LEARNING TRANSFERABLE SKILLS IN ACTION RPGS VIA DIRECTED SKILL GRAPHS AND SELECTIVE ADAPTATION

Ali Najar
Sharif University of Technology
anajar13750@gmail.com

ABSTRACT

Lifelong agents should expand their competence over time without retraining from scratch or overwriting previously learned behaviors. We investigate this in a challenging real-time control setting (Dark Souls III) by representing combat as a directed skill graph and training its components in a hierarchical curriculum. The resulting agent decomposes control into five reusable skills: camera control, target lock-on, movement, dodging, and a heal–attack decision policy, each optimized for a narrow responsibility. This factorization improves sample efficiency by reducing the burden on any single policy and supports selective post-training: when the environment shifts from Phase 1 to Phase 2, only a subset of skills must be adapted, while upstream skills remain transferable. Empirically, we find that targeted fine-tuning of just two skills rapidly recovers performance under a limited interaction budget, suggesting that skill-graph curricula together with selective fine-tuning offer a practical pathway toward evolving, continually learning agents in complex real-time environments.

1 INTRODUCTION

Lifelong agents should continuously evolve their capabilities, integrating new skills without the inefficiency of retraining from scratch or the risk of overwriting established behaviors. This objective, often studied under continual and lifelong reinforcement learning, is challenging because the agent must balance plasticity (rapid adaptation) with stability (retaining useful skills) under non-stationarity and limited interaction budgets (Hadsell et al., 2020; Khetarpal et al., 2020). In practice, monolithic end-to-end policies can be sample-inefficient and brittle when a task shifts, since the same parameters must simultaneously represent multiple competencies and their interactions.

These issues are amplified in complex real-time control domains such as modern video games. Most role-playing games feature tight reaction loops, partial observability, long-horizon credit assignment, and coupled subproblems. While large-scale deep RL has demonstrated that high-performance real-time game play is possible (Vinyals et al., 2019; Berner et al., 2019; Csereoka et al., 2022; O'Connor et al., 2025), achieving robust, transferable behavior typically requires extensive data and carefully engineered training protocols. A long-standing strategy for improving tractability is to introduce temporal abstraction and hierarchy (Pateria et al., 2021): the options framework formalizes temporally extended skills as reusable actions (Sutton et al., 1999), and the option-critic architecture learns such skills end-to-end (Bacon et al., 2016). More recently, multiple lines of work have emphasized structuring skills to improve exploration and composition, including graph-based skill representations (Bagaria et al., 2021; Lee et al., 2022).

Complementary to hierarchy, modular architectures reduce interference and encourage reuse by decomposing control into reusable components. In multi-task and lifelong settings, modular policies can improve sample efficiency and transfer by reusing submodules rather than relearning entire policies per task (Yang et al., 2020; Mendez et al., 2022). Curriculum learning offers a related lever: sequencing skills can substantially reduce the data needed in difficult domains (Narvekar et al., 2020). Together, these ideas motivate a directed skill graph in which narrowly scoped skills are linked by explicit dependencies, providing a scaffold for efficient learning and selective adaptation.

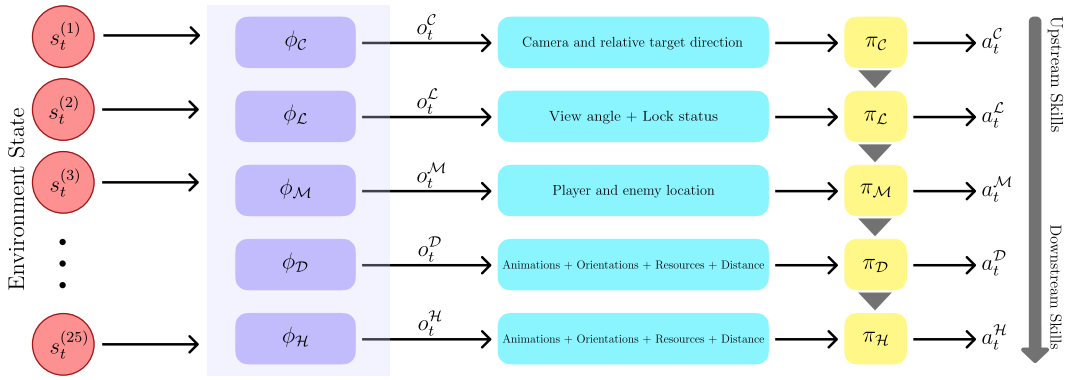


Figure 1: **The Modular Skill Graph Architecture.** At run-time, the global environment state (with dimension 25 in our experiments) is mapped into five specialized observation spaces (o_t^k), which are processed concurrently by independent skill policies (π_k) to generate action components. The grey arrows between policies denote the hierarchical training dependency: upstream skills (top) are trained first and held fixed, shaping the data distribution for the adaptive downstream skills (bottom).

In this work, this hypothesis is investigated in Dark Souls III, a challenging real-time combat environment. Recent work (Schuck, 2023; Towers et al., 2024) has begun to establish Dark Souls III as an RL research platform, often via memory instrumentation that exposes compact state signals for training. We model combat control as a directed skill graph whose nodes are five reusable skills: camera control, target lock-on, movement, dodging, and a heal–attack decision policy. The components are trained in a hierarchical curriculum aligned with the dependency structure. Skills are trained sequentially for sample efficiency but executed concurrently at run time, enabling upstream skills to remain reusable while downstream skills specialize.

A core motivation in lifelong learning is selective post-training under domain shift: when the environment changes, only the most sensitive components should be adapted while others remain stable and transferable. This aligns with rapid adaptation via fine-tuning (Finn et al., 2017) and with continual RL methods emphasizing forward transfer and reuse (Wołczyk et al., 2022; Wang et al., 2020). Selective adaptation is evaluated by splitting a boss encounter into Phase 1 and Phase 2 domains and measuring both zero-shot transfer and targeted fine-tuning. Empirically, downstream ablations sharply degrade performance while upstream skills retain utility across domains, and fine-tuning a small subset of skills rapidly recovers performance under a limited interaction budget.

Contributions. (i) We formulate Dark Souls III combat as a directed skill graph and instantiate a modular agent with five reusable skills. (ii) We propose a hierarchical training protocol that improves sample efficiency by isolating narrow competencies and reusing previously learned skills. (iii) We demonstrate selective post-training under a Phase 1 to Phase 2 domain shift, supported by ablations that characterize which skills transfer and which require adaptation.

2 Skill Graph and Hierarchical Training

Our design targets two constraints typical of real-time game control: (i) decision-making comprises heterogeneous subproblems (view control, targeting, positioning, defense, and resource/action selection), and (ii) post-training updates should be localized under domain shift. Accordingly, a modular controller is used in which each component has a narrow responsibility and a small action set, reducing entanglement and enabling selective adaptation.

Skill Set. We implement five skills, denoted by $\mathcal{C}$ (camera control), $\mathcal{L}$ (lock-on), $\mathcal{M}$ (movement/positioning), $\mathcal{D}$ (dodging), and $\mathcal{H}$ (heal–attack decisions), with corresponding policies $\pi_{\mathcal{C}}, \pi_{\mathcal{L}}, \pi_{\mathcal{M}}, \pi_{\mathcal{D}}$, and $\pi_{\mathcal{H}}$. Each skill $k \in \{\mathcal{C}, \mathcal{L}, \mathcal{M}, \mathcal{D}, \mathcal{H}\}$ receives its own observation $o_t^k = \phi_k(s_t)$, where s_t is the available game state and ϕ_k selects variables relevant to that responsibility (Figure 1). For completeness, the full specification of each skill's observation features is provided in Appendix A.

Policy Composition. At each control step, every skill policy π_k produces a low-dimensional control output a_t^k. This mirrors how humans control the game: view control, positioning, and combat decisions are issued concurrently rather than as a single atomic action. We execute the agent by composing these per-skill outputs into a single control signal,

$$a_t \;=\; \mathcal{C}\big(a_t^{\mathcal{C}}, a_t^{\mathcal{L}}, a_t^{\mathcal{M}}, a_t^{\mathcal{D}}, a_t^{\mathcal{H}}\big), \tag{1}$$

where $\mathcal{C}(\cdot)$ is a fixed composition operator. This composition is mostly straightforward because most of the skills occupy a distinct control subspace. In our implementation, skills are evaluated concurrently (multi-threaded) and the control loop applies a_t at a fixed rate by merging the most recent outputs from each skill. This yields a practical approximation to synchronous composition. More details of $\mathcal{C}$ and the per-skill action spaces are provided in Appendix A.

2.1 Curriculum Learning

Figure 1 summarizes the training dependency chain,

$$\mathcal{C} \to \mathcal{L} \to \mathcal{M} \to \mathcal{D} \to \mathcal{H}, \tag{2}$$

where $i \to j$ means π_j is trained with all upstream policies fixed. Training proceeds sequentially from left to right. When optimizing π_k, we load and freeze $\{\pi_j : j \prec k\}$, and train only π_k under the closed-loop behavior induced by the frozen upstream skills. This staging reduces effective exploration burden for later skills by constraining the reachable state distribution to task-relevant configurations.

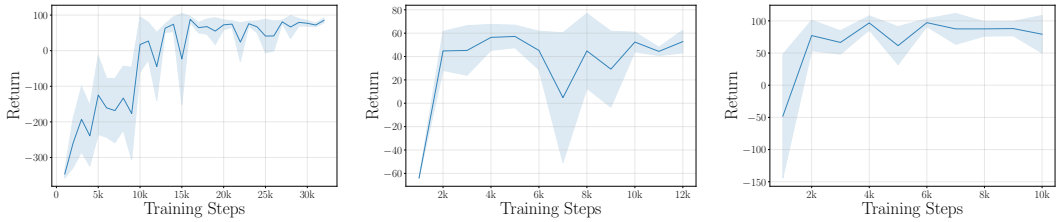
Downstream Adaptation. Because skills are trained in a staged curriculum, each downstream policy is optimized in the presence of fixed upstream behavior and therefore must adapt to it. Upstream skills also reshape the downstream learning problem by constraining reachable states and shifting the visitation distribution. For instance, a competent $\mathcal{C}$ and $\mathcal{L}$ stabilize viewpoint and targeting, and $\mathcal{M}$ biases encounters toward meaningful engagement configurations, which reduces the exploration burden for $\mathcal{D}$ and $\mathcal{H}$ and typically accelerates training.

Cooperation. Upstream skills define constraints that downstream skills should respect. For example, if $\mathcal{H}$ is trained or enabled before $\mathcal{D}$ is reliable, an aggressive $\mathcal{H}$ policy may over-commit to attacks and incur frequent hits; conversely, once $\mathcal{D}$ is competent, $\mathcal{H}$ should learn to avoid actions that systematically undermine defensive timing. Thus, the curriculum encourages *cooperative specialization*: each skill optimizes its objective while minimizing interference with established upstream competencies.

Per-skill Objectives. Each skill is trained with its own reward designed to reflect its narrow responsibility and to avoid encoding boss-specific scripts. At a high level, $\mathcal{C}$ is rewarded for improving viewpoint alignment to the target, $\mathcal{L}$ for maintaining a valid lock state, $\mathcal{M}$ for encouraging purposeful positioning and engagement rather than wandering, $\mathcal{D}$ for avoiding damage while respecting resource constraints, and $\mathcal{H}$ for trading off damage dealt, damage taken, and heal usage. The exact reward definitions used in our experiments are reported in Appendix A.3.

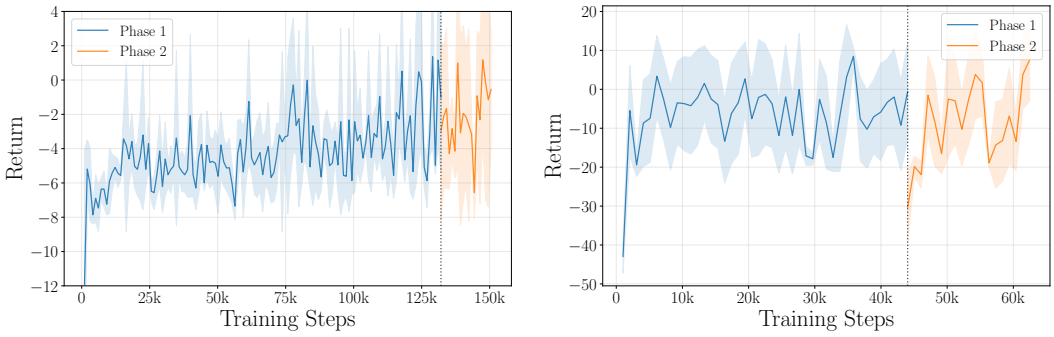
Selective Fine-tuning Under Phase Shift. A key motivation for the skill graph is localized post-training when the domain changes. We do this by transferring from the boss encounter in Phase 1 to Phase 2. Because upstream skills primarily capture phase-invariant mechanisms ($\mathcal{C}, \mathcal{L}, \mathcal{M}$), we keep $\pi_{\mathcal{C}}$, $\pi_{\mathcal{L}}$, and $\pi_{\mathcal{M}}$ fixed during Phase 2 adaptation, and fine-tune only the phase-sensitive skills ($\pi_{\mathcal{D}}$ and $\pi_{\mathcal{H}}$). Section 3 reports both zero-shot transfer and the effect of targeted fine-tuning under a limited interaction budget.

Skill Ablations. To characterize specialization and dependencies, we perform controlled ablations by replacing a selected policy π_k with a random policy over its action space while keeping all other skills unchanged. This provides a direct test of whether a skill is necessary for the composed agent's performance and whether upstream skills remain useful when downstream skills are removed. The results are reported in Section 3.



(a) Camera (b) Lock-on (c) Movement

Figure 2: Return vs. training steps for $\mathcal{C}$, $\mathcal{L}$, and $\mathcal{M}$, averaged over five evaluation episodes at each 1k-step checkpoint. The shaded region denotes the 95% confidence interval.



(a) Dodge (b) Heal–attack

Figure 3: Return vs. training steps for $\mathcal{D}$ and $\mathcal{H}$, averaged over five evaluation episodes at each 1k-step checkpoint. The shaded region denotes the 95% confidence interval, and the dashed vertical line marks the Phase 1 $\rightarrow$ Phase 2 transition.

3 Experiments

3.1 Setup

Game Interface and Observations. A process-memory state readout interface is used for Dark Souls III. Cheat Engine (Cheat Engine contributors, 2025) is used to locate the required memory variables (e.g., position, pose, resources, lock status, and animation signals). Pixel-based perception is not evaluated in this work due to compute and engineering constraints; instead, this compact state interface is used to keep the focus on skill decomposition, transfer, and selective fine-tuning. Full details of the experiments are provided in Appendix C.

Learning Algorithm. We intentionally use a simple, widely-used value-based baseline, Deep Q-Networks (DQN) (Mnih et al., 2013), for all skills. The goal is not to maximize raw performance through algorithmic sophistication, but to test whether the skill-graph factorization and selective fine-tuning are sufficient to produce transferable behavior. This is a conservative choice: vanilla DQN is known to be sensitive to exploration design, brittle under non-stationarity, and prone to interference when used in continual settings (i.e., it provides no explicit mechanism to prevent forgetting) (Pateria et al., 2021; Khetarpal et al., 2020).

3.2 Results

Sample Efficiency. Figure 2 shows that the upstream skills ($\mathcal{C}$, $\mathcal{L}$, $\mathcal{M}$) are learned quickly under the proposed curriculum, reaching near-maximal return. To contextualize sample efficiency, a single atomic end-to-end agent is trained on the same Phase 1 encounter using the same state interface and a comparable DQN-based setup. With the proposed skill graph, a competitive Phase 1 policy is obtained with an overall interaction budget of approximately 230k steps (Table 1a). In contrast, the atomic end-to-end baseline gets no where close to learning a reliable combat behavior even after

Table 1: Win rates (%) measured over 25 episodes per setting. "Random" replaces the corresponding policy with a uniform random policy over its action space, while keeping other skills fixed.

(a) Phase 1

Setting	**Win (%)**
$\pi_{\mathcal{H}}$ random, $\pi_{\mathcal{D}}$ random	0.0
$\pi_{\mathcal{H}}$ random, $\pi_{\mathcal{D}}$ trained	4.0
$\pi_{\mathcal{H}}$ trained, $\pi_{\mathcal{D}}$ random	16.0
$\pi_{\mathcal{H}}$ trained, $\pi_{\mathcal{D}}$ trained	**44.0**
Single atomic agent (end-to-end)	0.0

(b) Phase 2

Setting	**Win (%)**
Zero-shot transfer (mid-range start)	33.3
Zero-shot transfer (long-range start)	12.5
Selective fine-tuning ($\pi_{\mathcal{D}} + \pi_{\mathcal{H}}$ only)	**52.0**

plenty of steps (Table 1a). The return curve for the end-to-end baseline over training steps and further discussions are reported in Appendix B.

Downstream Skills. The downstream skills are more challenging and account for the majority of interaction. $\mathcal{D}$ requires precise timing and is the hardest component to learn. This difficulty is reflected in Figure 3a as poorly timed dodges can be harmful, making early exploration misleading and slow to improve. The gradual increase in return indicates that the agent is learning to survive longer and reduce early deaths. For interpretation, note that the maximum achievable return for $\pi_{\mathcal{D}}$ under our shaping is approximately 10, while returns near 0 correspond to "survival for a non-trivial duration" (i.e., not dying quickly), which already indicates meaningful defensive competence.

$\mathcal{H}$ Dependence on $\mathcal{D}$. The $\mathcal{H}$ policy typically reaches a reasonable attack strategy quickly, but reliable healing is hindered due to structural data sparsity: the number of healing opportunities is capped by the environment, making credit assignment particularly challenging for DQN. Under our reward scale, the maximum return for $\pi_{\mathcal{H}}$ is approximately 15. Figure 3b shows that the agent wins in fewer than half of the trials. One more thing to note is that if $\pi_{\mathcal{D}}$ is imperfect, $\pi_{\mathcal{H}}$ is incentivized to behave more aggressively, attempting to end the fight before accumulating fatal damage, which can reduce the learning signal for conservative healing.

Ablations Confirm Specialization. Table 1 shows that downstream skills are critical for overall success. With both $\pi_{\mathcal{D}}$ and $\pi_{\mathcal{H}}$ randomized, performance drops to 0%. Randomizing only $\pi_{\mathcal{D}}$ reduces win rate to 16%, while randomizing only $\pi_{\mathcal{H}}$ reduces it to 4%. The higher win rate under a randomized $\pi_{\mathcal{D}}$ reflects a shift toward aggression, where ending the fight quickly is the only viable path when defense is unreliable. With both trained, the composed policy achieves a 44% win rate in Phase 1, indicating that wins depend on competent downstream timing and decision-making.

Transfer. When transferring the Phase 1-trained system to Phase 2 without additional training, non-trivial zero-shot performance is obtained (33.3% mid-range starts; 12.5% long-range starts; Table 1b). Because the Phase 2 boss has higher health and damage but is less aggressive at close range, transfer is evaluated under different initializations: episodes are started both at mid-range and at longer range from the boss to probe how well the Phase 1 $\pi_{\mathcal{D}}$ behavior generalizes across engagement distances. Selective fine-tuning of only $\pi_{\mathcal{D}}$ and $\pi_{\mathcal{H}}$ (mid-range starts) improves the Phase 2 win rate to 52%, demonstrating that adaptation can be localized to a small subset of policies under a limited interaction budget.

4 Conclusion

We studied lifelong adaptation in a challenging real-time domain (Dark Souls III) by modeling it as a directed skill graph trained via a hierarchical curriculum. We achieved sample-efficient learning far exceeding a monolithic baseline. Crucially, this factorization enabled selective post-training: when the domain shifted, performance was recovered by fine-tuning only downstream, phase-sensitive skills under a limited interaction budget while reusing upstream behaviors. These results suggest that structuring agents around skill dependencies is a promising pathway for scalable lifelong learning in complex environments.

REFERENCES

Pierre-Luc Bacon, Jean Harb, and Doina Precup. The option-critic architecture. *Proceedings of the AAAI Conference on Artificial Intelligence*, 31, 09 2016. doi: 10.1609/aaai.v31i1.10916.

Akhil Bagaria, Jason K. Senthil, and George Dimitri Konidaris. Skill discovery for exploration and planning using deep skill graphs. In *International Conference on Machine Learning*, 2021. URL https://api.semanticscholar.org/CorpusID:221094080.

Christopher Berner, Greg Brockman, Brooke Chan, Vicki Cheung, Przemyslaw Debiak, Christy Dennison, David Farhi, Quirin Fischer, Shariq Hashme, Christopher Hesse, Rafal Józefowicz, Scott Gray, Catherine Olsson, Jakub W. Pachocki, Michael Petrov, Henrique Pondé de Oliveira Pinto, Jonathan Raiman, Tim Salimans, Jeremy Schlatter, Jonas Schneider, Szymon Sidor, Ilya Sutskever, Jie Tang, Filip Wolski, and Susan Zhang. Dota 2 with large scale deep reinforcement learning. *ArXiv*, abs/1912.06680, 2019. URL https://api.semanticscholar.org/CorpusID:209376771.

Cheat Engine contributors. Cheat engine. https://www.cheatengine.org/downloads.php, February 2025. Version 7.6 (public release Feb 12, 2025). Accessed 2026-01-24.

Petra Csereoka, Bogdan-Ionuţ Roman, Mihai Victor Micea, and Călin-Adrian Popa. Novel reinforcement learning research platform for role-playing games. *Mathematics*, 10(22), 2022. ISSN 2227-7390. doi: 10.3390/math10224363. URL https://www.mdpi.com/2227-7390/10/22/4363.

Chelsea Finn, P. Abbeel, and Sergey Levine. Model-agnostic meta-learning for fast adaptation of deep networks. In *International Conference on Machine Learning*, 2017. URL https://api.semanticscholar.org/CorpusID:6719686.

Raia Hadsell, Dushyant Rao, Andrei A. Rusu, and Razvan Pascanu. Embracing change: Continual learning in deep neural networks. *Trends in Cognitive Sciences*, 24:1028–1040, 2020. URL https://api.semanticscholar.org/CorpusID:226240885.

Khimya Khetarpal, Matthew Riemer, Irina Rish, and Doina Precup. Towards continual reinforcement learning: A review and perspectives. *J. Artif. Intell. Res.*, 75:1401–1476, 2020. URL https://api.semanticscholar.org/CorpusID:229679944.

Seungjae Lee, Jigang Kim, and Inkyu Jang. Dhrl: A graph-based approach for long-horizon and sparse hierarchical reinforcement learning, 10 2022.

Jorge Armando Mendez Mendez, Harm van Seijen, and Eric Eaton. Modular lifelong reinforcement learning via neural composition. *ArXiv*, abs/2207.00429, 2022. URL https://api.semanticscholar.org/CorpusID:247656619.

Volodymyr Mnih, Koray Kavukcuoglu, David Silver, Alex Graves, Ioannis Antonoglou, Daan Wierstra, and Martin Riedmiller. Playing atari with deep reinforcement learning, 2013. URL https://arxiv.org/abs/1312.5602.

Sanmit Narvekar, Bei Peng, Matteo Leonetti, Jivko Sinapov, Matthew E. Taylor, and Peter Stone. Curriculum learning for reinforcement learning domains: A framework and survey. *ArXiv*, abs/2003.04960, 2020. URL https://api.semanticscholar.org/CorpusID:212657666.

Jim O'Connor, Gary B. Parker, and Mustafa Bugti. Learning dark souls combat through pixel input with neuroevolution, 2025. URL https://arxiv.org/abs/2507.03793.

Shubham Pateria, Budhitama Subagdja, Ah-Hwee Tan, and Chai Quek. Hierarchical reinforcement learning: A comprehensive survey. *ACM Computing Surveys*, 54:1–35, 06 2021. doi: 10.1145/3453160.

Martin Schuck. Soulsgym: Gymnasium extension for dark souls iii, elden ring, and other souls games. https://pypi.org/project/soulsgym/, December 2023. Version 1.1.0. Released Dec 6, 2023. Accessed 2026-01-24.

Richard S. Sutton, Doina Precup, and Satinder Singh. Between mdps and semi-mdps: A framework for temporal abstraction in reinforcement learning. *Artificial Intelligence*, 112(1):181–211, 1999. ISSN 0004-3702. doi: https://doi.org/10.1016/S0004-3702(99)00052-1. URL https://www.sciencedirect.com/science/article/pii/S0004370299000521.

Mark Towers, Ariel Kwiatkowski, Jordan Terry, John U. Balis, Gianluca De Cola, Tristan Deleu, Manuel Goulão, Andreas Kallinteris, Markus Krimmel, Arjun KG, Rodrigo Perez-Vicente, Andrea Pierré, Sander Schulhoff, Jun Jet Tai, Hannah Tan, and Omar G. Younis. Gymnasium: A standard interface for reinforcement learning environments. *arXiv preprint arXiv:2407.17032*, 2024. doi: 10.48550/arXiv.2407.17032.

Oriol Vinyals, Igor Babuschkin, Wojciech M. Czarnecki, Michaël Mathieu, Andrew Joseph Dudzik, Junyoung Chung, David Choi, Richard Powell, Timo Ewalds, Petko Georgiev, Junhyuk Oh, Dan Horgan, Manuel Kroiss, Ivo Danihelka, Aja Huang, L. Sifre, Trevor Cai, John P. Agapiou, Max Jaderberg, Alexander Sasha Vezhnevets, Rémi Leblond, Tobias Pohlen, Valentin Dalibard, David Budden, Yury Sulsky, James Molloy, Tom Le Paine, Caglar Gulcehre, Ziyun Wang, Tobias Pfaff, Yuhuai Wu, Roman Ring, Dani Yogatama, Dario Wünsch, Katrina McKinney, Oliver Smith, Tom Schaul, Timothy P. Lillicrap, Koray Kavukcuoglu, Demis Hassabis, Chris Apps, and David Silver. Grandmaster level in starcraft ii using multi-agent reinforcement learning. *Nature*, 575:350 – 354, 2019. URL https://api.semanticscholar.org/CorpusID:204972004.

Zhi Wang, Chunlin Chen, and Daoyi Dong. Lifelong incremental reinforcement learning with online bayesian inference. *IEEE Transactions on Neural Networks and Learning Systems*, 33:4003–4016, 2020. URL https://api.semanticscholar.org/CorpusID:220831603.

Maciej Wołczyk, Michal Zajkac, Razvan Pascanu, Lukasz Kuci'nski, and Piotr Milo's. Disentangling transfer in continual reinforcement learning. *ArXiv*, abs/2209.13900, 2022. URL https://api.semanticscholar.org/CorpusID:252567887.

Ruihan Yang, Huazhe Xu, Yi Wu, and Xiaolong Wang. Multi-task reinforcement learning with soft modularization, 2020. URL https://arxiv.org/abs/2003.13661.

A ENVIRONMENTS

A.1 DARK SOULS III ENCOUNTER AND PHASE SPLIT

The first boss encounter in Dark Souls III, *Iudex Gundyr*, was used as the evaluation environment. The boss has a total health pool of 1037 HP, and the fight was partitioned into two phases to be analyzed as distinct domains. Phase 1 was defined over approximately 415 HP, while Phase 2 was defined over approximately 500 HP, reflecting the change in the boss move set and combat dynamics. A typical player light attack deals roughly 30–50 damage per hit.

To keep the boss loaded in the arena throughout training (avoiding the in-game defeat sequence and expensive reload/reset logic), episodes are terminated before the boss reaches zero HP. Concretely, in Phase 1 we end an episode once the boss health falls below 622. In Phase 2 we end an episode once the boss HP falls below 60. These thresholds ensure the encounter remains persistent while still providing a consistent success signal for learning and evaluation. Across both phases.

To avoid reward and value-scale issues, health variables provided to the RL algorithm were normalized by their corresponding maxima. In particular, player and boss health were represented as ratios in $[0, 1]$, which kept reward magnitudes comparable across phases and reduced sensitivity to absolute HP scaling. A base-level character instantiated with the default *Knight* class was used for all runs. Figure 4 illustrates the visual appearance of the two domains used in our experiments.

(a) Phase 1. (b) Phase 2.

Figure 4: **Phase split used as a domain shift.** Each subfigure shows a concatenation of two screenshots from the corresponding phase. The encounter was divided into Phase 1 and Phase 2 to evaluate transfer and selective fine-tuning under a controlled shift in boss behavior.

Each episode begins with the player positioned in the boss arena and ends on either boss defeat, player death, or a fixed time limit (skill-dependent in Appendix C).

A.2 OBSERVATION AND ACTION SPACES

Process-Memory Interface. A compact state interface is obtained via process-memory readout. Cheat Engine (Cheat Engine contributors, 2025) is used offline to locate the relevant variables and resolve stable relative addresses. During data collection and training, these variables are read from memory using the Python interface (`pyMeow`); Cheat Engine is not used in the training loop. The resulting interface exposes positions, orientations, lock status, resources, and animation signals required by the skill policies.

Global State and Derived Features. Let the raw memory variables at time t be grouped into a global state s_t (dimension 25 in our experiments; Figure 1). From s_t, skill-specific observations are formed as $o_t^k = \phi_k(s_t)$ via lightweight feature construction. The following derived quantities are

used repeatedly:

$$\boldsymbol{p}_t = (x_t, y_t, z_t) \in \mathbb{R}^3 \quad \text{(player position)}, \tag{3}$$

$$\boldsymbol{e}_t = (x_t^e, y_t^e, z_t^e) \in \mathbb{R}^3 \quad \text{(enemy position)}, \tag{4}$$

$$\boldsymbol{d}_t = \boldsymbol{e}_t - \boldsymbol{p}_t, \tag{5}$$

$$d_t = \|\boldsymbol{d}_t\|_2 \quad \text{(distance)}, \tag{6}$$

$$\hat{\boldsymbol{d}}_t = \boldsymbol{d}_t / \|\boldsymbol{d}_t\|_2 \in \mathbb{R}^3 \quad \text{(unit direction to enemy)}, \tag{7}$$

$$\boldsymbol{c}_t = (c_t^x, c_t^y, c_t^z) \in [-1, 1]^3 \quad \text{(camera direction vector)}, \tag{8}$$

$$\alpha_t = \arccos\left(\frac{\langle \boldsymbol{c}_t, \hat{\boldsymbol{d}}_t \rangle}{\|\boldsymbol{c}_t\|_2}\right) \in [0, \pi] \quad \text{(camera–target angle)}. \tag{9}$$

In Equation equation 9, $\alpha_t \in [0, \pi]$ is the angle between the camera direction $\boldsymbol{c}_t$ and the normalized enemy direction $\hat{\boldsymbol{d}}_t$. Enemy and player orientations are read as angles in radians, $O_t^e, O_t^p \in [-\pi, \pi]$. Health and stamina are normalized to $[0, 1]$, denoted by h_t (player HP ratio), h_t^e (enemy HP ratio), and σ_t (stamina ratio). We also show the target lock-on status with $\ell_t \in \{0, 1\}$. Animation state is represented by an integer animation identifier κ_t and a normalized progress ratio $\rho_t \in [0, 1]$, computed as $\rho_t = \frac{\texttt{anim_time}}{\texttt{max_anim_time}}$ for the currently active animation. Additionally, the remaining number of healing Estus flasks available in the episode is denoted by $n_t^{\text{estus}} \in \mathbb{N}$.

Table 2: Skill-specific observation vectors used for training each of the agents. Each skill receives $o_t^k = \phi_k(s_t)$.

Skill	**Observation** o_t^k	**Dim.**
$\mathcal{C}$	$[\hat{\boldsymbol{d}}_t^\top,\ \boldsymbol{c}_t^\top,\ \alpha_t]$	7
$\mathcal{L}$	$[\alpha_t,\ \ell_t]$	2
$\mathcal{M}$	$[\boldsymbol{p}_t^\top,\ \boldsymbol{e}_t^\top]$	6
$\mathcal{D}$	$[\kappa_t^e,\ \rho_t^e,\ O_t^e,\ O_t^p,\ \sigma_t,\ h_t,\ d_t]$	7
$\mathcal{H}$	$[\kappa_t^e,\ \rho_t^e,\ \kappa_t^p,\ \rho_t^p,\ \sigma_t,\ h_t,\ h_t^e,\ O_t^e,\ O_t^p,\ n_t^{\text{estus}},\ d_t]$	11

Table 3: Skill action spaces used for each of the agents. Actions are discrete and are applied using the keybindings of the game.

Skill	**Action set** $\mathcal{A}_k$	**Dim.**
$\mathcal{C}$	{cam up, cam down, cam left, cam right, idle}	5
$\mathcal{L}$	{toggle lock-on, idle}	2
$\mathcal{M}$	{forward, back, left, right, 4 diagonals, idle}	9
$\mathcal{D}$	{dodge, idle}	2
$\mathcal{H}$	{light attack, heal (consume flask), idle}	3

Skill-Specific Observations and Actions. The exact observation vectors and actions used in code are summarized in Tables 2 and 3. All observation variables not explicitly stated as normalized are treated as raw floats/integers.

A.3 Skill-Specific Reward Objectives

Each skill is trained with its own reward r_t^k that targets a narrow responsibility and is computed from the same process-memory interface used for observations. The shaping terms depend only on generic combat variables (e.g., angles, distances, HP/stamina ratios, lock status, remaining flasks, and terminal events) and do not encode boss-specific action sequences or phase-specific scripts. Weights are chosen so that per-episode returns fall in the scales reported in Section 3, improving stability across skills.

Camera ($\mathcal{C}$). Camera control is rewarded for reducing the camera–target angle. A dense term is used

$$r_t^{\mathcal{C}} \;=\; -\alpha_t, \tag{10}$$

optionally combined with a small bonus $\mathbb{I}[\alpha_t < \varepsilon_{\mathcal{C}}]$ when alignment is within a tolerance to encourage maintaining lockable framing. In our experiments $\varepsilon_{\mathcal{C}} = 0.6$ was used.

Lock-on ($\mathcal{L}$). Lock-on is rewarded for maintaining a valid lock state while the target is in view

$$r_t^{\mathcal{L}} \;=\; \mathbb{I}[\ell_t = 1] \;-\; \mathbb{I}[\ell_t = 0], \tag{11}$$

with lock toggling treated as a discrete action and encouraged only when it improves sustained lock status.

Movement ($\mathcal{M}$). Movement is rewarded for reaching and maintaining engagement configurations rather than wandering. A distance-based term is used

$$r_t^{\mathcal{M}} \;=\; -\lambda_{\mathcal{M}} d_t, \tag{12}$$

where $\lambda_{\mathcal{M}} = 1/10$ in our experiments. Optionally, a term $\mathbb{I}[d_t < d_{\min} \wedge \texttt{Sideways Movement}]$ can be used to encourage circling around the enemy when close by. Surprisingly, the agent learns to do so even without the optional part in a few more training timesteps.

Dodge ($\mathcal{D}$). Dodging is rewarded for survival and damage avoidance, while discouraging wasteful defensive actions. Let $\Delta h_t = h_t - h_{t-1}$ be the change in player HP ratio (negative under damage). A typical term is

$$r_t^{\mathcal{D}} \;=\; \lambda_{\text{alive}} \;+\; \lambda_{\text{dmg}} \Delta h_t \;-\; \lambda_{\text{dead}} \mathbb{I}[\texttt{Agent Dead}] \;-\; \mathbb{I}[\sigma_t < \varepsilon_{\mathcal{D}}], \tag{13}$$

where in our experiments $(\lambda_{\text{alive}}, \lambda_{\text{dmg}}, \lambda_{\text{dead}}, \varepsilon_{\mathcal{D}}) = (0.02, 5, 5, 0.05)$. The terminal penalty is applied on death. Optionally instead of $\mathbb{I}[\sigma_t < \varepsilon_{\mathcal{D}}]$ a more dense reward defined by $\Delta\sigma_t = \sigma_t - \sigma_{t-1}$ can be used to continuously signal stamina consumption. This construction makes poorly timed dodges harmful (via the induced damage term) and yields a slowly improving return curve when survival time increases.

Heal–attack ($\mathcal{H}$). The heal–attack policy is rewarded for trading off damage dealt, damage taken, and resource usage. Let $\Delta h_t^e = h_t^e - h_{t-1}^e$ denote enemy HP change (negative when damage is dealt), and let $\Delta n_t^{\text{estus}}$ denote the change in remaining flasks (negative when consumed). The objective is

$$r_t^{\mathcal{H}} \;=\; \lambda_{\text{dmg}} \Delta h_t \;-\; \lambda_{\text{hit}} \Delta h_t^e \;-\; \lambda_{\text{dead}} \mathbb{I}[\texttt{Agent Dead}] \;+\; \lambda_{\text{success}} \mathbb{I}[\texttt{Enemy Dead}], \tag{14}$$

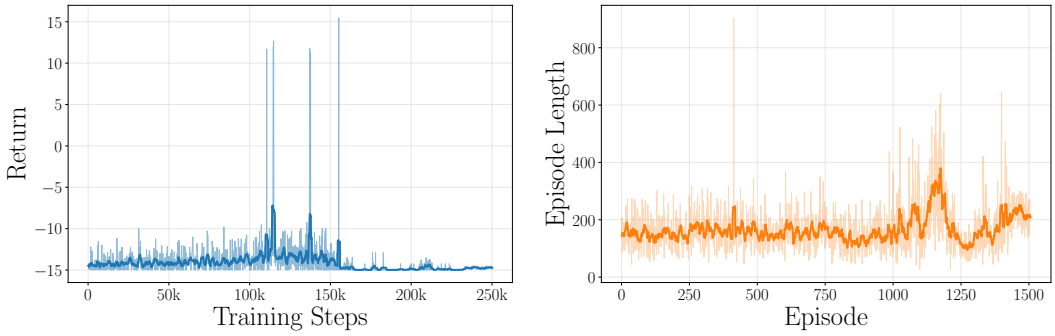
where in our experiments we used $(\lambda_{\text{dmg}}, \lambda_{\text{hit}}, \lambda_{\text{dead}}, \lambda_{\text{success}}) = (5, 15, 5, 5)$. To penalize resource consumption we can use the same method as in $\mathcal{D}$. Either of $\mathbb{I}[\sigma_t < \varepsilon_{\mathcal{H}}]$ or $\Delta\sigma_t$ can be used to penalize overcommitting. Note that healing is rewarded using the term Δh_t which encourages healing when health is low. Because the number of flasks per episode is capped (one in Phase 1 and two in Phase 2), transitions containing heal events remain sparse, which makes credit assignment for heal timing significantly harder than learning an attack policy.

B End-to-End Baseline

To contextualize the benefits of the directed skill-graph factorization, we train an atomic end-to-end baseline: a single DQN policy that directly controls camera, lock-on, locomotion, dodging, and attack/heal decisions through one discrete action space. The baseline uses the same process-memory state interface and training setup as the modular agent. The only difference is that control is learned with a single monolithic policy.

The end-to-end agent observes the full global state s_t (25 dimensions), i.e., the union of the features used across the skill-specific policies. Concretely, we use

$$o_t^{\text{e2e}} = s_t = [\kappa_t^e,\ \rho_t^e,\ \kappa_t^p,\ \rho_t^p,\ \sigma_t,\ h_t,\ h_t^e,\ O_t^e,\ O_t^p,\ n_t^{\text{estus}},\ d_t,\ \hat{\boldsymbol{d}}_t^\top,\ \boldsymbol{c}_t^\top, \boldsymbol{p}_t^\top,\ \boldsymbol{e}_t^\top,\ \alpha_t,\ \ell_t]. \tag{15}$$



(a) Training return.

(b) Episode length.

Figure 5: End-to-end baseline struggles under the same state interface. The DQN baseline quickly plateaus, exhibiting little change in return or survival time beyond early training. The episode length plot shows that the agent couldn't even postpone early deaths even with its conservative strategy. The translucent line shows the raw value, and the darker line shows the 10-period moving average.

The baseline uses a 16-action discrete set that merges the skill action subspaces: (i) movement (forward/back/left/right), (ii) movement+dodge in each cardinal direction, (iii) camera adjustments (up/down/left/right), (iv) toggle lock-on, (v) light attack, (vi) heal (consume flask), and (vii) idle.

To avoid misleading the end-to-end policy with a mixture of skill-specific shaping signals, we train the end-to-end baseline using the same reward as the heal–attack objective (Eq. 14). This keeps the supervision consistent with the overall combat objective and avoids injecting additional dense signals that are specific to individual skills.

Figure 5 shows the training curves for return and episode length. Although we initially planned to train for 500k steps, the baseline plateaued early (already by $\sim$250k steps) with no meaningful improvement, so we stopped training due to the high wall-clock cost. Qualitatively, the learned behavior collapses into a poor survival heuristic: the agent typically locks on and repeatedly dodges backward to avoid immediate death, without learning a reliable dodging policy (timing/direction) or an effective attack strategy.

C Experimental Details

Implementation and Algorithm. All skills are trained with Deep Q-Networks (DQN) using Stable-Baselines3. Unless otherwise noted, we use the default SB3 DQN hyperparameters and a single environment instance (no parallel rollouts). Observations are normalized using `VecNormalize` and rewards are not normalized.

We use skill-specific maximum episode lengths to match the temporal scope of each subtask: 128 steps for $\mathcal{C}$ and $\mathcal{M}$, 64 for $\mathcal{L}$, 512 for $\mathcal{D}$, 1024 for $\mathcal{H}$, and 2048 for end-to-end baseline. For $\mathcal{D}$, $\mathcal{H}$, and end-to-end agents the episode also terminates upon death or success. Episodes also terminate on standard environment termination conditions (e.g., agent death or boss defeat).

Hyperparameters. Across all skills, we use a learning rate of 3×10^{-4} and batch size of 256. For the modular skills, replay buffer capacity is set to approximately one-third of the corresponding training interaction budget. For the end-to-end baseline, we use a replay buffer size of 100,000 transitions. Other optimizer and DQN parameters follow the SB3 defaults (including the exploration schedule and target network update settings).

Initialization and State Randomization. For the upstream skills ($\mathcal{C}$, $\mathcal{L}$, $\mathcal{M}$), episodes are initialized by placing the player at random locations within the arena to diversify camera/targeting/locomotion configurations and avoid overfitting to a single spawn state.

To avoid full game restarts during training, in-game death handling is disabled. Instead, we treat the agent as "dead" and terminate the episode when the normalized player HP falls below $h_t < 0.05$.

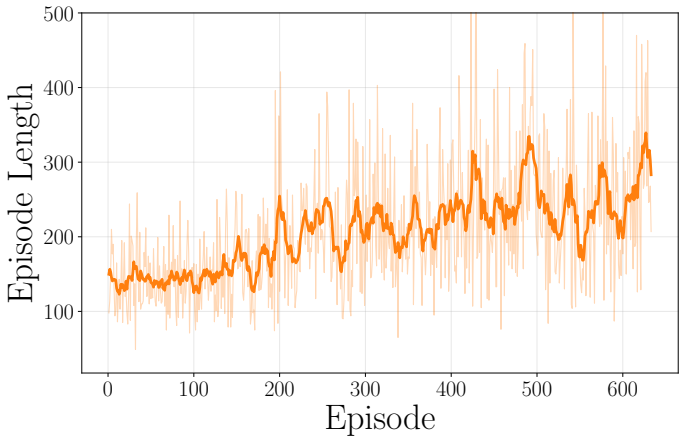


Figure 6: Episode length of $\mathcal{D}$ agent vs. training episodes.

Episodes also terminate on boss defeat and are truncated at the skill-specific horizon limits reported above.

Action Execution Timing. Because the environment is controlled through key-press macros, each discrete action is implemented with a fixed real-time delay to allow the in-game animation/transition to register before the next observation. For $\mathcal{C}$, $\mathcal{L}$, and $\mathcal{M}$, each action step uses a 0.1s delay. For $\mathcal{D}$, a dodge action occupies 0.5s. In $\mathcal{H}$, light attack uses 0.2s, drinking Estus uses 0.3s, and attempting to drink with zero flasks uses 0.5s (a no-op delay). The idle action uses 0.1s.

D Additional Results

D.1 $\mathcal{D}$ Analysis

For skills whose reward is strongly correlated with survival (notably $\mathcal{D}$), episode length provides an interpretable diagnostic: longer episodes typically indicate fewer early deaths and improved defensive timing. We report episode length curves to make learning progress easier to interpret.

Figure 6 shows that the mean episode length increases from roughly 150 to 300 steps over training, indicating that even without access to healing (no Estus use during $\mathcal{D}$ training), the learned policy survives about $2\times$ longer than a random-dodge baseline, consistent with a competent defensive timing strategy.