REINFORCEMENT LEARNING WITH UNSUPERVISED AUXILIARY TASKS

Max Jaderberg*, **Volodymyr Mnih***, **Wojciech Marian Czarnecki***
Tom Schaul, Joel Z Leibo, David Silver & Koray Kavukcuoglu
DeepMind
London, UK
{jaderberg,vmnih,lejlot,schaul,jzl,davidsilver,korayk}@google.com

ABSTRACT

Deep reinforcement learning agents have achieved state-of-the-art results by directly maximising cumulative reward. However, environments contain a much wider variety of possible training signals. In this paper, we introduce an agent that also maximises many other pseudo-reward functions simultaneously by reinforcement learning. All of these tasks share a common representation that, like unsupervised learning, continues to develop in the absence of extrinsic rewards. We also introduce a novel mechanism for focusing this representation upon extrinsic rewards, so that learning can rapidly adapt to the most relevant aspects of the actual task. Our agent significantly outperforms the previous state-of-the-art on Atari, averaging 880% expert human performance, and a challenging suite of first-person, three-dimensional *Labyrinth* tasks leading to a mean speedup in learning of 10× and averaging 87% expert human performance on Labyrinth.

Natural and artificial agents live in a stream of sensorimotor data. At each time step t, the agent receives observations o_t and executes actions a_t. These actions influence the future course of the sensorimotor stream. In this paper we develop agents that learn to predict and control this stream, by solving a host of reinforcement learning problems, each focusing on a distinct feature of the sensorimotor stream. Our hypothesis is that an agent that can flexibly control its future experiences will also be able to achieve any goal with which it is presented, such as maximising its future rewards.

The classic reinforcement learning paradigm focuses on the maximisation of extrinsic reward. However, in many interesting domains, extrinsic rewards are only rarely observed. This raises questions of what and how to learn in their absence. Even if extrinsic rewards are frequent, the sensorimotor stream contains an abundance of other possible learning targets. Traditionally, unsupervised learning attempts to reconstruct these targets, such as the pixels in the current or subsequent frame. It is typically used to accelerate the acquisition of a useful representation. In contrast, our learning objective is to predict and control features of the sensorimotor stream, by treating them as pseudo-rewards for reinforcement learning. Intuitively, this set of tasks is more closely matched with the agent's long-term goals, potentially leading to more useful representations.

Consider a baby that learns to maximise the cumulative amount of red that it observes. To correctly predict the optimal value, the baby must understand how to increase "redness" by various means, including manipulation (bringing a red object closer to the eyes); locomotion (moving in front of a red object); and communication (crying until the parents bring a red object). These behaviours are likely to recur for many other goals that the baby may subsequently encounter. No understanding of these behaviours is required to simply reconstruct the redness of current or subsequent images.

Our architecture uses reinforcement learning to approximate both the optimal policy and optimal value function for many different pseudo-rewards. It also makes other auxiliary predictions that serve to focus the agent on important aspects of the task. These include the long-term goal of predicting cumulative extrinsic reward as well as short-term predictions of extrinsic reward. To learn more efficiently, our agents use an experience replay mechanism to provide additional updates

*Joint first authors. Ordered alphabetically by first name.

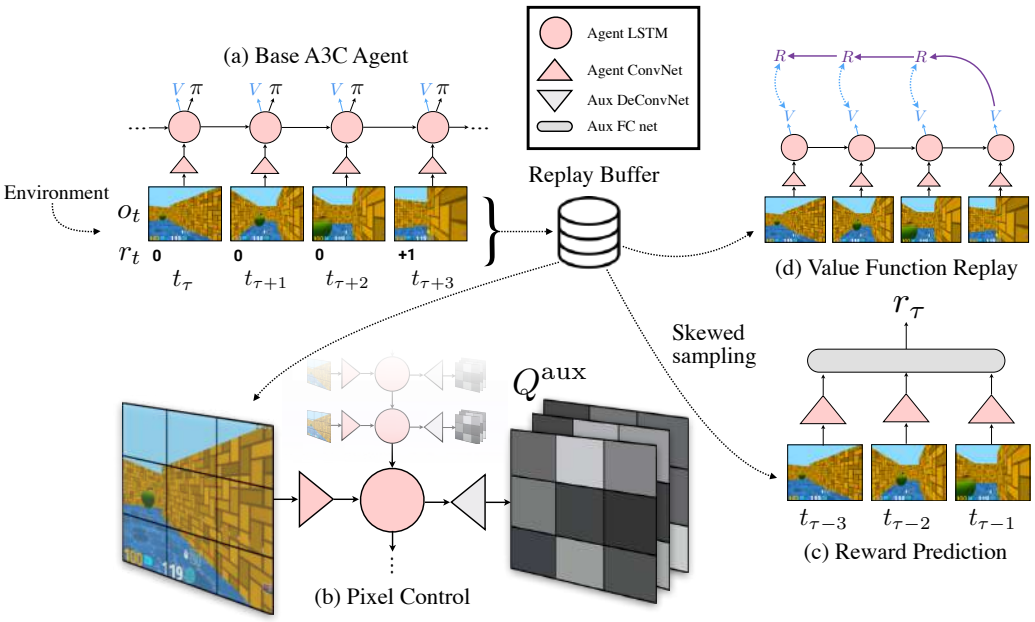


Figure 1: Overview of the *UNREAL* agent. (a) The base agent is a CNN-LSTM agent trained on-policy with the A3C loss (Mnih et al., 2016). Observations, rewards, and actions are stored in a small replay buffer which encapsulates a short history of agent experience. This experience is used by auxiliary learning tasks. (b) Pixel Control – auxiliary policies Q^{aux} are trained to maximise change in pixel intensity of different regions of the input. The agent CNN and LSTM are used for this task along with an auxiliary deconvolution network. This auxiliary control task requires the agent to learn how to control the environment. (c) Reward Prediction – given three recent frames, the network must predict the reward that will be obtained in the next unobserved timestep. This task network uses instances of the agent CNN, and is trained on reward biased sequences to remove the perceptual sparsity of rewards. (d) Value Function Replay – further training of the value function using the agent network is performed to promote faster value iteration. Further visualisation of the agent can be found in `https://youtu.be/Uz-zGYrYEjA`

to the critics. Just as animals dream about positively or negatively rewarding events more frequently (Schacter et al., 2012), our agents preferentially replay sequences containing rewarding events.

Importantly, both the auxiliary control and auxiliary prediction tasks share the convolutional neural network and LSTM that the base agent uses to act. By using this jointly learned representation, the base agent learns to optimise extrinsic reward much faster and, in many cases, achieves better policies at the end of training.

This paper brings together the state-of-the-art Asynchronous Advantage Actor-Critic (A3C) framework (Mnih et al., 2016), outlined in Section 2, with auxiliary control tasks and auxiliary reward tasks, defined in sections Section 3.1 and Section 3.2 respectively. These auxiliary tasks do not require any extra supervision or signals from the environment than the vanilla A3C agent. The result is our UNsupervised REinforcement and Auxiliary Learning (*UNREAL*) agent (Section 3.4)

In Section 4 we apply our *UNREAL* agent to a challenging set of 3D-vision based domains known as the *Labyrinth* (Mnih et al., 2016), learning solely from the raw RGB pixels of a first-person view. Our agent significantly outperforms the baseline agent using vanilla A3C, even when the baseline was augmented with an unsupervised reconstruction loss, in terms of speed of learning, robustness to hyperparameters, and final performance. The result is an agent which on average achieves 87% of expert human-normalised score, compared to 54% with A3C, and on average 10× faster than A3C. Our *UNREAL* agent also significantly outperforms the previous state-of-the-art in the Atari domain.

1 Related Work

A variety of reinforcement learning architectures have focused on learning temporal abstractions, such as options (Sutton et al., 1999b), with policies that may maximise pseudo-rewards (Konidaris & Barreto, 2009; Silver & Ciosek, 2012). The emphasis here has typically been on the development of temporal abstractions that facilitate high-level learning and planning. In contrast, our agents do not make any direct use of the pseudo-reward maximising policies that they learn (although this is

an interesting direction for future research). Instead, they are used solely as auxiliary objectives for developing a more effective representation.

The Horde architecture (Sutton et al., 2011) also applied reinforcement learning to identify value functions for a multitude of distinct pseudo-rewards. However, this architecture was not used for representation learning; instead each value function was trained separately using distinct weights.

The UVFA architecture (Schaul et al., 2015a) is a factored representation of a continuous set of optimal value functions, combining features of the state with an embedding of the pseudo-reward function. Initial work on UVFAs focused primarily on architectural choices and learning rules for these continuous embeddings. A pre-trained UVFA representation was successfully transferred to novel pseudo-rewards in a simple task.

Similarly, the successor representation (Dayan, 1993; Barreto et al., 2016; Kulkarni et al., 2016) factors a continuous set of expected value functions for a fixed policy, by combining an expectation over features of the state with an embedding of the pseudo-reward function. Successor representations have been used to transfer representations from one pseudo-reward to another (Barreto et al., 2016) or to different scales of reward (Kulkarni et al., 2016).

Another, related line of work involves learning models of the environment (Schmidhuber, 2010; Xie et al., 2015; Oh et al., 2015). Although learning environment models as auxiliary tasks could improve RL agents (*e.g.* Lin & Mitchell (1992); Li et al. (2015)), this has not yet been shown to work in rich visual environments.

More recently, auxiliary predictions tasks have been studied in 3D reinforcement learning environments. Lample & Chaplot (2016) showed that predicting internal features of the emulator, such as the presence of an enemy on the screen, is beneficial. Mirowski et al. (2016) study auxiliary prediction of depth in the context of navigation.

2 BACKGROUND

We assume the standard reinforcement learning setting where an agent interacts with an environment over a number of discrete time steps. At time t the agent receives an observation o_t along with a reward r_t and produces an action a_t. The agent's state s_t is a function of its experience up until time t, $s_t = f(o_1, r_1, a_1, ..., o_t, r_t)$. The n-step return $R_{t:t+n}$ at time t is defined as the discounted sum of rewards, $R_{t:t+n} = \sum_{i=1}^{n} \gamma^i r_{t+i}$. The value function is the expected return from state s, $V^\pi(s) = \mathbb{E}\left[R_{t:\infty} | s_t = s, \pi\right]$, when actions are selected accorded to a policy $\pi(a|s)$. The action-value function $Q^\pi(s,a) = \mathbb{E}\left[R_{t:\infty} | s_t = s, a_t = a, \pi\right]$ is the expected return following action a from state s.

Value-based reinforcement learning algorithms, such as Q-learning (Watkins, 1989), or its deep learning instantiations DQN (Mnih et al., 2015) and asynchronous Q-learning (Mnih et al., 2016), approximate the action-value function $Q(s,a;\theta)$ using parameters θ, and then update parameters to minimise the mean-squared error, for example by optimising an n-step lookahead loss (Peng & Williams, 1996), $\mathcal{L}_Q = \mathbb{E}\left[\left(R_{t:t+n} + \gamma^n \max_{a'} Q(s', a'; \theta^-) - Q(s,a;\theta)\right)^2\right]$; where θ^- are previous parameters and the optimisation is with respect to θ.

Policy gradient algorithms adjust the policy to maximise the expected reward, $\mathcal{L}_\pi = -\mathbb{E}_{s\sim\pi}\left[R_{1:\infty}\right]$, using the gradient $\frac{\partial \mathbb{E}_{s\sim\pi}[R_{1:\infty}]}{\partial\theta} = \mathbb{E}\left[\frac{\partial}{\partial\theta}\log\pi(a|s)(Q^\pi(s,a) - V^\pi(s))\right]$ (Watkins, 1989; Sutton et al., 1999a); in practice the true value functions Q^π and V^π are substituted with approximations. The Asynchronous Advantage Actor-Critic (A3C) algorithm (Mnih et al., 2016) constructs an approximation to both the policy $\pi(a|s,\theta)$ and the value function $V(s,\theta)$ using parameters θ. Both policy and value are adjusted towards an n-step lookahead value, $R_{t:t+n} + \gamma^n V(s_{t+n+1}, \theta)$, using an entropy regularisation penalty, $\mathcal{L}_{\text{A3C}} \approx \mathcal{L}_{\text{VR}} + \mathcal{L}_\pi - \mathbb{E}_{s\sim\pi}\left[\alpha H(\pi(s,\cdot,\theta)\right]$, where $\mathcal{L}_{\text{VR}} = \mathbb{E}_{s\sim\pi}\left[\left(R_{t:t+n} + \gamma^n V(s_{t+n+1}, \theta^-) - V(s_t,\theta)\right)^2\right]$.

In A3C many instances of the agent interact in parallel with many instances of the environment, which both accelerates and stabilises learning. The A3C agent architecture we build on uses an LSTM to jointly approximate both policy π and value function V, given the entire history of experience as inputs (see Figure 1 (a)).

3 AUXILIARY TASKS FOR REINFORCEMENT LEARNING

In this section we incorporate *auxiliary tasks* into the reinforcement learning framework in order to promote faster training, more robust learning, and ultimately higher performance for our agents. Section 3.1 introduces the use of auxiliary control tasks, Section 3.2 describes the addition of reward focussed auxiliary tasks, and Section 3.4 describes the complete *UNREAL* agent combining these auxiliary tasks.

3.1 AUXILIARY CONTROL TASKS

The auxiliary control tasks we consider are defined as additional pseudo-reward functions in the environment the agent is interacting with. We formally define an auxiliary control task c by a reward function $r^{(c)} : \mathcal{S} \times \mathcal{A} \to \mathbb{R}$, where $\mathcal{S}$ is the space of possible states and $\mathcal{A}$ is the space of available actions. The underlying state space $\mathcal{S}$ includes both the history of observations and rewards as well as the state of the agent itself, i.e. the activations of the hidden units of the network.

Given a set of auxiliary control tasks $\mathcal{C}$, let $\pi^{(c)}$ be the agent's policy for each auxiliary task $c \in \mathcal{C}$ and let π be the agent's policy on the base task. The overall objective is to maximise total performance across all these auxiliary tasks,

$$\arg\max_{\theta} \mathbb{E}_{\pi}[R_{1:\infty}] + \lambda_c \sum_{c \in \mathcal{C}} \mathbb{E}_{\pi_c}[R^{(c)}_{1:\infty}], \tag{1}$$

where, $R^{(c)}_{t:t+n} = \sum_{k=1}^{n} \gamma^k r^{(c)}_t$ is the discounted return for auxiliary reward $r^{(c)}$, and θ is the set of parameters of π and all $\pi^{(c)}$'s. By sharing some of the parameters of π and all $\pi^{(c)}$ the agent must balance improving its performance with respect to the global reward r_t with improving performance on the auxiliary tasks.

In principle, any reinforcement learning method could be applied to maximise these objectives. However, to efficiently learn to maximise many different pseudo-rewards simultaneously in parallel from a single stream of experience, it is necessary to use off-policy reinforcement learning. We focus on value-based RL methods that approximate the optimal action-values by Q-learning. Specifically, for each control task c we optimise an n-step Q-learning loss $\mathcal{L}^{(c)}_Q = \mathbb{E}\left[\left(R_{t:t+n} + \gamma^n \max_{a'} Q^{(c)}(s', a', \theta^-) - Q^{(c)}(s, a, \theta)\right)^2\right]$, as described in Mnih et al. (2016).

While many types of auxiliary reward functions can be defined from these quantities we focus on two specific types:

- **Pixel changes** - Changes in the perceptual stream often correspond to important events in an environment. We train agents that learn a separate policy for maximally changing the pixels in each cell of an $n \times n$ non-overlapping grid placed over the input image. We refer to these auxiliary tasks as *pixel control*. See Section 4 for a complete description.
- **Network features** - Since the policy or value networks of an agent learn to extract task-relevant high-level features of the environment (Mnih et al., 2015; Zahavy et al., 2016; Silver et al., 2016) they can be useful quantities for the agent to learn to control. Hence, the activation of any hidden unit of the agent's neural network can itself be an auxiliary reward. We train agents that learn a separate policy for maximally activating each of the units in a specific hidden layer. We refer to these tasks as *feature control*.

The Figure 1 (b) shows an A3C agent architecture augmented with a set of auxiliary pixel control tasks. In this case, the base policy π shares both the convolutional visual stream and the LSTM with the auxiliary policies. The output of the auxiliary network head is an $N_{\text{act}} \times n \times n$ tensor Q^{aux} where $Q^{\text{aux}}(a, i, j)$ represents the network's current estimate of the optimal discounted expected change in cell (i, j) of the input after taking action a. We exploit the spatial nature of the auxiliary tasks by using a deconvolutional neural network to produce the auxiliary values Q^{aux}.

3.2 AUXILIARY REWARD TASKS

In addition to learning generally about the dynamics of the environment, an agent must learn to maximise the global reward stream. To learn a policy to maximise rewards, an agent requires features

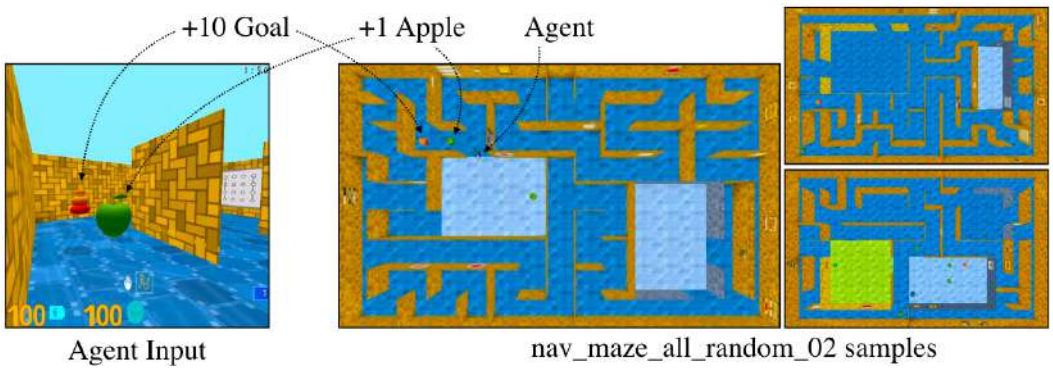


Figure 2: The raw RGB frame from the environment is the observation that is given as input to the agent, along with the last action and reward. This observation is shown for a sample of a maze from the nav_maze_all_random_02 level in Labyrinth. The agent must navigate this unseen maze and pick up apples giving +1 reward and reach the goal giving +10 reward, after which it will respawn. Top down views of samples from this maze generator show the variety of mazes procedurally created. A video showing the agent playing Labyrinth levels can be viewed at `https://youtu.be/Uz-zGYrYEjA`

that recognise states that lead to high reward and value. An agent with a good representation of rewarding states, will allow the learning of good value functions, and in turn should allow the easy learning of a policy.

However, in many interesting environments reward is encountered very sparsely, meaning that it can take a long time to train feature extractors adept at recognising states which signify the onset of reward. We want to remove the perceptual sparsity of rewards and rewarding states to aid the training of an agent, but to do so in a way which does not introduce bias to the agent's policy.

To do this, we introduce the auxiliary task of *reward prediction* – that of predicting the onset of immediate reward given some historical context. This task consists of processing a sequence of consecutive observations, and requiring the agent to predict the reward picked up in the subsequent unseen frame. This is similar to value learning focused on immediate reward ($\gamma = 0$).

Unlike learning a value function, which is used to estimate returns and as a baseline while learning a policy, the reward predictor is not used for anything other than shaping the features of the agent. This keeps us free to bias the data distribution, therefore biasing the reward predictor and feature shaping, without biasing the value function or policy.

We train the reward prediction task on sequences $S_\tau = (s_{\tau-k}, s_{\tau-k+1}, \ldots, s_{\tau-1})$ to predict the reward r_τ, and sample S_τ from the experience of our policy π in a skewed manner so as to over-represent rewarding events (presuming rewards are sparse within the environment). Specifically, we sample such that zero rewards and non-zero rewards are equally represented, i.e. the predicted probability of a non-zero reward is $P(r_\tau \neq 0) = 0.5$. The reward prediction is trained to minimise a loss $\mathcal{L}_{\text{RP}}$. In our experiments we use a multiclass cross-entropy classification loss across three classes (zero, positive, or negative reward), although a mean-squared error loss is also feasible.

The auxiliary reward predictions may use a different architecture to the agent's main policy. Rather than simply "hanging" the auxiliary predictions off the LSTM, we use a simpler feedforward network that concatenates a stack of states S_τ after being encoded by the agent's CNN, see Figure 1 (c). The idea is to simplify the temporal aspects of the prediction task in both the future direction (focusing only on immediate reward prediction rather than long-term returns) and past direction (focusing only on immediate predecessor states rather than the complete history); the features discovered in this manner is shared with the primary LSTM (via shared weights in the convolutional encoder) to enable the policy to be learned more efficiently.

3.3 Experience Replay

Experience replay has proven to be an effective mechanism for improving both the data efficiency and stability of deep reinforcement learning algorithms (Mnih et al., 2015). The main idea is to store transitions in a replay buffer, and then apply learning updates to sampled transitions from this buffer.

Experience replay provides a natural mechanism for skewing the distribution of reward prediction samples towards rewarding events: we simply split the replay buffer into rewarding and non-rewarding subsets, and replay equally from both subsets. The skewed sampling of transitions from

a replay buffer means that rare rewarding states will be oversampled, and learnt from far more frequently than if we sampled sequences directly from the behaviour policy. This approach can be viewed as a simple form of prioritised replay (Schaul et al., 2015b).

In addition to reward prediction, we also use the replay buffer to perform *value function replay*. This amounts to resampling recent historical sequences from the behaviour policy distribution and performing extra value function regression in addition to the on-policy value function regression in A3C. By resampling previous experience, and randomly varying the temporal position of the truncation window over which the n-step return is computed, value function replay performs value iteration and exploits newly discovered features shaped by reward prediction. We do not skew the distribution for this case.

Experience replay is also used to increase the efficiency and stability of the auxiliary control tasks. Q-learning updates are applied to sampled experiences that are drawn from the replay buffer, allowing features to be developed extremely efficiently.

3.4 *UNREAL* AGENT

The *UNREAL* algorithm combines the benefits of two separate, state-of-the-art approaches to deep reinforcement learning. The primary policy is trained with A3C (Mnih et al., 2016): it learns from parallel streams of experience to gain efficiency and stability; it is updated online using policy gradient methods; and it uses a recurrent neural network to encode the complete history of experience. This allows the agent to learn effectively in partially observed environments.

The auxiliary tasks are trained on very recent sequences of experience that are stored and randomly sampled; these sequences may be prioritised (in our case according to immediate rewards) (Schaul et al., 2015b); these targets are trained off-policy by Q-learning; and they may use simpler feedforward architectures. This allows the representation to be trained with maximum efficiency.

The *UNREAL* algorithm optimises a single combined loss function with respect to the joint parameters of the agent, θ, that combines the A3C loss $\mathcal{L}_{\mathrm{A3C}}$ together with an auxiliary control loss $\mathcal{L}_{\mathrm{PC}}$, auxiliary reward prediction loss $\mathcal{L}_{\mathrm{RP}}$ and replayed value loss $\mathcal{L}_{\mathrm{VR}}$,

$$\mathcal{L}_{\textit{UNREAL}}(\theta) = \mathcal{L}_{\mathrm{A3C}} + \lambda_{\mathrm{VR}}\mathcal{L}_{\mathrm{VR}} + \lambda_{\mathrm{PC}}\sum_{c}\mathcal{L}_{Q}^{(c)} + \lambda_{\mathrm{RP}}\mathcal{L}_{\mathrm{RP}} \tag{2}$$

where $\lambda_{\mathrm{VR}}, \lambda_{\mathrm{PC}}, \lambda_{\mathrm{RP}}$ are weighting terms on the individual loss components.

In practice, the loss is broken down into separate components that are applied either on-policy, directly from experience; or off-policy, on replayed transitions. Specifically, the A3C loss $\mathcal{L}_{\mathrm{A3C}}$ is minimised on-policy; while the value function loss $\mathcal{L}_{\mathrm{VR}}$ is optimised from replayed data, in addition to the A3C loss (of which it is one component, see Section 2). The auxiliary control loss $\mathcal{L}_{\mathrm{PC}}$ is optimised off-policy from replayed data, by n-step Q-learning. Finally, the reward loss $\mathcal{L}_{\mathrm{RP}}$ is optimised from rebalanced replay data.

4 EXPERIMENTS

In this section we give the results of experiments performed on the 3D environment *Labyrinth* in Section 4.1 and Atari in Section 4.2.

In all our experiments we used an A3C CNN-LSTM agent as our baseline and the *UNREAL* agent along with its ablated variants added auxiliary outputs and losses to this base agent. The agent is trained on-policy with 20-step returns and the auxiliary tasks are performed every 20 environment steps, corresponding to every update of the base A3C agent. The replay buffer stores the most recent 2k observations, actions, and rewards taken by the base agent. In Labyrinth we use the same set of 17 discrete actions for all games and on Atari the action set is game dependent (between 3 and 18 discrete actions). The full implementation details can be found in Section B.

4.1 LABYRINTH RESULTS

Labyrinth is a first-person 3D game platform extended from OpenArena (contributors, 2005), which is itself based on Quake3 (id software, 1999). Labyrinth is comparable to other first-person 3D game

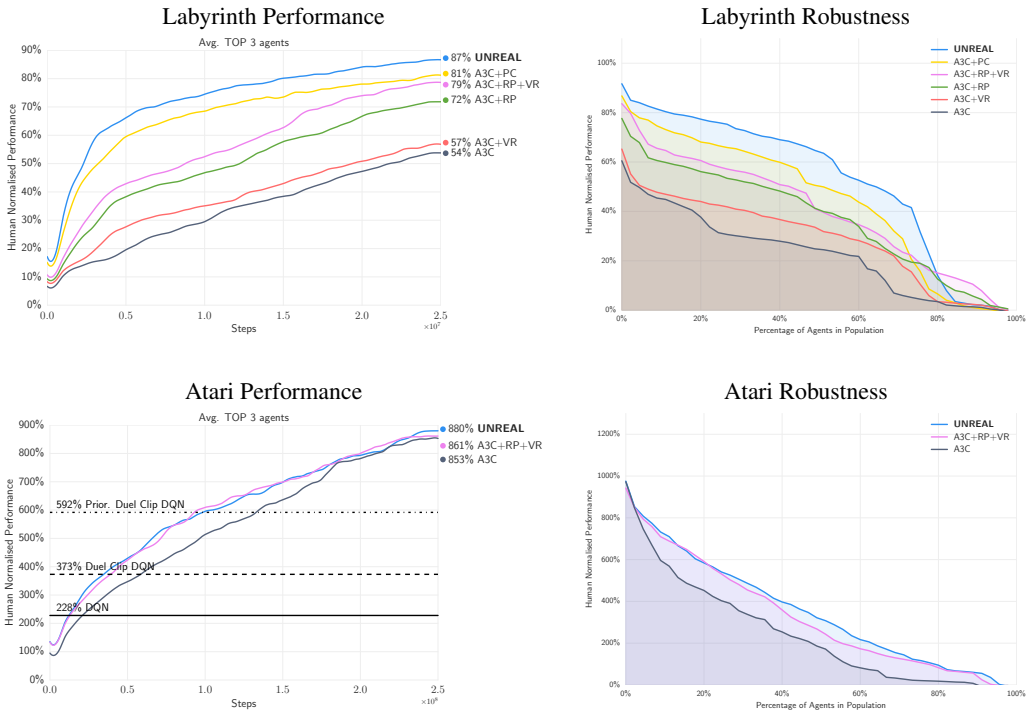


Figure 3: An overview of performance averaged across all levels on Labyrinth (Top) and Atari (Bottom). In the ablated versions RP is reward prediction, VR is value function replay, and PC is pixel control, with the *UNREAL* agent being the combination of all. *Left*: The mean human-normalised performance over last 100 episodes of the top-3 jobs at every point in training. We achieve an average of 87% human-normalised score, with every element of the agent improving upon the 54% human-normalised score of vanilla A3C. *Right*: The final human-normalised score of every job in our hyperparameter sweep, sorted by score. On both Labyrinth and Atari, the *UNREAL* agent increases the robustness to the hyperparameters (namely learning rate and entropy cost).

platforms for AI research like VizDoom (Kempka et al., 2016) or Minecraft (Tessler et al., 2016). However, in comparison, Labyrinth has considerably richer visuals and more realistic physics. Textures in Labyrinth are often dynamic (animated) so as to convey a game world where walls and floors shimmer and pulse, adding significant complexity to the perceptual task. The action space allows for fine-grained pointing in a fully 3D world. Unlike in VizDoom, agents can look up to the sky or down to the ground. Labyrinth also supports continuous motion unlike the Minecraft platform of (Oh et al., 2016), which is a 3D grid world.

We evaluated agent performance on 13 Labyrinth levels that tested a range of different agent abilities. A top-down visualization showing the layout of each level can be found in Figure 7 of the Appendix. A gallery of example images from the first-person perspective of the agent are in Figure 8 of the Appendix. The levels can be divided into four categories:

1. Simple fruit gathering levels with a static map (seekavoid_arena_01 and stairway_to_melon_01). The goal of these levels is to collect apples (small positive reward) and melons (large positive reward) while avoiding lemons (small negative reward).
2. Navigation levels with a static map layout (nav_maze_static_0$\{1, 2, 3\}$ and nav_maze_random_goal_0$\{1, 2, 3\}$). These levels test the agent's ability to find their way to a goal in a fixed maze that remains the same across episodes. The starting location is random. In this case, agents could encode the structure of the maze in network weights. In the random goal variant, the location of the goal changes in every episode. The optimal policy is to find the goal's location at the start of each episode and then use long-term knowledge of the maze layout to return to it as quickly as possible from any location. The static variant is simpler in that the goal location is always fixed for all episodes and only the agent's starting location changes so the optimal policy does not require the first step of exploring to find the current goal location.
3. Procedurally-generated navigation levels requiring effective exploration of a new maze generated on-the-fly at the start of each episode (nav_maze_all_random_0$\{1, 2, 3\}$). These levels test the agent's ability to effectively explore a totally new environment. The optimal

policy would begin by exploring the maze to rapidly learn its layout and then exploit that knowledge to repeatedly return to the goal as many times as possible before the end of the episode (between 60 and 300 seconds).

4. Laser-tag levels requiring agents to wield laser-like science fiction gadgets to tag bots controlled by the game's in-built AI (lt_horse_shoe_color and lt_hallway_slope). A reward of 1 is delivered whenever the agent tags a bot by reducing its shield to 0. These levels approximate the default OpenArena/Quake3 gameplay mode. In lt_hallway_slope there is a sloped arena, requiring the agent to look up and down. In lt_horse_shoe_color, the colors and textures of the bots are randomly generated at the start of each episode. This prevents agents from relying on color for bot detection. These levels test aspects of fine-control (for aiming), planning (to anticipate where bots are likely to move), strategy (to control key areas of the map such as gadget spawn points), and robustness to the substantial visual complexity arising from the large numbers of independently moving objects (gadget projectiles and bots).

4.1.1 RESULTS

We compared the full *UNREAL* agent to a basic A3C LSTM agent along with several ablated versions of *UNREAL* with different components turned off. A video of the final agent performance, as well as visualisations of the activations and auxiliary task outputs can be viewed at `https://youtu.be/Uz-zGYrYEjA`.

Figure 3 (right) shows curves of mean human-normalised scores over the 13 Labyrinth levels. Adding each of our proposed auxiliary tasks to an A3C agent substantially improves the performance. Combining different auxiliary tasks leads to further improvements over the individual auxiliary tasks. The *UNREAL* agent, which combines all three auxiliary tasks, achieves more than twice the final human-normalised mean performance of A3C, increasing from 54% to 87% (45% to 92% for median performance). This includes a human-normalised score of 116% on lt_hallway_slope and 100% on nav_maze_random_goal_02.

Perhaps of equal importance, aside from final performance on the games, *UNREAL* is significantly faster at learning and therefore more data efficient, achieving a mean speedup of the number of steps to reach A3C best performance of $10\times$ (median $11\times$) across all levels and up to $18\times$ on nav_maze_random_goal_02. This translates in a drastic improvement in the data efficiency of *UNREAL* over A3C, requiring less than 10% of the data to reach the final performance of A3C. We can also measure the robustness of our learning algorithms to hyperparameters by measuring the performance over all hyperparameters (namely learning rate and entropy cost). This is shown in Figure 3 Top: every auxiliary task in our agent improves robustness. A breakdown of the performance of A3C, *UNREAL* and *UNREAL* without pixel control on the individual Labyrinth levels is shown in Figure 4.

Unsupervised Reinforcement Learning In order to better understand the benefits of auxiliary control tasks we compared it to two simple baselines on three Labyrinth levels. The first baseline was A3C augmented with a pixel reconstruction loss, which has been shown to improve performance on 3D environments (Kulkarni et al., 2016). The second baseline was A3C augmented with an input change prediction loss, which can be seen as simply predicting the immediate auxiliary reward instead of learning to control. Finally, we include preliminary results for A3C augmented with the feature control auxiliary task on one of the levels. We retuned the hyperparameters of all methods (including learning rate and the weight placed on the auxiliary loss) for each of the three Labyrinth levels. Figure 5 shows the learning curves for the top 5 hyperparameter settings on three Labyrinth navigation levels. The results show that learning to control pixel changes is indeed better than simply predicting immediate pixel changes, which in turn is better than simply learning to reconstruct the input. In fact, learning to reconstruct only led to faster initial learning and actually made the final scores worse when compared to vanilla A3C. Our hypothesis is that input reconstruction hurts final performance because it puts too much focus on reconstructing irrelevant parts of the visual input instead of visual cues for rewards, which rewarding objects are rarely visible. Encouragingly, we saw an improvement from including the feature control auxiliary task. Combining feature control with other auxiliary tasks is a promising future direction.

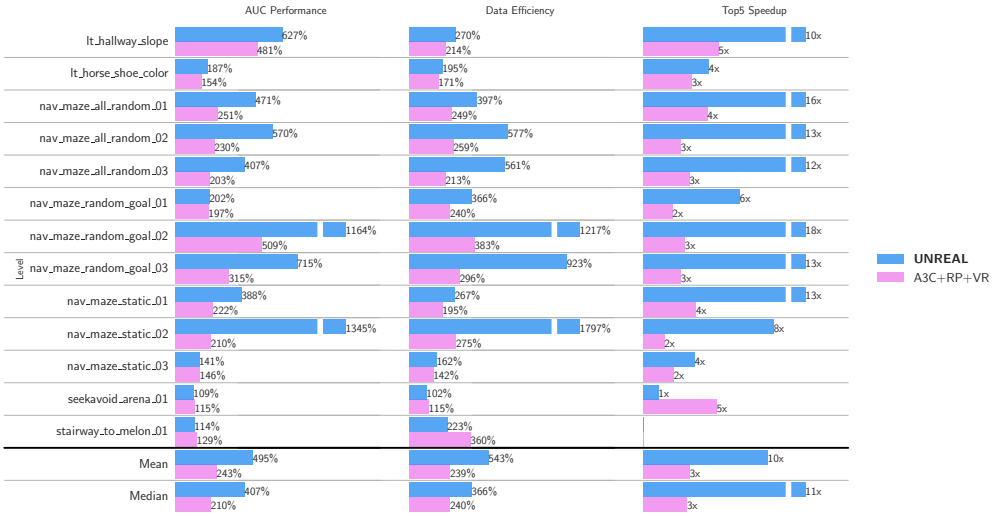


Figure 4: A breakdown of the improvement over A3C due to our auxiliary tasks for each level on Labyrinth. The values for A3C+RP+VR (reward prediction and value function replay) and *UNREAL* (reward prediction, value function replay and pixel control) are normalised by the A3C value. AUC Performance gives the robustness to hyperparameters (area under the robustness curve Figure 3 Right). Data Efficiency is area under the mean learning curve for the top-5 jobs, and Top5 Speedup is the speedup for the mean of the top-5 jobs to reach the maximum top-5 mean score set by A3C. Speedup is not defined for stairway_to_melon as A3C did not learn throughout training.

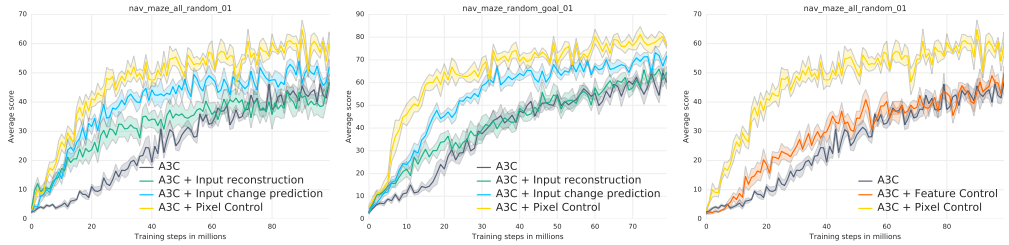


Figure 5: Comparison of various forms of self-supervised learning on random maze navigation. Adding an input reconstruction loss to the objective leads to faster learning compared to an A3C baseline. Predicting changes in the inputs works better than simple image reconstruction. Learning to control changes leads to the best results.

4.2 Atari

We applied the *UNREAL* agent as well as *UNREAL* without pixel control to 57 Atari games from the Arcade Learning Environment (Bellemare et al., 2012) domain. We use the same evaluation protocol as for our Labyrinth experiments where we evaluate 50 different random hyper parameter settings (learning rate and entropy cost) on each game. The results are shown in the bottom row of Figure 3. The left side shows the average performance curves of the top 3 agents for all three methods the right half shows sorted average human-normalised scores for each hyperparameter setting. More detailed learning curves for individual levels can be found in Figure 7. We see that *UNREAL* surpasses the current state-of-the-art agents, *i.e.* A3C and Prioritized Dueling DQN (Wang et al., 2016), across all levels attaining 880% mean and 250% median performance. Notably, *UNREAL* is also substantially more robust to hyper parameter settings than A3C.

5 Conclusion

We have shown how augmenting a deep reinforcement learning agent with auxiliary control and reward prediction tasks can drastically improve both data efficiency and robustness to hyperparameter settings. Most notably, our proposed *UNREAL* architecture more than doubled the previous state-of-the-art results on the challenging set of 3D Labyrinth levels, bringing the average scores to over 87% of human scores. The same *UNREAL* architecture also significantly improved both the learning speed and the robustness of A3C over 57 Atari games.

ACKNOWLEDGEMENTS

We thank Charles Beattie, Julian Schrittwieser, Marcus Wainwright, and Stig Petersen for environment design and development, and Amir Sadik and Sarah York for expert human game testing. We also thank Joseph Modayil, Andrea Banino, Hubert Soyer, Razvan Pascanu, and Raia Hadsell for many helpful discussions.

REFERENCES

André Barreto, Rémi Munos, Tom Schaul, and David Silver. Successor features for transfer in reinforcement learning. *arXiv preprint arXiv:1606.05312*, 2016.

Marc G Bellemare, Yavar Naddaf, Joel Veness, and Michael Bowling. The arcade learning environment: An evaluation platform for general agents. *Journal of Artificial Intelligence Research*, 2012.

OpenArena contributors. The openarena manual. 2005. URL `http://openarena.wikia.com/wiki/Manual`.

Peter Dayan. Improving generalization for temporal difference learning: The successor representation. *Neural Computation*, 5(4):613–624, 1993.

Felix A Gers, Jürgen Schmidhuber, and Fred Cummins. Learning to forget: Continual prediction with lstm. *Neural computation*, 12(10):2451–2471, 2000.

id software. Quake3. 1999. URL `https://github.com/id-Software/Quake-III-Arena`.

Michał Kempka, Marek Wydmuch, Grzegorz Runc, Jakub Toczek, and Wojciech Jaśkowski. Vizdoom: A doom-based ai research platform for visual reinforcement learning. *arXiv preprint arXiv:1605.02097*, 2016.

George Konidaris and Andre S Barreto. Skill discovery in continuous reinforcement learning domains using skill chaining. In *Advances in Neural Information Processing Systems*, pp. 1015–1023, 2009.

Tejas D Kulkarni, Ardavan Saeedi, Simanta Gautam, and Samuel J Gershman. Deep successor reinforcement learning. *arXiv preprint arXiv:1606.02396*, 2016.

Guillaume Lample and Devendra Singh Chaplot. Playing FPS games with deep reinforcement learning. *CoRR*, abs/1609.05521, 2016.

Xiujun Li, Lihong Li, Jianfeng Gao, Xiaodong He, Jianshu Chen, Li Deng, and Ji He. Recurrent reinforcement learning: A hybrid approach. *arXiv preprint arXiv:1509.03044*, 2015.

Long-Ji Lin and Tom M Mitchell. Memory approaches to reinforcement learning in non-markovian domains. Technical report, Carnegie Mellon University, School of Computer Science, 1992.

Piotr Mirowski, Razvan Pascanu, Fabio Viola, Andrea Banino, Hubert Soyer, Andy Ballard, Misha Denil, Ross Goroshin, Laurent Sifre, Koray Kavukcuoglu, Dharshan Kumaran, and Raia Hadsell. Learning to navigate in complex environments. 2016.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning. In *NIPS Deep Learning Workshop*. 2013.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Andrei A. Rusu, Joel Veness, Marc G. Bellemare, Alex Graves, Martin Riedmiller, Andreas K. Fidjeland, Georg Ostrovski, Stig Petersen, Charles Beattie, Amir Sadik, Ioannis Antonoglou, Helen King, Dharshan Kumaran, Daan Wierstra, Shane Legg, and Demis Hassabis. Human-level control through deep reinforcement learning. *Nature*, 518(7540):529–533, 02 2015. URL `http://dx.doi.org/10.1038/nature14236`.

Volodymyr Mnih, Adrià Puigdomènech Badia, Mehdi Mirza, Alex Graves, Timothy P. Lillicrap, Tim Harley, David Silver, and Koray Kavukcuoglu. Asynchronous methods for deep reinforcement learning. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, pp. 1928–1937, 2016.

Junhyuk Oh, Xiaoxiao Guo, Honglak Lee, Richard L Lewis, and Satinder Singh. Action-conditional video prediction using deep networks in atari games. In *Advances in Neural Information Processing Systems*, pp. 2863–2871, 2015.

Junhyuk Oh, Valliappa Chockalingam, Satinder Singh, and Honglak Lee. Control of memory, active perception, and action in minecraft. *arXiv preprint arXiv:1605.09128*, 2016.

Jing Peng and Ronald J Williams. Incremental multi-step q-learning. *Machine Learning*, 22(1-3): 283–290, 1996.

Daniel L Schacter, Donna Rose Addis, Demis Hassabis, Victoria C Martin, R Nathan Spreng, and Karl K Szpunar. The future of memory: remembering, imagining, and the brain. *Neuron*, 76(4): 677–694, 2012.

Tom Schaul, Daniel Horgan, Karol Gregor, and David Silver. Universal value function approximators. In *Proceedings of the 32nd International Conference on Machine Learning (ICML-15)*, pp. 1312–1320, 2015a.

Tom Schaul, John Quan, Ioannis Antonoglou, and David Silver. Prioritized experience replay. *arXiv preprint arXiv:1511.05952*, 2015b.

Jürgen Schmidhuber. Formal theory of creativity, fun, and intrinsic motivation (1990–2010). *IEEE Transactions on Autonomous Mental Development*, 2(3):230–247, 2010.

David Silver and Kamil Ciosek. Compositional planning using optimal option models. *arXiv preprint arXiv:1206.6473*, 2012.

David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George Van Den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

Richard S Sutton, David A McAllester, Satinder P Singh, Yishay Mansour, et al. Policy gradient methods for reinforcement learning with function approximation. In *NIPS*, volume 99, pp. 1057–1063, 1999a.

Richard S Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial intelligence*, 1999b.

Richard S Sutton, Joseph Modayil, Michael Delp, Thomas Degris, Patrick M Pilarski, Adam White, and Doina Precup. Horde: A scalable real-time architecture for learning knowledge from unsupervised sensorimotor interaction. In *The 10th International Conference on Autonomous Agents and Multiagent Systems-Volume 2*, pp. 761–768. International Foundation for Autonomous Agents and Multiagent Systems, 2011.

Chen Tessler, Shahar Givony, Tom Zahavy, Daniel J Mankowitz, and Shie Mannor. A deep hierarchical approach to lifelong learning in minecraft. *arXiv preprint arXiv:1604.07255*, 2016.

Z. Wang, N. de Freitas, and M. Lanctot. Dueling Network Architectures for Deep Reinforcement Learning. In *Proceedings of the 33rd International Conference on Machine Learning (ICML)*, 2016.

Christopher John Cornish Hellaby Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge England, 1989.

Christopher Xie, Sachin Patil, Teodor Mihai Moldovan, Sergey Levine, and Pieter Abbeel. Model-based reinforcement learning with parametrized physical models and optimism-driven exploration. *CoRR*, abs/1509.06824, 2015.

Tom Zahavy, Nir Ben Zrihem, and Shie Mannor. Graying the black box: Understanding dqns. In *Proceedings of the 33rd International Conference on Machine Learning*, 2016.

A ATARI GAMES

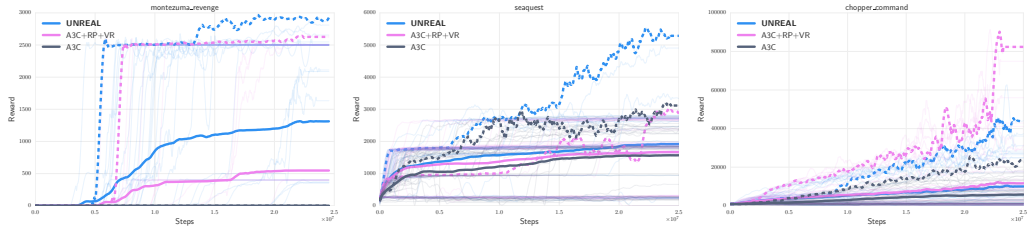


Figure 6: Learning curves for three example Atari games. Semi-transparent lines are agents with different seeds and hyperparameters, the bold line is a mean over population and dotted line is the best agent (in terms of final performance).

B IMPLEMENTATION DETAILS

The input to the agent at each timestep was an 84×84 RGB image. All agents processed the input with the convolutional neural network (CNN) originally used for Atari by Mnih et al. (2013). The network consists of two convolutional layers. The first one has 16 8×8 filters applied with stride 4, while the second one has 32 4×4 filters with stride 2. This is followed by a fully connected layer with 256 units. All three layers are followed by a ReLU non-linearity. All agents used an LSTM with forget gates (Gers et al., 2000) with 256 cells which take in the CNN-encoded observation concatenated with the previous action taken and curren:t reward. The policy and value function are linear projections of the LSTM output. The agent is trained with 20-step unrolls. The action space of the agent in the environment is game dependent for Atari (between 3 and 18 discrete actions), and 17 discrete actions for Labyrinth. Labyrinth runs at 60 frames-per-second. We use an action repeat of four, meaning that each action is repeated four times, with the agent receiving the final fourth frame as input to the next processing step.

For the pixel control auxiliary tasks we trained policies to control the central 80×80 crop of the inputs. The cropped region was subdivided into a 20×20 grid of non-overlapping 4×4 cells. The instantaneous reward in each cell was defined as the average absolute difference from the previous frame, where the average is taken over both pixels and channels in the cell. The output tensor of auxiliary values, Q^{aux}, is produced from the LSTM outputs by a deconvolutional network. The LSTM outputs are first mapped to a $32 \times 7 \times 7$ spatial feature map with a linear layer followed by a ReLU. Deconvolution layers with 1 and N_{act} filters of size 4×4 and stride 2 map the $32 \times 7 \times 7$ into a value tensor and an advantage tensor respectively. The spatial map is then decoded into Q-values using the dueling parametrization (Wang et al., 2016) producing the $N_{\text{act}} \times 20 \times 20$ output Q^{aux}.

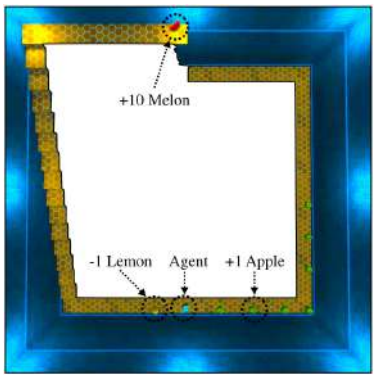
The architecture for feature control was similar. We learned to control the second hidden layer, which is a spatial feature map with size $32 \times 9 \times 9$. Similarly to pixel control, we exploit the spatial structure in the data and used a deconvolutional network to produce Q^{aux} from the LSTM outputs. Further details are included in the supplementary materials.

The reward prediction task is performed on a sequence of three observations, which are fed through three instances of the agent's CNN. The three encoded CNN outputs are concatenated and fed through a fully connected layer of 128 units with ReLU activations, followed by a final linear three-class classifier and softmax. The reward is predicted as one of three classes: positive, negative, or zero and trained with a task weight $\lambda_{\text{RP}} = 1$. The value function replay is performed on a sequence of length 20 with a task weight $\lambda_{\text{VR}} = 1$.

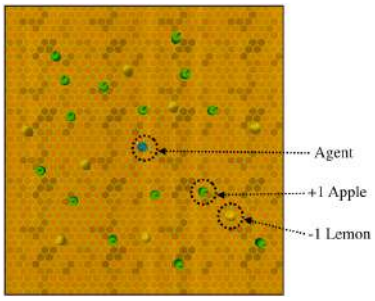
The auxiliary tasks are performed every 20 environment steps, corresponding to every update of the base A3C agent, once the replay buffer has filled with agent experience. The replay buffer stores the most recent 2k observations, actions, and rewards taken by the base agent.

The agents are optimised over 32 asynchronous threads with shared RMSprop (Mnih et al., 2016). The learning rates are sampled from a log-uniform distribution between 0.0001 and 0.005. The entropy costs are sampled from the log-uniform distribution between 0.0005 and 0.01. Task weight λ_{PC} is sampled from log-uniform distribution between 0.01 and 0.1 for Labyrinth and 0.0001 and 0.01 for Atari (since Atari games are not homogeneous in terms of pixel intensities changes, thus we need to fit this normalization factor).

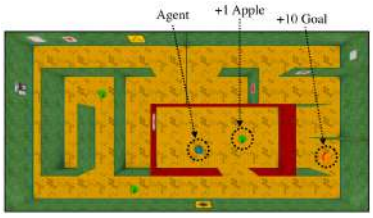
C LABYRINTH LEVELS



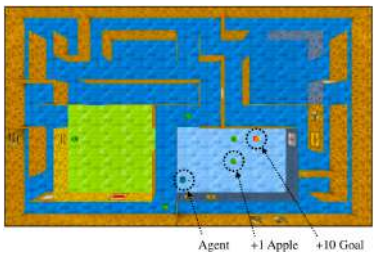
stairway_to_melon



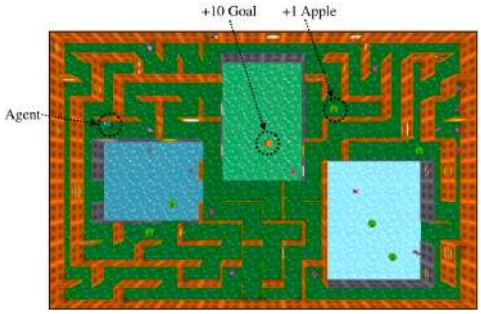
seekavoid_arena_01



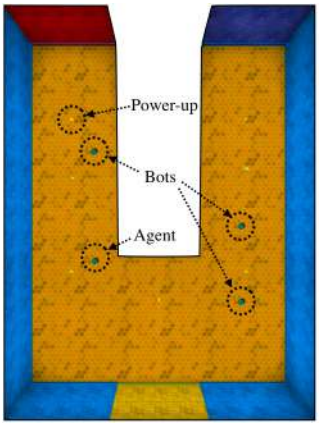
nav_maze ∗ 01



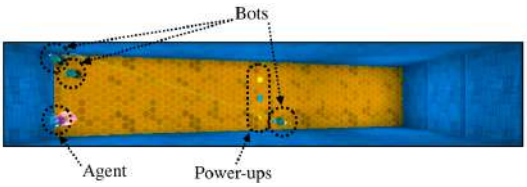
nav_maze ∗ 02



nav_maze ∗ 03



lt_horse_shoe_color



lt_hallway_slope

Figure 7: Top-down renderings of each Labyrinth level. The $\mathrm{nav\_maze} * \_0\{1, 2, 3\}$ levels show one example maze layout. In the $\mathrm{all\_random}$ case, a new maze was randomly generated at the start of each episode.

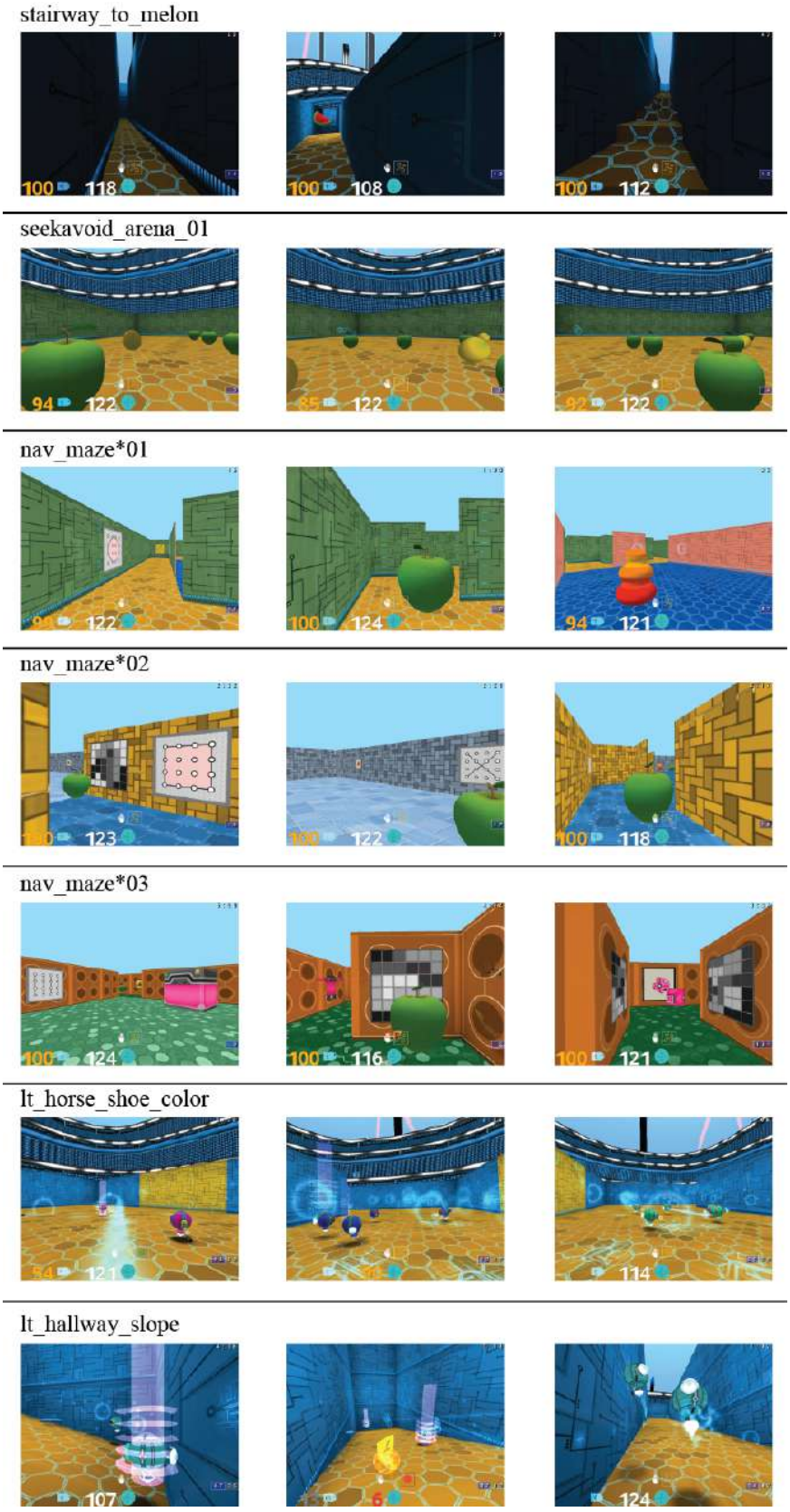


Figure 8: Example images from the agent's egocentric viewpoint for each Labyrinth level.