinforcement learning agent. A dense reward function also provides an increased performance, but when combined, these two changes can make a larger improvement than what they could separately. The benefit of correct choice in feature extraction and reward shaping for improved performance is therefore evident. The mixed dense agent beat the original vector sparse agent by 6.32%, showing a significant change in performance. It can also be seen that the spatial features of the Catan board play a larger role in an agent's ability to successfully win Catan games, possibly due to the vast difference in feature count. The board sparse agent had a performance improvement of 8.05% over the better of the numeric only models, demonstrating the importance of spatial features when training a Catan-playing agent.

Appendix

The github repository for the project can be found here.

Agent	RF	T	Run	WR (%)	Avg/Min/Max Rewards	W/L
Mixed Agent	Previous Dense Reward Function	125,000	1	43.30	2.53 / -107.81 / 180.51	433 / 567
			2	41.80	1.09 / -107.42 / 224.33	418 / 582
			3	36.80	-10.21 / -105.74 / 203.72	368 / 632
			4	40.10	-3.26 / -108.83 / 211.79	401 / 599
			5	26.00	-35.69 / -110.26 / 225.03	260 / 740
		250,000	1	32.40	-21.28 / -111.78 / 273.25	324 / 676
			2	46.40	9.96 / -106.52 / 206.72	464 / 536
			3	43.40	3.61 / -108.78 / 189.56	434 / 566
			4	43.60	4.56 / -110.18 / 237.33	436 / 564
			5	43.50	4.07 / -106.62 / 212.64	435 / 565
		500,000	1	47.70	15.07 / -106.17 / 247.54	477 / 523
			2	51.70	21.00 / -108.07 / 243.75	517 / 483
			3	47.70	15.17 / -108.07 / 220.85	477 / 523
			4	44.60	5.55 / -107.47 / 213.57	446 / 554
			5	33.60	-19.02 / -120.81 / 199.65	336 / 664
		1,000,000	1	58.50	37.41 / -105.09 / 225.52	585 / 415
			2	53.80	26.97 / -108.98 / 224.63	538 / 462
			3	50.40	20.38 / -109.91 / 225.67	504 / 496
			4	51.40	21.76 / -107.80 / 217.76	514 / 486
			5	55.90	31.18 / -106.77 / 267.01	559 / 441
	Improved Dense Reward Function	125,000	1	39.70	-4.51 / -100.30 / 126.60	397 / 603
			2	36.20	-11.96 / -106.00 / 125.90	362 / 638
			3	37.30	-9.68 / -104.20 / 125.40	373 / 627
			4	41.70	0.57 / -99.00 / 128.10	417 / 583
			5	38.10	-7.45 / -107.20 / 126.30	381 / 619
		250,000	1	43.90	4.50 / -101.90 / 125.50	439 / 561
			2	42.20	0.99 / -103.30 / 125.20	422 / 578
			3	42.60	1.15 / -117.70 / 128.30	426 / 574
			4	48.10	12.98 / -105.20 / 127.60	481 / 519
			5	46.00	8.60 / -105.00 / 126.40	460 / 540
		500,000	1	45.50	7.83 / -107.80 / 126.60	455 / 545
			2	46.30	8.97 / -100.00 / 127.50	463 / 537
			3	39.10	-5.74 / -103.50 / 126.00	391 / 609
			4	50.70	18.73 / -96.50 / 125.80	507 / 493
			5	49.60	16.01 / -102.30 / 125.60	496 / 504
		1,000,000	1	54.50	26.49 / -105.60 / 126.60	545 / 455
			2	55.30	28.54 / -95.60 / 126.50	553 / 447
			3	56.50	30.72 / -95.20 / 126.20	565 / 435
			4	51.70	21.45 / -106.60 / 127.20	517 / 483
			5	54.30	26.34 / -102.90 / 126.60	543 / 457

Table 2: Comparison of results obtained by the mixed agent using the previous and improved dense reward functions.

Agent	T	Run	WR (%)	Avg/Min/Max Rewards	W/L	TT (min)
Vector Agent (Sparse Reward Function)	125,000	1	37.40	-25.20 / -100.00 / 100.00	374 / 626	
		2	41.00	-18.00 / -100.00 / 100.00	410 / 590	
		3	44.40	-11.20 / -100.00 / 100.00	444 / 556	
		4	32.20	-35.60 / -100.00 / 100.00	322 / 678	
		5	44.80	-10.40 / -100.00 / 100.00	448 / 552	
		6	40.10	-19.80 / -100.00 / 100.00	401 / 599	9.6
		7	44.70	-10.60 / -100.00 / 100.00	447 / 553	10.0
		8	50.20	0.40 / -100.00 / 100.00	502 / 498	9.5
		9	43.20	-13.60 / -100.00 / 100.00	432 / 568	10.0
		10	48.70	-2.60 / -100.00 / 100.00	487 / 513	10.7
	250,000	1	48.80	-2.40 / -100.00 / 100.00	488 / 512	
		2	45.00	-10.00 / -100.00 / 100.00	450 / 550	
		3	49.90	-0.20 / -100.00 / 100.00	499 / 501	
		4	46.50	-7.00 / -100.00 / 100.00	465 / 535	
		5	47.00	-6.00 / -100.00 / 100.00	470 / 530	
		6	42.40	-15.20 / -100.00 / 100.00	424 / 576	18.3
		7	41.70	-16.60 / -100.00 / 100.00	417 / 583	17.9
		8	44.30	-11.40 / -100.00 / 100.00	443 / 557	17.5
		9	51.10	2.20 / -100.00 / 100.00	511 / 489	18.2
		10	48.70	-2.60 / -100.00 / 100.00	487 / 513	17.7
	500,000	1	41.30	-17.40 / -100.00 / 100.00	413 / 587	
		2	52.70	5.40 / -100.00 / 100.00	527 / 473	
		3	44.30	-11.40 / -100.00 / 100.00	443 / 557	
		4	53.00	6.00 / -100.00 / 100.00	530 / 470	
		5	52.00	4.00 / -100.00 / 100.00	520 / 480	
		6	41.60	-16.80 / -100.00 / 100.00	416 / 584	39.4
		7	53.50	7.00 / -100.00 / 100.00	535 / 465	39.9
		8	42.90	-14.20 / -100.00 / 100.00	429 / 571	37.8
		9	39.90	-20.20 / -100.00 / 100.00	399 / 601	37.9
		10	45.70	-8.60 / -100.00 / 100.00	457 / 543	38.0
	1,000,000	1	51.50	3.00 / -100.00 / 100.00	515 / 485	
		2	53.50	7.00 / -100.00 / 100.00	535 / 465	
		3	47.50	-5.00 / -100.00 / 100.00	475 / 525	
		4	46.80	-6.40 / -100.00 / 100.00	468 / 532	
		5	53.80	7.60 / -100.00 / 100.00	538 / 462	
		6	56.90	13.80 / -100.00 / 100.00	569 / 431	78.5
		7	49.20	-1.60 / -100.00 / 100.00	492 / 508	86.5
		8	52.90	5.80 / -100.00 / 100.00	529 / 471	85.4
		9	49.40	-1.20 / -100.00 / 100.00	494 / 506	73.0
		10	51.10	2.20 / -100.00 / 100.00	511 / 489	72.7

Table 3: Vector Sparse agent results. Training times only recorded for runs 6-10.

Agent	T	Run	WR (%)	Avg/Min/Max Rewards	W/L	TT (min)
Vector Agent (Dense Reward Function)	125,000	1	43.30	2.75 / -107.60 / 124.50	433 / 567	10.1
		2	50.30	17.51 / -99.40 / 126.00	503 / 497	10.3
		3	45.60	8.46 / -96.30 / 127.20	456 / 544	10.2
		4	35.40	-13.58 / -101.30 / 125.30	354 / 646	9.4
		5	49.20	15.84 / -103.40 / 126.00	492 / 508	10.8
		6	46.00	8.85 / -99.50 / 127.20	460 / 540	10.5
		7	41.60	-0.38 / -104.00 / 126.00	416 / 584	10.6
		8	36.10	-11.61 / -105.00 / 125.90	361 / 639	9.6
		9	39.70	-4.52 / -112.00 / 126.40	397 / 603	10.7
		10	41.90	0.46 / -99.40 / 126.00	419 / 581	11.3
	250,000	1	42.10	1.01 / -99.50 / 125.50	421 / 579	20.9
		2	45.80	8.53 / -104.60 / 127.20	458 / 542	19.8
		3	37.50	-9.13 / -106.20 / 127.90	375 / 625	20.5
		4	49.00	14.80 / -102.60 / 127.10	490 / 510	21.3
		5	46.00	8.31 / -108.00 / 127.00	460 / 540	20.7
		6	52.40	22.33 / -98.70 / 126.80	524 / 476	21.0
		7	44.40	5.20 / -107.40 / 125.70	444 / 556	20.4
		8	46.80	10.64 / -96.90 / 126.90	468 / 532	20.7
		9	45.80	8.53 / -93.20 / 127.30	458 / 542	21.1
		10	49.40	16.43 / -97.60 / 126.00	494 / 506	20.9
	500,000	1	48.20	13.58 / -95.70 / 126.30	482 / 518	44.6
		2	50.20	17.36 / -108.30 / 126.30	502 / 498	45.8
		3	51.40	19.36 / -102.50 / 127.90	514 / 486	43.6
		4	41.40	-1.62 / -108.30 / 126.00	414 / 586	49.3
		5	50.20	16.89 / -108.60 / 128.40	502 / 498	46.7
		6	43.10	2.22 / -109.70 / 127.40	431 / 569	41.9
		7	50.40	17.47 / -115.80 / 126.50	504 / 496	42.1
		8	36.20	-12.21 / -106.40 / 126.00	362 / 638	38.2
		9	52.30	21.71 / -94.80 / 129.50	523 / 477	42.5
		10	50.90	19.03 / -103.40 / 127.00	509 / 491	41.8
	1,000,000	1	54.30	25.41 / -104.10 / 126.60	543 / 457	75.2
		2	50.70	18.81 / -97.20 / 126.00	507 / 493	75.6
		3	58.90	35.48 / -98.70 / 127.60	589 / 411	74.8
		4	57.90	33.27 / -99.50 / 128.60	579 / 421	77.6
		5	55.60	28.51 / -106.10 / 126.90	556 / 444	83.0
		6	50.80	19.35 / -100.30 / 125.80	508 / 492	78.8
		7	53.40	23.92 / -117.10 / 125.60	534 / 466	76.3
		8	51.90	21.02 / -98.30 / 126.00	519 / 481	76.2
		9	47.50	11.36 / -107.70 / 127.30	475 / 525	77.4
		10	50.90	18.74 / -109.10 / 125.40	509 / 491	76.4

Table 4: Vector Dense agent results.

Agent	T	Run	WR (%)	Avg/Min/Max Rewards	W/L	TT (min)
Mixed Agent (Sparse Reward Function)	125,000	1	41.30	-17.40 / -100.00 / 100.00	413 / 587	
		2	46.80	-6.40 / -100.00 / 100.00	468 / 532	
		3	32.50	-35.00 / -100.00 / 100.00	325 / 675	
		4	44.00	-12.00 / -100.00 / 100.00	440 / 560	
		5	32.80	-34.40 / -100.00 / 100.00	328 / 672	
		6	40.80	-18.40 / -100.00 / 100.00	408 / 592	16.7
		7	44.20	-11.60 / -100.00 / 100.00	442 / 558	19.6
		8	35.90	-28.20 / -100.00 / 100.00	359 / 641	16.3
		9	42.40	-15.20 / -100.00 / 100.00	424 / 576	16.3
		10	45.00	-10.00 / -100.00 / 100.00	450 / 550	16.7
	250,000	1	41.10	-17.80 / -100.00 / 100.00	411 / 589	
		2	38.60	-22.80 / -100.00 / 100.00	386 / 614	
		3	36.00	-28.00 / -100.00 / 100.00	360 / 640	
		4	41.90	-16.20 / -100.00 / 100.00	419 / 581	
		5	40.10	-19.80 / -100.00 / 100.00	401 / 599	
		6	44.90	-10.20 / -100.00 / 100.00	449 / 551	32.7
		7	36.80	-26.40 / -100.00 / 100.00	368 / 632	32.8
		8	47.40	-5.20 / -100.00 / 100.00	474 / 526	32.1
		9	40.70	-18.60 / -100.00 / 100.00	407 / 593	32.0
		10	30.40	-39.20 / -100.00 / 100.00	304 / 696	31.0
	500,000	1	54.10	8.20 / -100.00 / 100.00	541 / 459	
		2	38.40	-23.20 / -100.00 / 100.00	384 / 616	
		3	45.80	-8.40 / -100.00 / 100.00	458 / 542	
		4	50.60	1.20 / -100.00 / 100.00	506 / 494	
		5	44.50	-11.00 / -100.00 / 100.00	445 / 555	
		6	45.90	-8.20 / -100.00 / 100.00	459 / 541	60.8
		7	42.90	-14.20 / -100.00 / 100.00	429 / 571	61.0
		8	45.10	-9.80 / -100.00 / 100.00	451 / 549	61.0
		9	49.00	-2.00 / -100.00 / 100.00	490 / 510	60.7
		10	42.80	-14.40 / -100.00 / 100.00	428 / 572	60.4
	1,000,000	1	54.60	9.20 / -100.00 / 100.00	546 / 454	
		2	54.00	8.00 / -100.00 / 100.00	540 / 460	
		3	57.70	15.40 / -100.00 / 100.00	577 / 423	
		4	50.30	0.60 / -100.00 / 100.00	503 / 497	
		5	51.90	3.80 / -100.00 / 100.00	519 / 481	
		6	51.10	2.20 / -100.00 / 100.00	511 / 489	122.2
		7	50.80	1.60 / -100.00 / 100.00	508 / 492	123.0
		8	56.50	13.00 / -100.00 / 100.00	565 / 435	122.8
		9	44.30	-11.40 / -100.00 / 100.00	443 / 557	129.3
		10	48.40	-3.20 / -100.00 / 100.00	484 / 516	134.4

Table 5: Mixed Sparse agent results. Training times only recorded for runs 6-10.

Agent	T	Run	WR (%)	Avg/Min/Max Rewards	W/L	TT (min)
Mixed Agent (Dense Reward Function)	125,000	1	39.70	-4.51 / -100.30 / 126.60	397 / 603	16.7
		2	36.20	-11.96 / -106.00 / 125.90	362 / 638	16.4
		3	37.30	-9.68 / -104.20 / 125.40	373 / 627	15.6
		4	41.70	0.57 / -99.00 / 128.10	417 / 583	17.2
		5	38.10	-7.45 / -107.20 / 126.30	381 / 619	17.0
		6	37.80	-8.20 / -100.00 / 124.90	378 / 622	18.0
		7	42.60	2.12 / -98.30 / 128.00	426 / 574	18.2
		8	37.60	-8.27 / -102.40 / 127.30	376 / 624	17.8
		9	41.10	-1.18 / -96.10 / 125.90	411 / 589	16.8
		10	37.50	-8.93 / -108.60 / 126.70	375 / 625	17.3
	250,000	1	43.90	4.50 / -101.90 / 125.50	439 / 561	32.9
		2	42.20	0.99 / -103.30 / 125.20	422 / 578	31.0
		3	42.60	1.15 / -117.70 / 128.30	426 / 574	31.2
		4	48.10	12.98 / -105.20 / 127.60	481 / 519	33.3
		5	46.00	8.60 / -105.00 / 126.40	460 / 540	34.9
		6	39.20	-5.77 / -102.60 / 125.50	392 / 608	34.0
		7	42.40	0.98 / -103.00 / 126.50	424 / 576	31.7
		8	50.50	17.64 / -98.90 / 126.30	505 / 495	32.6
		9	43.50	3.34 / -104.50 / 125.10	435 / 565	31.7
		10	49.90	16.81 / -104.30 / 126.10	499 / 501	31.1
	500,000	1	45.50	7.83 / -107.80 / 126.60	455 / 545	67.3
		2	46.30	8.97 / -100.00 / 127.50	463 / 537	61.6
		3	39.10	-5.74 / -103.50 / 126.00	391 / 609	66.9
		4	50.70	18.73 / -96.50 / 125.80	507 / 493	69.6
		5	49.60	16.01 / -102.30 / 125.60	496 / 504	72.1
		6	56.90	31.23 / -99.80 / 127.20	569 / 431	62.5
		7	53.60	24.35 / -105.00 / 125.40	536 / 464	63.6
		8	40.10	-3.09 / -102.10 / 126.20	401 / 599	63.4
		9	43.40	2.60 / -105.00 / 128.80	434 / 566	63.2
		10	46.00	8.85 / -101.70 / 127.60	460 / 540	63.7
	1,000,000	1	54.50	26.49 / -105.60 / 126.60	545 / 455	123.0
		2	55.30	28.54 / -95.60 / 126.50	553 / 447	120.2
		3	56.50	30.72 / -95.20 / 126.20	565 / 435	120.8
		4	51.70	21.45 / -106.60 / 127.20	517 / 483	122.2
		5	54.30	26.34 / -102.90 / 126.60	543 / 457	122.2
		6	54.00	25.24 / -109.70 / 127.30	540 / 460	123.4
		7	57.10	32.11 / -100.60 / 127.80	571 / 429	124.5
		8	53.30	24.46 / -96.80 / 125.90	533 / 467	125.0
		9	52.40	22.37 / -99.00 / 126.60	524 / 476	125.3
		10	55.90	29.27 / -97.00 / 126.70	559 / 441	132.5

Table 6: Mixed Dense agent results.

Agent	T	Run	WR (%)	Avg/Min/Max Rewards	W/L	TT (min)
Numeric Only Agent (Sparse Reward Function	125,000	1	38.00	-24.00 / -100.00 / 100.00	380 / 620	12.4
		2	38.20	-23.60 / -100.00 / 100.00	382 / 618	12.2
		3	44.50	-11.00 / -100.00 / 100.00	445 / 555	12.5
		4	40.20	-19.60 / -100.00 / 100.00	402 / 598	12.3
		5	36.30	-27.40 / -100.00 / 100.00	363 / 637	11.0
		6	31.30	-37.40 / -100.00 / 100.00	313 / 687	12.4
		7	39.90	-20.20 / -100.00 / 100.00	399 / 601	12.0
		8	23.20	-53.60 / -100.00 / 100.00	232 / 768	12.2
		9	38.50	-23.00 / -100.00 / 100.00	385 / 615	12.3
		10	38.50	-23.00 / -100.00 / 100.00	385 / 615	12.8
	250,000	1	41.10	-17.80 / -100.00 / 100.00	411 / 589	23.5
		2	32.80	-34.40 / -100.00 / 100.00	328 / 672	23.8
		3	42.50	-15.00 / -100.00 / 100.00	425 / 575	24.9
		4	47.10	-5.80 / -100.00 / 100.00	471 / 529	22.8
		5	40.20	-19.60 / -100.00 / 100.00	402 / 598	22.7
		6	37.60	-24.80 / -100.00 / 100.00	376 / 624	23.6
		7	35.70	-28.60 / -100.00 / 100.00	357 / 643	23.1
		8	50.50	1.00 / -100.00 / 100.00	505 / 495	23.5
		9	44.50	-11.00 / -100.00 / 100.00	445 / 555	22.7
		10	42.80	-14.40 / -100.00 / 100.00	428 / 572	22.9
	500,000	1	47.00	-6.00 / -100.00 / 100.00	470 / 530	47.2
		2	42.10	-15.80 / -100.00 / 100.00	421 / 579	54.2
		3	47.00	-6.00 / -100.00 / 100.00	470 / 530	52.5
		4	43.70	-12.60 / -100.00 / 100.00	437 / 563	46.9
		5	38.70	-22.60 / -100.00 / 100.00	387 / 613	47.7
		6	50.70	1.40 / -100.00 / 100.00	507 / 493	45.4
		7	36.40	-27.20 / -100.00 / 100.00	364 / 636	44.5
		8	45.10	-9.80 / -100.00 / 100.00	451 / 549	44.5
		9	47.40	-5.20 / -100.00 / 100.00	474 / 526	45.5
		10	43.60	-12.80 / -100.00 / 100.00	436 / 564	47.7
	1,000,000	1	48.00	-4.00 / -100.00 / 100.00	480 / 520	88.2
		2	46.00	-8.00 / -100.00 / 100.00	460 / 540	86.2
		3	46.00	-8.00 / -100.00 / 100.00	460 / 540	85.6
		4	46.30	-7.40 / -100.00 / 100.00	463 / 537	85.4
		5	40.00	-20.00 / -100.00 / 100.00	400 / 600	86.3
		6	45.70	-8.60 / -100.00 / 100.00	457 / 543	88.6
		7	48.90	-2.20 / -100.00 / 100.00	489 / 511	87.3
		8	53.00	6.00 / -100.00 / 100.00	530 / 470	88.4
		9	54.30	8.60 / -100.00 / 100.00	543 / 457	88.5
		10	41.90	-16.20 / -100.00 / 100.00	419 / 581	88.5

Table 7: Numeric Sparse agent results.

Agent	T	Run	WR (%)	Avg/Min/Max Rewards	W/L	TT (min)
Numeric Only Agent (Dense Reward Function)	125,000	1	21.70	-42.85 / -103.10 / 126.20	217 / 783	12.9
		2	38.00	-7.42 / -104.10 / 126.40	380 / 620	13.7
		3	46.80	10.19 / -103.90 / 126.60	468 / 532	13.3
		4	20.90	-44.08 / -104.70 / 123.20	209 / 791	13.0
		5	42.40	1.55 / -99.00 / 126.80	424 / 576	12.9
		6	34.70	-14.10 / -105.60 / 126.20	347 / 653	12.7
		7	33.40	-17.30 / -105.00 / 126.60	334 / 666	12.6
		8	38.40	-6.47 / -97.30 / 128.00	384 / 616	13.6
		9	29.80	-24.79 / -106.30 / 127.40	298 / 702	13.2
		10	25.50	-34.33 / -104.40 / 124.70	255 / 745	13.8
	250,000	1	45.00	7.79 / -97.40 / 127.90	450 / 550	25.8
		2	40.50	-2.20 / -103.90 / 125.90	405 / 595	27.1
		3	46.10	9.40 / -94.70 / 126.30	461 / 539	26.9
		4	42.70	2.33 / -97.40 / 126.10	427 / 573	25.0
		5	42.00	0.01 / -101.20 / 126.50	420 / 580	27.4
		6	47.40	11.84 / -100.70 / 126.10	474 / 526	24.5
		7	42.70	2.08 / -109.00 / 127.90	427 / 573	26.9
		8	52.40	21.86 / -97.70 / 126.60	524 / 476	24.9
		9	33.70	-16.43 / -99.90 / 125.30	337 / 663	27.7
		10	43.70	4.02 / -115.30 / 127.90	437 / 563	25.5
	500,000	1	51.70	20.40 / -102.20 / 126.60	517 / 483	57.7
		2	42.10	0.18 / -102.00 / 125.10	421 / 579	53.8
		3	51.10	19.59 / -105.00 / 126.50	511 / 489	51.2
		4	39.40	-5.14 / -108.50 / 125.50	394 / 606	57.2
		5	45.40	7.64 / -101.90 / 128.20	454 / 546	57.5
		6	46.50	10.05 / -97.20 / 126.60	465 / 535	48.3
		7	40.00	1050.79 / 97.90 / 4261.10	400 / 600	52.3
		8	31.50	1078.23 / 146.50 / 4321.80	315 / 685	49.6
		9	40.60	-2.50 / -102.10 / 127.00	406 / 594	52.2
		10	47.70	12.65 / -103.20 / 127.70	477 / 523	50.5
	1,000,000	1	46.90	10.88 / -98.10 / 125.70	469 / 531	92.4
		2	41.20	-1.64 / -101.90 / 125.70	412 / 588	96.1
		3	39.80	-4.64 / -108.50 / 124.90	398 / 602	93.7
		4	51.10	19.38 / -103.70 / 127.30	511 / 489	94.4
		5	57.30	32.24 / -97.40 / 126.50	573 / 427	93.9
		6	49.70	16.79 / -96.10 / 128.30	497 / 503	96.3
		7	47.90	12.85 / -102.20 / 126.60	479 / 521	93.5
		8	54.20	25.88 / -105.20 / 126.50	542 / 458	93.1
		9	51.60	20.36 / -115.00 / 126.80	516 / 484	93.6
		10	49.80	16.13 / -109.00 / 125.80	498 / 502	93.5

Table 8: Numeric Dense agent results.

Agent	T	Run	WR (%)	Avg/Min/Max Rewards	W/L	TT (min)
Board Only Agent (Sparse Reward Function)	125,000	1	39.60	-20.80 / -100.00 / 100.00	396 / 604	14.4
		2	36.70	-26.60 / -100.00 / 100.00	367 / 633	15.6
		3	36.40	-27.20 / -100.00 / 100.00	364 / 636	16.2
		4	42.90	-14.20 / -100.00 / 100.00	429 / 571	13.8
		5	51.70	3.40 / -100.00 / 100.00	517 / 483	13.9
		6	45.30	-9.40 / -100.00 / 100.00	453 / 547	15.4
		7	36.90	-26.20 / -100.00 / 100.00	369 / 631	15.2
		8	37.70	-24.60 / -100.00 / 100.00	377 / 623	15.4
		9	41.30	-17.40 / -100.00 / 100.00	413 / 587	14.8
		10	42.00	-16.00 / -100.00 / 100.00	420 / 580	14.9
	250,000	1	36.90	-26.20 / -100.00 / 100.00	369 / 631	30.5
		2	35.60	-28.80 / -100.00 / 100.00	356 / 644	32.8
		3	41.70	-16.60 / -100.00 / 100.00	417 / 583	30.7
		4	38.20	-23.60 / -100.00 / 100.00	382 / 618	30.3
		5	35.20	-29.60 / -100.00 / 100.00	352 / 648	30.0
		6	37.00	-26.00 / -100.00 / 100.00	370 / 630	28.1
		7	52.40	4.80 / -100.00 / 100.00	524 / 476	27.4
		8	42.00	-16.00 / -100.00 / 100.00	420 / 580	29.8
		9	40.20	-19.60 / -100.00 / 100.00	402 / 598	30.5
		10	40.50	-19.00 / -100.00 / 100.00	405 / 595	30.4
	500,000	1	46.70	-6.60 / -100.00 / 100.00	467 / 533	60.9
		2	47.60	-4.80 / -100.00 / 100.00	476 / 524	59.4
		3	43.10	-13.80 / -100.00 / 100.00	431 / 569	64.0
		4	46.40	-7.20 / -100.00 / 100.00	464 / 536	68.1
		5	51.10	2.20 / -100.00 / 100.00	511 / 489	64.1
		6	46.80	-6.40 / -100.00 / 100.00	468 / 532	58.1
		7	44.30	-11.40 / -100.00 / 100.00	443 / 557	59.1
		8	44.80	-10.40 / -100.00 / 100.00	448 / 552	55.5
		9	53.80	7.60 / -100.00 / 100.00	538 / 462	59.7
		10	51.20	2.40 / -100.00 / 100.00	512 / 488	56.7
	1,000,000	1	51.30	2.60 / -100.00 / 100.00	513 / 487	125.3
		2	50.50	1.00 / -100.00 / 100.00	505 / 495	109.3
		3	57.00	14.00 / -100.00 / 100.00	570 / 430	107.7
		4	53.10	6.20 / -100.00 / 100.00	531 / 469	108.0
		5	54.30	8.60 / -100.00 / 100.00	543 / 457	108.9
		6	55.20	10.40 / -100.00 / 100.00	552 / 448	122.5
		7	50.30	0.60 / -100.00 / 100.00	503 / 497	114.9
		8	49.70	-0.60 / -100.00 / 100.00	497 / 503	109.2
		9	52.60	5.20 / -100.00 / 100.00	526 / 474	109.7
		10	54.90	9.80 / -100.00 / 100.00	549 / 451	109.6

Table 9: Board Sparse agent results.

Agent	T	Run	WR (%)	Avg/Min/Max Rewards	W/L	TT (min)
Board Only Agent (Dense Reward Function)	125,000	1	43.40	3.69 / -96.30 / 126.40	434 / 566	15.8
		2	36.80	-9.87 / -106.90 / 126.80	368 / 632	16.1
		3	41.10	-1.38 / -105.80 / 124.90	411 / 589	16.5
		4	29.60	-25.31 / -99.50 / 125.40	296 / 704	16.7
		5	35.90	-12.52 / -119.70 / 125.60	359 / 641	15.8
		6	38.10	-8.08 / -109.80 / 125.50	381 / 619	16.0
		7	35.70	-11.99 / -100.20 / 125.70	357 / 643	15.9
		8	37.50	-9.18 / -105.30 / 126.20	375 / 625	16.0
		9	40.20	-2.80 / -103.50 / 128.20	402 / 598	15.4
		10	44.80	5.94 / -130.70 / 126.20	448 / 552	15.4
	250,000	1	45.90	8.20 / -97.40 / 125.70	459 / 541	30.4
		2	38.60	-6.66 / -107.00 / 125.40	386 / 614	29.7
		3	43.10	2.53 / -107.20 / 126.30	431 / 569	33.0
		4	39.90	-4.19 / -106.00 / 127.90	399 / 601	32.8
		5	47.30	11.52 / -100.70 / 126.60	473 / 527	32.6
		6	43.20	2.71 / -109.20 / 125.60	432 / 568	32.8
		7	36.60	-10.94 / -104.30 / 126.90	366 / 634	30.3
		8	37.40	-9.29 / -102.60 / 126.00	374 / 626	32.0
		9	41.40	-0.26 / -103.70 / 127.40	414 / 586	31.7
		10	39.00	-6.07 / -103.10 / 129.40	390 / 610	31.6
	500,000	1	47.90	12.12 / -111.00 / 126.00	479 / 521	57.2
		2	46.80	10.27 / -105.80 / 126.50	468 / 532	56.3
		3	46.90	10.54 / -100.10 / 125.40	469 / 531	56.6
		4	48.60	14.23 / -96.80 / 124.40	486 / 514	57.5
		5	45.50	7.57 / -105.60 / 128.10	455 / 545	57.3
		6	52.10	21.38 / -103.40 / 127.90	521 / 479	60.9
		7	51.70	20.73 / -101.20 / 126.70	517 / 483	63.7
		8	47.40	11.41 / -102.00 / 125.40	474 / 526	61.4
		9	48.90	14.86 / -111.50 / 125.70	489 / 511	62.4
		10	53.80	25.34 / -98.90 / 126.40	538 / 462	61.7
	1,000,000	1	56.60	31.10 / -107.40 / 127.60	566 / 434	113.8
		2	51.80	20.62 / -99.90 / 127.70	518 / 482	114.1
		3	56.00	29.52 / -105.60 / 127.00	560 / 440	114.9
		4	55.30	28.50 / -94.50 / 127.70	553 / 447	114.0
		5	56.00	29.55 / -100.70 / 126.70	560 / 440	123.6
		6	45.80	9.05 / -103.70 / 125.40	458 / 542	131.1
		7	57.70	33.01 / -111.30 / 126.50	577 / 423	118.4
		8	47.30	11.32 / -116.10 / 126.50	473 / 527	115.3
		9	51.20	19.40 / -100.20 / 126.80	512 / 488	113.9
		10	46.40	10.26 / -109.80 / 125.30	464 / 536	114.5

Table 10: Board Dense agent results.

References

[1] Ahmad Ahmad, Mehdi Kermanshah, Kevin Leahy, Zachary Serlin, Ho Chit Siu, Makai Mann, Cristian-Ioan Vasile, Roberto Tron, and Calin Belta. Accelerating proximal policy optimization learning using task prediction for solving games with delayed rewards. *arXiv preprint arXiv:2411.17861*, 2024.

[2] Bryan Collazo and Contributors. Catanatron. `https://github.com/bcollazo/catanatron`, 2021.

[3] Heriberto Cuayáhuitl, Simon Keizer, and Oliver Lemon. Strategic dialogue management via deep reinforcement learning. *arXiv preprint arXiv:1511.08099*, 2015.

[4] Brahim Driss and Tristan Cazenave. Deep catan. In *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*, pages 503–513. Springer, 2022.

[5] Quentin Gendre and Tomoyuki Kaneko. Playing catan with cross-dimensional neural network. In *Neural Information Processing: 27th International Conference, ICONIP 2020, Bangkok, Thailand, November 23–27, 2020, Proceedings, Part II 27*, pages 580–592. Springer, 2020.

[6] Levente Kocsis and Csaba Szepesvári. Bandit based monte-carlo planning. In *European conference on machine learning*, pages 282–293. Springer, 2006.

[7] Jeremey Monin and Contributors. Jsettlers2 release-2.2.00. `https://github.com/jdmonin/JSettlers2/releases/tag/release-2.2.00`, 2020.

[8] Michael Pfeiffer. Reinforcement learning of strategies for settlers of catan. In *Proceedings of the International Conference on Computer Games: Artificial Intelligence, Design and Education*, 2004.

[9] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. *arXiv preprint arXiv:1707.06347*, 2017.

[10] David Silver, Julian Schrittwieser, Karen Simonyan, Ioannis Antonoglou, Aja Huang, Arthur Guez, Thomas Hubert, Lucas Baker, Matthew Lai, Adrian Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[11] István Szita, Guillaume Chaslot, and Pieter Spronck. Monte-carlo tree search in settlers of catan. In *Advances in computer games*, pages 21–32. Springer, 2009.

[12] Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, et al. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024.

[13] Matthew Whelan, Nitin Maddi, and Sidhardh Burre. Reinforcement learning in catan. In *37th Conference on Neural Information Processing Systems (NeurIPS 2023).*, 2023.

[14] Konstantia Xenou, Georgios Chalkiadakis, and Stergos Afantenos. Deep reinforcement learning in strategic board game environments. In *European Conference on Multi-Agent Systems*, pages 233–248. Springer, 2018.

Wits University Faculty of Science post-graduate student AI declaration

I understand that the use of generative AI tools (such as ChatGPT or similar) without explicitly declaring such use constitutes a form of plagiarism and is classified by Wits University as academic misconduct.

I declare that in the course of conducting the research towards my degree or in the preparation of this thesis/dissertation/research report (select one by marking with an X):

I **did not** make use of generative AI tools ☐

I **did** make use of generative AI tools for the following (tick all that apply):

1. Idea Generation (research problem/design, hypothesis) ☐
2. Sourcing Related Work (summarising, identifying sources) ☐
3. Methods and Experiment Design (experiment setup, model tuning) ☒
4. Data Analysis (presentation, coding, interpretation) ☐
5. Theoretical Development (theorem proving, conceptual analysis) ☐
6. Code Development (generating algorithms, writing scripts) ☒
7. Presentation (rendering graphics, formatting) ☒
8. Editing (grammar, readability) ☒
9. Writing (text generation, document structuring) ☐
10. Citation Formatting (structuring, organising) ☐

If other uses were involved, please specify below:

Generative AI tool used (list all)	**Used for?**

If generative AI tools were used as an integral part of the experimental design or in the direct execution of my research, I confirm that details of this use are clearly outlined in the relevant experimental/methodology chapters of my thesis/dissertation/research report.

Student number:

2596852

Candidate signature:

Date:

10/11/2025