



Figure 11: The map of PitFall!, cropped from https://atariage.com/2600/archives/strategy_pitfall_map.html?SystemID=2600

A ENVIRONMENT DETAILS

A.1 MINIGRID ENVIRONMENT DETAILS

- **MultiRoom-NxSy (MR)**: This is a sequential navigation environment composed of interconnected rooms with initially unlocked doors. Agents must traverse multiple rooms to reach a terminal goal state (marked by a green square), which provides the sole extrinsic reward. The parameters x and y denote the number of rooms and their respective sizes.
- **KeyCorridorSxR3 (KC)**: A puzzle-solving environment featuring a central corridor that connects multiple rooms via locked or unlocked doors. The agent’s task is to first locate a key in an accessible room, then use it to unlock the target room containing the objective item. The variable x controls the room count.
- **ObstructedMaze (OM)**: A complex 3×3 grid maze incorporating three primary challenge components: (1) locked doors requiring key retrieval, (2) keys concealed within containers, and (3) door obstructions caused by movable balls. The sole extrinsic reward is granted for box retrieval in designated corners. Four variants are used for evaluation:
 - **OM-2Dlh**: A simplified version with exactly 2 locked doors and keys stored in boxes.
 - **OM-1Q/2Q**: Partial mazes consisting of either 1 or 2 quadrants of the full 3×3 structure.
 - **OM-Full**: The complete 3×3 maze configuration, comprising 4 interconnected quadrants.

A.2 CHALLENGING ATARI ENVIRONMENT DETAILS

PitFall!. This environment consists of 255 interconnected rooms with only 32 extrinsic rewards (treasures) sparsely distributed across various locations. Most rooms contain hazardous elements (e.g., snakes, sand pits) that immediately terminate an episode upon contact, thus demanding precise,

long-horizon action sequences for successful navigation. A detailed map of PitFall! is provided in Figure 11.

Montezuma’s Revenge (MR). This is a notoriously challenging hard-exploration environment featuring three distinct levels. Each level consists of 24 interconnected rooms with various hazardous elements (e.g., laser gates, skulls) and traps. The agent must navigate these levels, collect items, and avoid enemies. As discussed in Section 4.3, the environment’s deterministic nature and cyclical progression (completing Level 3 returns the agent to its beginning) allow for the theoretical accumulation of infinite scores under specific conditions.

Domain Knowledge Extraction. Building upon insights from Ecoffet et al. (2021), we extract domain-specific knowledge by employing pixel-color detection of the agent’s face to determine its positional information. Room indices are subsequently derived through an analysis of positional shifting patterns. For MR, we additionally utilize the level index and the count of positive extrinsic rewards.

Domain Knowledge as Heuristic Model Input. We transform the extracted room index, agent position, level information, and extrinsic reward data into an 11×8 integer matrix (range: 0-7), initialized as a zero matrix. Positional information is encoded by setting relevant matrix elements to 7. Room indices, level, and extrinsic reward data are represented as octal digits within the matrix header. This matrix is then converted into an $8 \times 11 \times 8$ one-hot encoded tensor (channel-first) for input to the heuristic model.

Domain Knowledge for Sub-optimal Path Pruning. For PitFall!, we utilize room index and agent positional information to prune underground and leftward sub-optimal paths. Specifically, any y-position exceeding a predefined threshold is classified as an underground path, while the room index serves as an indicator for leftward paths. For MR, similar pruning methods are applied to prevent the acquisition of deceptive positive rewards. In each level, there are 6 doors but only 4 keys; selecting sub-optimal doors can provide positive rewards but permanently blocks advancement to the next level. Identifying and shaping such deceptive signals through long-term planning is typically beyond the scope of core efficient exploration methods, which this paper addresses.

Random Noise for Robustness Analysis. For the random noise setting discussed in Section 5, the last 8 elements of the domain knowledge array are overwritten with uniformly sampled integers from the range $[0, 7]$.

Stochasticity Settings. For the stochasticity analysis in Section 5, we implement a sticky-action wrapper. This wrapper introduces a 0.25 probability of repeating the previous action, contrasted with a 0.75 probability of executing the agent’s current action selection.

Preprocessing Pipeline. We apply the following preprocessing pipeline for the PitFall! environment:

- **Frame Skipping:** Each action is repeated for 4 consecutive timesteps. The environment returns the accumulated reward and the final observed frame.
- **Observation Preprocessing:**
 - The raw RGB observation ($210 \times 160 \times 3$) is converted to grayscale.
 - Downsampled to 84×84 resolution.
 - Normalized by scaling pixel values to $[0, 1)$ via division by 256.
- **Policy Network Input:** The policy network receives a stacked tensor of the 4 previously preprocessed frames (each 84×84), resulting in an input shape of $4 \times 84 \times 84$.

Episodic Frame Limits. The main experiments employ an 18,000-frame episode limit (matching PitFall!’s 20-minute time limit) for both PitFall! and MR. For the discussion section experiments, a 4,500-frame limit is used for PitFall!. During the post-processing phase for MR, the frame limit was extended to 180,000. All reported frames correspond to training frames, where frames skipped by the FrameSkip wrapper are excluded from the count.

B CODEBASE USED

Our codebase was built on the top of the following codebases:

- An Open-source RND implementation codebase Kazemipour (2020): <https://github.com/alirezakazemipour/PPO-RND> (MIT License) for the code structure/organization, the PPO with intrinsic reward framework and the Atari (for PitFall!) training details.
- The official NovelD codebase: <https://github.com/tianjunz/NovelD> (Creative Commons Attribution-NonCommercial 4.0 license) for the NovelD implementation details in MiniGrid.
- The official MADE codebase: <https://github.com/tianjunz/MADE> for the MADE implementation details.
- The official RIDE codebase: <https://github.com/facebookresearch/impact-driven-exploration> (Creative Commons Attribution-NonCommercial 4.0 license) for the RIDE implementation details.
- The official Go-Explore codebase: <https://github.com/uber-research/go-explore/> (Uber Non-Commercial License) for the domain knowledge extraction for PitFall!.

C EXPERIMENTS COMPUTE RESOURCES

All experiments were conducted on a high-performance server equipped with 2 AMD EPYC 7763 64-core CPUs, 8 NVIDIA RTX 4090 GPUs (24GB VRAM each), and 500GB RAM. For the PitFall!/MR experiments, each run required approximately 10/20 days of wall time utilizing 10 CPU processes and 5GB GPU memory. The MiniGrid experiments consumed about one day per run for 40M frames while using 2 CPU processes and 1.5GB GPU memory, with computational time scaling linearly with frame count.

D NEURAL NETWORK ARCHITECTURES

This section details the neural network architectures employed in our experiments. Unless otherwise specified, none of the architectures incorporate additional normalization layers.

D.1 MINIGRID

We maintain architectural consistency across all experiments, using identical network structures for both the policy model and intrinsic reward feature extractor. The sole exception is ETD, whose implementation failed to achieve meaningful extrinsic rewards in most environments. The network architecture of the CNN feature extractor is

- Conv2d(in=3, out=32, kernel=3, stride=1, pad=1),
- Conv2d(in=32, out=64, kernel=3, stride=2, pad=1),
- Conv2d(in=64, out=64, kernel=3, stride=2, pad=1),
- FC(in=256, out=128).

For the ETD implementation, we apply Layer Normalization to all convolutional and fully-connected layers in both the policy network and the intrinsic reward model. The network architecture of the CNN feature extractor is

- Conv2d(in=3, out=32, kernel=2, stride=1, pad=0),
- Conv2d(in=32, out=64, kernel=2, stride=1, pad=0),
- Conv2d(in=64, out=64, kernel=2, stride=1, pad=0),
- FC(in=1024, out=128).

D.2 PITFALL!

The feature extractor of the policy network is

- Conv2d(in=4, out=32, kernel=8, stride=4, pad=0),
- Conv2d(in=32, out=64, kernel=4, stride=2, pad=0),
- Conv2d(in=64, out=64, kernel=3, stride=1, pad=0),
- FC(in=3136, out=512).

The feature extractor of the heuristic model is

- Conv2d(in=8, out=32, kernel=3, stride=1, pad=1),
- Conv2d(in=32, out=64, kernel=3, stride=2, pad=1),
- Conv2d(in=64, out=64, kernel=3, stride=2, pad=1),
- FC(in=384, out=128).

E HYPERPARAMETERS

We provide complete hyperparameter configurations for both the PPO-with-intrinsic-reward framework and all baseline methods in the following tables.

Table 1: Hyperparameters for the PPO-with-intrinsic-reward framework.

Hyperparameter	MiniGrid	PitFall!	MR
learning rate	1e-4	2.5e-4	2.5e-4
γ_{ext}	0.999	0.999	0.99
γ_{int}	0.99	0.99	0.99
λ_{gae}	0.95	0.95	0.95
num workers	32	128	128
rollout length	128	512	512
num mini-batches	4	16	16
PPO clip range	0.1	0.1	0.1
RNN/CNN policy	CNN	CNN	CNN
Optimizer	Adam	Adam	Adam
num skipped frames	1	4	4
num stacked frames (for policy model)	1	4	4
extrinsic reward coef	12	12	1
entropy	1e-3	1e-3	1e-3

Table 2: Hyperparameters for SLA-v3.

Hyperparameter	MultiRoom KeyCorridor	ObstructedMaze	PitFall!	MR
SLA heuristic scale	0.06	0.01	0.01	0.01
leaky Relu ϵ^2	1/1000	1/1000	1/1000	1/1000
H_θ architecture	value model	value model	value model	value model
RND modulator	True	True	False	False
RND state (input) norm	False	False	-	-
RND novelty (output) norm	False	False	-	-
intrinsic reward coef	1	1	3	6

Table 3: Hyperparameters for NovelD in MiniGrid

Hyperparameter	MultiRoom KeyCorridor	ObstructedMaze
Use ERIR	True	True
intrinsic reward coef	0.1	0.01
α_{noveld}	0.5	0.5
β_{noveld}	0	0
RND state (input) norm	False	False
RND novelty (output) norm	False	False

Table 4: Hyperparameters for RIDE in MiniGrid

Hyperparameter	MiniGrid
Use ERIR	True
intrinsic reward coef	0.1

Table 5: Hyperparameters for RND in MiniGrid

Hyperparameter	MiniGrid
intrinsic reward coef	0.1
RND state (input) norm	False
RND novelty (output) norm	False

Table 6: Hyperparameters for ETD in MiniGrid

Hyperparameter	MiniGrid
$\gamma_{future-sampling}$	0.99
intrinsic reward coef	0.01
aggregate function	min
energy function	mrn-pot
loss function	inforce-symmetric
ETD reward norm	True
ETD reward norm momentum	0.9
NN norm type	LayerNorm

F REPRODUCTION CHALLENGES OF ETD

Despite extensive efforts to replicate the ETD algorithm’s reported performance—including dedicated investigations into implementation details exclusively for ETD such as normalization layers and neural network architectures—our reproduction attempts were unsuccessful. Notably, when using the exact hyperparameters and network architecture specified in the original implementation (including the direct LayerNorm connection to raw observations), our reproduction achieved inferior performance compared to our own carefully tuned configuration (reported in the main paper). We identify several potential factors contributing to this discrepancy:

- **Software Version Discrepancies:** Our codebase utilizes Python 3.7, while the original ETD implementation was developed with Python 3.9. Version differences in critical packages (e.g., PyTorch) may lead to divergent numerical behaviors.
- **Quasi-metric Computation Pipeline:** The original implementation employs a three-stage process: (1) observation-to-feature extraction, (2) feature-to-embedding transformation, and (3) embedding-to-quasi-metric calculation. Crucially, it maintains an episodic feature archive, recomputing embeddings for all states when new observations arrive. In contrast, our implementation maintains an embedding archive, potentially amplifying differences as the learning rate increases and the maximum episode length substantially exceeds the rollout length.
- **Implementation-Specific Optimizations:** The original ETD implementation builds upon stable-baseline3, which may incorporate undocumented but performance-critical optimizations.
- **Latent Implementation Artifacts:** While our codebase successfully reproduces other baseline results and achieves decent performance for SLA-v3, subtle implementation errors may persist that specifically degrade ETD’s performance.

G LLM USAGE STATEMENT

We affirm that all authors take full responsibility for the content of this submission.

In the preparation of this manuscript, large language models (LLMs) were utilized as general-purpose assistive tools to enhance clarity and presentation. Specifically, LLMs were employed for minor text polishing, grammatical corrections, and refining the visual representation and accompanying explanations of figures, such as Figure 1. These applications were limited to improving the readability and aesthetic quality of the text and figures, and did not involve any significant contribution to the core research ideation, experimental design, result analysis, or the generation of scientific facts or arguments. No LLM has been considered for authorship.