A LAW OF TOTAL VARIANCE

The *law of total variance* for two continuous random variables X and Y can be derived as follows:

$$\begin{aligned}
\mathbb{V}_Y[Y] &= \int (Y - \mathbb{E}_Y[Y])^2 \, p(Y) \, dY \quad = \quad \iint (Y - \mathbb{E}_Y[Y])^2 \, p(X, Y) \, dX \, dY \\
&= \iint (Y - \mathbb{E}_Y[Y])^2 \, p(Y|X) \, p(X) \, dX \, dY \quad = \quad \mathbb{E}_X\Big[\mathbb{E}_Y\big[(Y - \mathbb{E}_Y[Y])^2 \big| X\big]\Big] \\
&= \mathbb{E}_X\Big[\mathbb{E}_Y\big[(Y - \mathbb{E}_Y[Y|X] + \mathbb{E}_Y[Y|X] - \mathbb{E}_Y[Y])^2 \big| X\big]\Big] \\
&= \mathbb{E}_X\Big[\underbrace{\mathbb{E}_Y\big[(Y - \mathbb{E}_Y[Y|X])^2 \big| X\big]}_{\mathbb{V}_Y[Y|X]}\Big] + 2\,\mathbb{E}_X\Big[\big(\underbrace{\mathbb{E}_Y[Y|X] - \mathbb{E}_Y[Y|X]}_{0}\big)\big(\mathbb{E}_Y[Y|X] - \mathbb{E}_Y[Y]\big)\Big] \\
&\quad + \underbrace{\mathbb{E}_X\Big[\big(\mathbb{E}_Y[Y|X] - \mathbb{E}_Y[Y]\big)^2\Big]}_{\mathbb{V}_X[\mathbb{E}_Y[Y|X]]} \quad = \quad \mathbb{V}_X\big[\mathbb{E}_Y[Y|X]\big] + \mathbb{E}_X\big[\mathbb{V}_Y[Y|X]\big]
\end{aligned}$$

B EXPERIMENTAL SETUP

This section contains additional information regarding the choices of epistemic uncertainty estimation in the functions used by MuZero, modifications to the reward scheme of the environments used in the evaluation, and specific hyper parameter choices and tuning.

B.1 OP2E WITH OTHER MCTS SEARCH HEURISTICS

The UCT search heuristic is not the heuristic used by MuZero. Instead, MuZero uses a variant called probabilistic upper confidence tree (PUCT) bound (see MuZero's appendices at (Schrittwieser et al., 2020)):

$$a_k = \arg\max_{a \in A} q(\hat{s}_k, a) + \pi(\hat{s}_k, a)\lambda_{\hat{s}_k, a}, \tag{17}$$

where $\lambda_{\hat{s}_k,a} = \frac{\sqrt{\sum_{a'} N(\hat{s}_k, a')}}{1+N(\hat{s}_k,a)}\Big[C_1 + \log\big(\frac{N(\hat{s}_k,a')+C_2+1}{C_2}\big)\Big]$ and where $\pi(\hat{s}_k, a)$ is the probability of taking action a at node $\hat{s}_k$ according to the policy network π. A key difference between the two heuristics is that PUCT takes into account some prior knowledge over the outcomes of the actions in terms of $\pi(\hat{s}_k, a)$, while UCT does not involve a prior policy. In addition, since the publication of MuZero other search criteria have been proposed, such as the precise solution to the regularized policy optimization problem (Grill et al., 2020) and a variation of this approach by (Danihelka et al., 2021).

To incorporate OP2E into PUCT, in our experiments we have modified PUCT similarly to the proposed modification to UCT, by adding the environmental uncertainty in the value prediction as an additional term, in the form of the average standard deviation:

$$a_{k-1} = \arg\max_{a \in A} q(\hat{s}_{k-1}, a) + \pi(\hat{s}_{k-1}, a)\lambda_{\hat{s}_{k-1}, a} + C_\sigma\sqrt{\tfrac{\sigma_\nu^2}{N(\hat{s}_{k-1},a)}}. \tag{18}$$

Incorporating OP2E into the criteria used by the more recent approaches can be done by treating the averaged standard deviation term $C_\sigma\sqrt{\frac{\sigma_\nu^2}{N(\hat{s}_{k-1},a)}}$ as directly part of the Q-value and replacing every estimate of $q(\hat{s}_{k-1}, a)$ with $q(\hat{s}_{k-1}, a) + C_\sigma\sqrt{\frac{\sigma_\nu^2}{N(\hat{s}_{k-1},a)}}$. Deciding whether these modification should be extended to the policy targets generated based on these estimates can follow the reasoning of max policy targets from section 3.3.

B.2 ESTIMATING EPISTEMIC VALUE AND REWARD UNCERTAINTY WITH STATE VISITATION COUNTING

Counting of state-action pairs' visitations can naturally be used as an epistemic uncertainty estimate. We identify three challenges to incorporating this epistemic uncertainty estimator into MuZero: 1)

MuZero is planning with *abstracted* states, while the counter is designed to work against discrete real states of the system. 2) Our method requires (at least) two independent estimates for the uncertainty: one in reward, and one in value, while state visitation counting provides a single source for uncertainty estimation. 3) Estimating uncertainty in continuous state-space environments, such as Mountain Car. In this section, we will describe the approaches used to overcome each one of those challenges.

B.2.1 ESTIMATING COUNTING UNCERTAINTY IN PLANNING

In order to estimate the uncertainty associated with *real* states during planning, we allow MuZero access to a real model of the environment. This of course violates the assumption that MuZero is able to learn entirely from interactions with the environment without access to any prior knowledge. This is only introduced in order to evaluate the soundness of our method, and the real model is only used to estimate the uncertainty with planned actions. The second uncertainty estimation method we use, ensembles, does not violate any such assumptions. The epistemic uncertainty in reward-prediction is estimated as follows:

$$u_{r_k} = \beta \frac{1}{n_{s_k} + \epsilon}$$

Where n_{s_k} denotes the count of visitations to *real* state s_k, the state associated with action trajectory $a_{0:k}$ and observation o in a deterministic environment. β is some constant used to scale the uncertainty, and ϵ is a constant used to guarantee numerical stability when $n_{s_k} = 0$. During the MCTS planning phase, in each expansion step, the agent uses the real model to predict the transition associated with the chosen action from the chosen state, and uses this prediction to estimate the reward uncertainty.

B.2.2 CREATING SEPARATE ESTIMATES FOR REWARD UNCERTAINTY AND VALUE UNCERTAINTY

In order to use state visitation counting as an independent uncertainty estimate for both the value of a *leaf* planning-tree-node k as well as the reward predicted for a transition k, we employ two ideas: 1) we assume that the reward uncertainty of future transitions $u_{r_{k+i}}, \forall i > 0$, can be crudely estimated as equal the local reward uncertainty u_{r_k} without completely debilitating the value uncertainty estimation's reliability. 2) We utilize a similar approach to MC simulations to arrive at an approximation of the value uncertainty u_{v_k} that is expected to be better than that provided by 1).

We combine both ideas to arrive at a final computation for the value-uncertainty estimate for leaf-node k. First, the agent plans from real state s_k forward, using the real model, with some action-selection policy π_σ one trajectory h steps into the future. At each step, the agent evaluates the uncertainty of each transition with the state-counter. Second, upon arriving at step $k + h$, the agent uses the geometric-series formula to approximate the uncertainty of following the same policy to infinity, with the approximation that all uncertainties from state s_{k+h} into the future, following policy π_σ, are constant and equal $u_{r_{k+h}}$:

$$u_{v_k} \approx \sum_{i=0}^{h-1} \gamma^{2i} u_{r_{k+i}} + \gamma^{2h} u_{v_{k+h}} \approx \sum_{i=0}^{h-1} \gamma^{2i} u_{r_{k+i}} + \sum_{i=h}^{\infty} \gamma^{2i} u_{r_{k+h}} = \sum_{i=0}^{h-1} \gamma^{2i} u_{r_{k+i}} + \frac{\gamma^{2h}}{1-\gamma^2} u_{r_{k+h}}$$

The first step approximates u_{v_k} as the discounted sum of reward-uncertainties along the trajectory $a_{k:k+h-1}$, and then with an as yet unknown discounted end-of-trajectory value-uncertainty estimate $u_{v_{k+h}}$. The second step approximates the end-of-trajectory value-uncertainty estimate $u_{v_{k+h}}$ as the sum of a geometric series with the constant reward uncertainty attained at the end of the trajectory, $u_{r_{k+h}}$. The policy π_σ we chose to follow is "repeat action a_{k-1}". For example, if the action leading to planning node k was "accelerate to the right", π_σ chooses "accelerate to the right" for all actions along the trajectory $a_{k:k+h}$. This enables u_{v_k} to propagate information from future decisions that (may) be taken by the algorithm, which should provide rather-independent uncertainty estimation from the local reward uncertainty estimates u_{r_k}.

B.2.3 EXTENDING STATE- VISITATIONS COUNTING TO CONTINUOUS STATE-SPACE ENVIRONMENTS

The state space of the Mountain Car environment is continuous. In order to employ visitations-counting in a continuous position-velocity state-space environment, we use discretization of the state-space to a 50 by 50 grid of possible position-velocity combinations, which is made possible because the ranges of both the velocity as well as the positions are finite.

B.3 ESTIMATING EPISTEMIC VALUE AND REWARD UNCERTAINTY WITH AN ENSEMBLE

Estimating the same quantities with the ensemble is done in a much more straight forward manner. The variance in the predictions of the different ensemble members is computed, and is used as the direct measure of the uncertainty in each function - reward and value. MuZero predicts the rewards and values using a categorical representation rather than a simple regression to scalar, however. The categorical representation can represent numbers in the range $(-support, support)$, for some hyperparameter $support$ that specifies the size of the output layer of the network, which is $support \times 2 + 1$. The vector-output of the network is passed through a SoftMax function. The weights of the categorical distribution are multiplied by the values represented by the $(-support, support)$. Finally, the entries are summed to produce the final prediction. This architecture introduces an additional challenge to the variance computation - rather than computing the variance over a set of scalars, now one is presented with a set of distributions over which to compute the variance.

As an additional effect of this architecture, we have observed that in under-trained areas of the input space, the networks have tendency to converge to outputs close to 0. We explain this with the claim that for inputs that for the network are arbitrary, the network is likely to produce outputs that are arbitrary. Under the assumption that each entry in the soft-max output vector is somewhat independent from any other for arbitrary inputs, we expect arbitrary outputs for a categorical representation to, on average, not be concentrated in one extreme side of the representation. Further, they are likely to be about as concentrated on one "side" of the vector as on the other, which will reduce the total absolute value of the scalar represented by the vector, pushing it closer to 0. This suspected phenomenon has two noteworthy effects: 1) the variance in the scalar-representation of the ensemble prediction reduces to zero in under-trained areas of the input space, which is exactly adverse to the behavior we require. 2) This results in an implicit, if unreliable, optimistic or pessimistic initialization of rewards and value predictions (depending on the reward scheme of the environment). Specifically, in environments where the true values are all negative and represented by the extreme state of the soft max vector $-support$, this may induce an inherent optimistic-initialization effect to the agent's value and reward estimates, implicitly encouraging the agent to explore the unknown, because unknown state are associated with value and reward predictions that are more likely to be close to zero. This effect is expected to be even stronger when an ensemble is used, because the averaging effect goes stronger with the size of the ensemble.

In order to mitigate these unintended effects, we have taken two steps. First, we have modified the reward schemes of the environments we have tested against to only produce positive rewards, and only in the goal state, to disable the effect of any unintended optimistic initialization, which may conflict with the method of this work and give the vanilla version an unintended advantage. Second, rather than compute the straight-forward variance in an ensemble as the variance over the translated-to-scalar predictions, we compute the variance as the variance between the entries of the different categorical representation-entries, entry by entry, and sum them as the final variance measure:

$$\mathbb{V}[y] \approx \sum_{i=0}^{2 \times support+1} \mathbb{V}[y_i]$$

for $y \in [0,1]^{2support+1}$ denoting the categorical-vector output of the NN. An additional variance computation that was considered but had not shown advantage in our preliminary experiments was computing the average categorical distribution of the ensemble, and then taking the average Jensen-Shannon distance (Nielsen, 2019) between each ensemble-member's categorical distribution, and the mean categorical distribution. While both approaches cannot be expected to be in the correct scale of the real variance of the scalar reward or value predictions, our experiments show that the entry-by-entry variance, at least, is sufficient to achieve both directed as well as deep exploration

(figure 1). In addition, the tuning of the C_σ parameter can alleviate errors in the scale of the variance with respect to the actual rewards and values.

B.4 REWARD SCHEMES

The standard reward scheme of the Mountain Car environment produces a reward of -1 at each time step. The only escape available to the agent is the goal state, which is terminal. The optimal policy induced by this reward scheme is "arrive at the goal in the smallest number of timesteps possible". As mentioned in section B.3, reward schemes that induce negative values are likely to cause unexpected and averse effects for the purpose of evaluating the modifications to the agent. For this reason, we use an additional non-Markovian reward scheme.

In the non-Markovian reward scheme, the agent receives a reward of 0 at each transition accept the transitions into the goal, for which it receives a non-constant reward equal $r_{goal} = T_{timeout} - T_{elapsed}$. $T_{timeout}$ denotes the maximum number of timesteps the environment allows for, before sending a timeout signal and terminating the agent. $T_{elapsed}$ is the number of timesteps elapsed in the environment, up until the agent transitioned into the terminal goal state. While this non-Markovian reward induces a non-Markovian environment, the optimal policy remains the same, and so does the learning process of the agent.

In the final experiments conducted, the effects of the original -1s reward scheme of Mountain Car that were observed in earlier experiments were not observed, and thus we present experiments against the original reward scheme for the main experiments. The ablations were experimented against the non-Markovian reward scheme, to reduce any additional influencing factors on the ablations.

B.5 HYPERPARAMETER OPTIMIZATION

The purpose of the main evaluation presented in this work is to illustrate the effect of planning to explore compared to the vanilla version of the algorithm. Further, as the the two versions of the algorithm are not too different from each other, we expect that the majority of hyperparamer optimization will effect all versions similarly. For this reason, no dedicated tuning of hyperparamers was conducted as part of the experiments conducted in this work. The hyperparameters used were chosen based on existing implementations for other environments in the original code base (Duvaud, 2021). The network architecture used for mountaincar was based on another implementation of MuZero (de Vries et al., 2021) that was evaluated against the Mountain Car environment.

Two hyperparameters are somewhat exempt from this statement, however. The exploration coefficient c_σ introduced with our proposed methodology for planning for exploration was tuned for each task and for each variant. The temperature parameter T was not tuned explicitly, but a different temperature parameter was used between the different variants. Motivation and description of the reasoning behind the optimization process and the process itself are provided in the following sections.

B.5.1 TEMPERATURES

The temperature parameter T is used by MuZero for action sampling in the environment as follows:

$$a_t \sim p(\cdot), \quad p(a_i) = \frac{N^{\frac{1}{T}}(n_0, a_i)}{\sum_{a' \in A} N^{\frac{1}{T}}(n_0, a')} \tag{19}$$

$p(a_i)$ denotes the probability of sampling action a_i according to the temperature and the visitation counts. When the temperature $T \to 0$, the probability distribution collapses to greedy action selection according the the maximum number of visitations. When the temperature $T \to \infty$, the distribution becomes uniform. The temperature induces exploration in the environment through random action selection weighted towards "better" actions from an exploitatory perspective, according to the estimates of the tree. As the modifications proposed in this work are meant to provide much more informed exploration, the temperatures used were lower than the original configuration used by other implementations. The original range was $1 \to 0.25$, and the modified range was $0.25 \to 0.1$. In the experiments, vanilla MuZero was evaluated both with the lower temperatures used by the

planning with uncertainty variants (to not introduce any unexpected advantage from the lower temperatures), as well as with higher temperatures pre-configured for other environments in the code base we built upon, to give MuZero a chance with the weighted random action selection used by MuZero for exploration. There was no significant difference between the performance of the different temperatures with vanilla MuZero. The experiments presented are with higher temperatures. Exact hyperparameters are specified in appendix C.3.

B.5.2 TUNING THE EXPLORATION COEFFICIENT C_σ

The exploration coefficient C_σ (see section 3.2) was tuned independently for each uncertainty mechanism used, and again per environment. The tuning aimed to achieve preference by the UCB of unvisited states over everything else, and the goal-reward over anything except un-visited states. The tuning was stopped upon observation that deep exploration was achieved successfully in most seeds. Once tuning was stopped, the experiments were initiated for the chosen number of seeds (10 for the main experiments and 5 for the ablations). The range of C_σ investigated for the state-visitation-counting method was between 0.1 and 100 and was done using rough and then fine grid-search. The range of C_σ investigated for the ensemble-variance method was between 10^1 and 10^8 and was search with a rough and then fine grid search.

C IMPLEMENTATION

The implementation used to evaluate the agent is accessible in {commented for review}. This implementation was built on the implementation by (Duvaud, 2021), which in turn is built on the official psuedocode released in the original MuZero paper (Schrittwieser et al., 2020). The implementation of the ensemble architecture was based on (Hansen, 2019), which is an implementation of the bootstrapped-DQN with randomize prior networks proposed in (Osband et al., 2018). In the following sections, we specify first the details of the network architecture used, and second the hyperparameters used.

C.1 NETWORK ARCHITECTURE

Two network architectures were used in this work to evaluate the *planning for exploration* methodology. These architecture are divided between agents that used ensemble-variance as an uncertainty mechanism, and the agents that didn't. Both architectures use blocks of feed-forward networks for every estimator used by the agent: 1) the representation function $g(o)$. 2) the transition dynamics $f(s, a)$. 3) the reward function $r(s)$. 4) the value function $v(s)$ and 5) the policy function $\pi(s)$.
Representation function block This feed-forward network consisted of an input layer of size 1 (the dimensionality of the observation space), a hidden layer of size 16, and an output layer of size 4.
Transition dynamics function block This feed-forward network consists of an input layer of size 7 (state-abstraction-encoding size of 4, and action space of 3), two hidden layers of size 16, and an output layer of size 4.
Reward & value function blocks These two feed-forward blocks have identical architecture, consisting of an input layer of size 4, two hidden layer of size 16 and an output layer of size $support \cdot 2 + 1$, for a categorical representation of real numbers, as discussed in section B.3. The support size used was 15, for a output-layer size of 31.
Policy function block This feed forward block used an input layer of size 4, two hidden layers of size 16, and an output layer of size 3, the size of the action space.
Ensemble architecture The architecture of networks used by the ensemble-using agents formulated the relevant blocks (reward and value blocks) as ensembles rather than individual blocks. This translates to having 5 (the ensemble size used) independent blocks of reward, and 5 of value. The prediction from the block is taken as the average of the individual blocks' predictions.

C.2 HYPERPARAMETERS CONFIGURATION

We divide the hyperparameters into 3 distinct classes: 1) *planning for exploration* target-adaptation parameters, such as whether to use n-step or 0-step targets. These parameters are described in the experimental setup, section B. 2) Network-architecture details. These parameters are described in

a dedicated appendix, C.1. 3) Additional hyperparameters, such as number of training steps, batch size, learning steps decay, etc. These hyperparameters are detailed in this section, in table 1.

	Slide	Slide, abl.	Mountain Car	Mountain Car, abl.	Comment
Num. of stacked obs.	1	1	1	1	-
Discount parameter γ	0.95	0.95	0.997	0.997	-
Planning nodes budget	30	30	200	200	1
Root dirichlet α	0.25	0.25	0.25	0.25	-
Root exploration fraction	0.25	0.25	0.25	0.25	-
UCB's pb-c-base	19652	19652	19652	19652	2
UCB's pb-c-init	1.25	1.25	1.25	1.25	3
Training steps	70000	45000	120000	100000	4
Traning ratio	2.25	2.25	1.75	1.75	
Batch size	128	128	128	128	-
Value-loss weight	1	1	1	1	5
Training hardware	distributed CPUs	distributed CPUs	distributed CPUs	distributed CPUs	-
Learning rate λ	0.02	0.02	0.02	0.02	6
Rate of λ decay	0.9	0.9	0.9	0.9	
Decay steps c_{steps}	500	500	2000	2000	
Replay buffer size	500	500	1000	1000	-
Unroll steps in loss	10	10	10	10	-
n-step target's n	50	50	50	50	-
Prioritized replay	0.5	0.5	0.5	0.5	7
Reanalyze	True	True	True	True	8
Ensemble size	5	-	5	-	-

Table 1: Hyperparemeters used in the results presented in section 4

We provide a list of comments for additional details regarding some of the hyper-parameters:

1. The number of nodes in each MCTS planning tree. Preliminary results in Mountain Car with planning budget of 50, showed the same behavior as in the results presented in section 4 but with lower stability.
2. A UCT parameter used by MuZero's MCTS variant. For more details, see (Schrittwieser et al., 2020).
3. A UCT parameter used by MuZero's MCTS variant. For more details, see (Schrittwieser et al., 2020).
4. The implementation maintains a ratio of *Training ratio* between the training steps and the environmental steps. A ratio of x represents x training steps for each environment steps. Additional results attained but not presented in the work experimented with up to 300000 steps. The behavior observed was the same as the one showed in the results presented. Observing that the large number of training steps does not appear to be necessary to demonstrate the capacity of our method, and due to compute and time resource limitations, the experiments with the final hyperparameters, presented in section 4, were conducted with a smaller number of timesteps.
5. MuZero enables scaling of the different losses in the loss computation independently.
6. The learning rate decay is computed as follows: $\lambda\rho^{n/c_{steps}}$, for n the current training step count.
7. The sampling-priorities of trajectories in the replay buffer are computed as the absolute value of the difference between the value and the value target, to the power of this hyperparameter.
8. The rudimentary implementation of MuZero-Reanlyze used in this implementation replaces old value estimates from planning trees with new value estimates from the value function directly, rather than from newly computed planning trees.

C.3 TEMPERATURES

Two ranges of temperatures were used in each environment. The regular temperatures were used to evaluate the vanilla agent, while the low temperatures were used to evaluate the exploratory agent. The exact temperature values are specified in table 2. The regular temperatures' values switched at

0.3, and 0.5 of the total training step budget. The low temperatures' values switched at 0.3, 0.5 and 0.75 of the total training step budget.

	Slide	Mountain Car
Regular	1.0, 0.5, 0.25	1, 0.5, 0.25
Low	0.75, 0.25, 0.175, 0.02	0.5, 0.25, 0.175, 0.1

Table 2: Temperature ranges used in this work