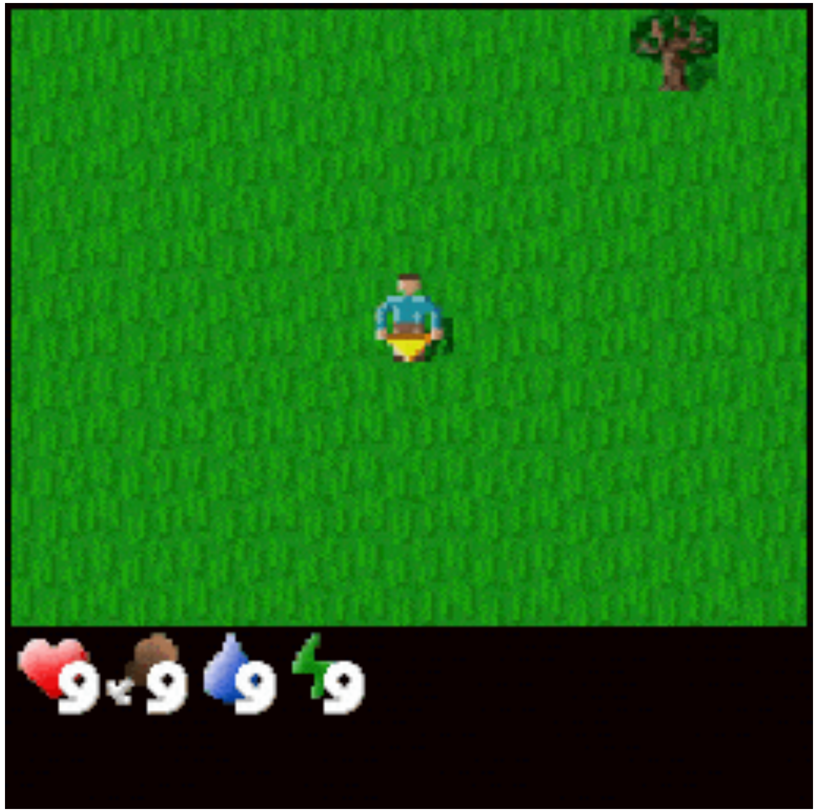
Curiosity-driven exploration for PPO agents in sparse reward environments



[1]

Artur Ganzha 10019651

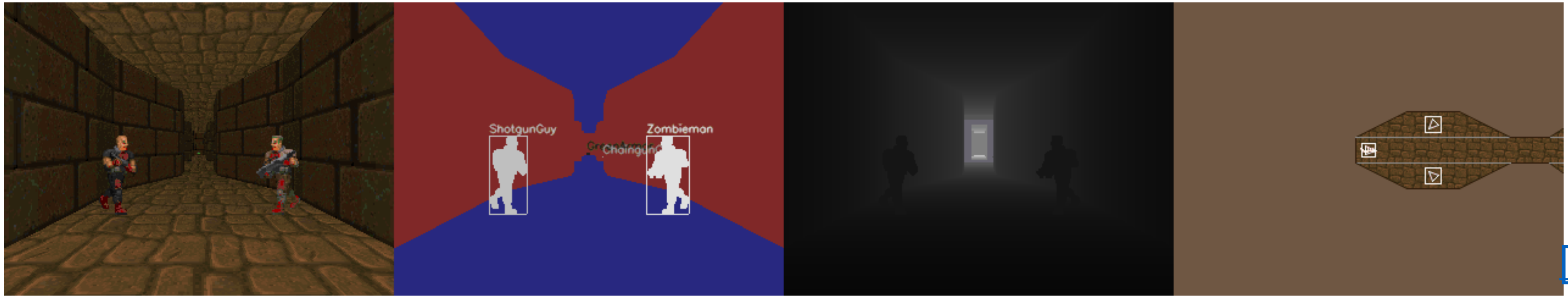
Motivation

- Learning an agent for low-dimensional state spaces has become a trivial task
- Many environments are openly available with OpenAIs Gym
- Proximal Policy Optimization is one of the most popular methods [5]
  - due to stability and performance and versatility
  - Still has limitations in sparse reward environments and deep exploration tasks
- To enhance the agent to explore and discover novel states, intrinsic rewards could offer a solution
  - New or distant rewards could be discovered which can lead to faster learning
  - Faster learning can lead to better overall performance with the same interaction budget
- Random Network Distillation offers the potential enhancement for exploration behavior
- Crafter is a suitable environment to test this method, because of its complexity and deep exploration challenges

Related Work

[2]

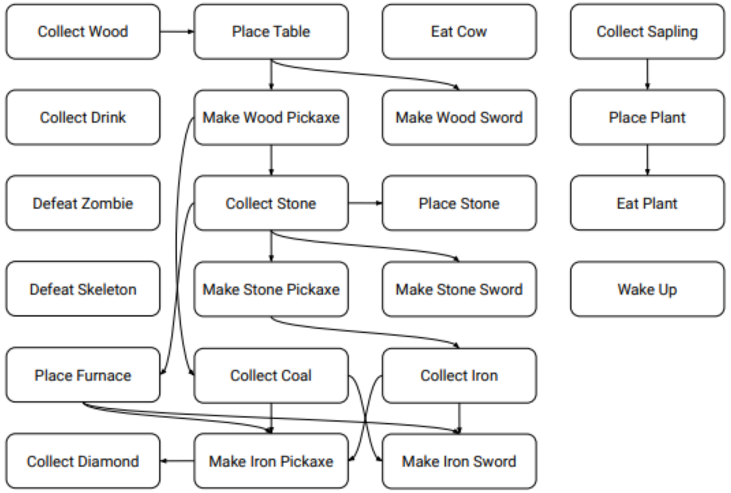
- Crafter with roguelike elements: in Jax Craftax [2]
- VizDoom: Open-Source adaptation of the original Doom for AI research [3]
  - Which was could be solved by enhancing exploration
- PPO - RND for low-complexity state space games [6]



[3]

Crafter Environment

- 2D abstraction of Minecraft
- 17 possible actions (e.g., move (W, A, S, D))
- 22 Achievements ~ 22 max score
- Map gets generated (64,64,3)
- 4 levels of survival (HP, food, water, etc.)
- Needs to collect resources for crafting
- Creatures can attack at night
- -0.1 reward for damage, most episodes end on .1 because the agent died
- Episodes ends after 10.000 steps



[1]

PPO

- Improves the decision-making by optimizing an agents policy through interactions [5]
- Adjusts the agents policy based on the rewards that are accumulated during episodes
- Clipping - penalty for policy updates that diverge highly compared to the old policy
    - The probability ratio $r_t(\theta)$ between the updates is limited with $1 - \epsilon, 1 + \epsilon$
    - Idea comes from Kullback-Leibler divergence which is used is PPOs predecessor TRPO [7].
- Entropy term balances exploration behavior
- Advantage function A_t provides the probability of which action should be taken by subtracting the Q-value with the output of the value-function

$$L^{\text{CLIP}}(\theta) = \mathbb{E}_t[\min(r_t(\theta)A_t, \text{clip}(r_t(\theta), 1 - \epsilon, 1 + \epsilon)A_t)]$$

RND

- Starts from randomly initialized untrained neural network
- Over time distill it into a trained network - Network is reduced to a smaller size
- Initially trained on Montezumas Revenge which requires the agent to work towards late rewards with collected keys
    - Which aligns with Crafters structure
- Generates intrinsic rewards that are based on the prediction error
    - Uses fixed target network and a predictor network to calculate error
    - High prediction error: Agent does not know the state → high intrinsic reward for state novelty
- Learning process: intrinsic rewards + reward structure from the environment
- Useful for sparse reward environments and efficient exploration

Approach: PPO with RND

- PPO uses Generalized Advantage Estimation (GAE)
  - Controls bias-variance trade-off and uses TD-error
  - In the project the GAE is calculated once for external and intrinsic rewards
  - Both results are summed and weighted with fix coefficients in favour of the extrinsic reward

$$A_t^{\text{GAE}} = \sum_{t=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}$$

$$A_{internal} = \sum_{t=0}^{\infty} \left(r_t^{\text{int}} + \gamma V_{\text{int}}(s_{t+1}) - V_{\text{int}}(s_t) \right)$$

$$A_{external} = \sum_{t=0}^{\infty} \left(r_t^{\text{ext}} + \gamma V(s_{t+1}) - V(s_t) \right)$$

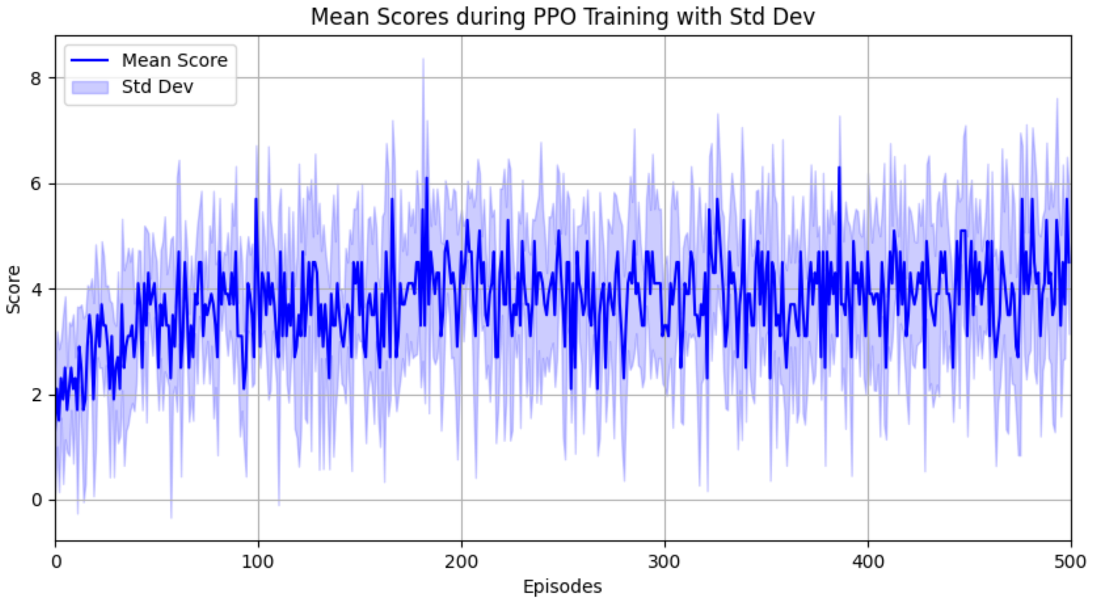
$$A_t = \alpha * A_{external} + \beta * A_{internal}$$

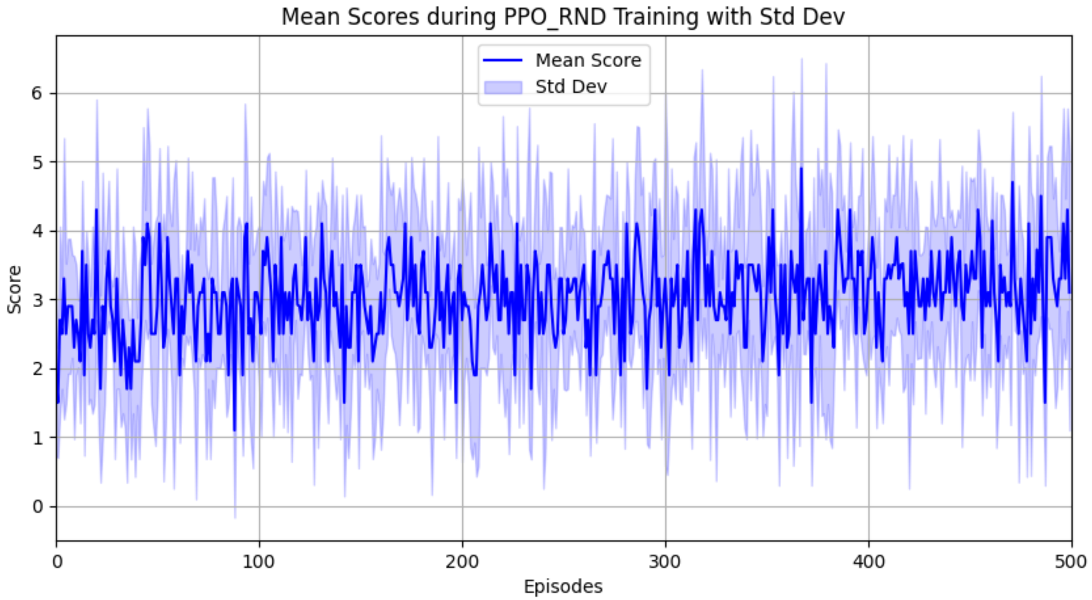
Approach: Algorithm flow

1. Collect experience from environment
2. Compute extrinsic rewards (milestones)
3. Compute intrinsic rewards (prediction error with unvisited states)
   - RND generates intrinsic rewards based on unvisited states
     - Aimed to balance task completion and exploration
4. Combine both rewards and update PPO policy
   - With modified Advantage calculation
5. Update RND network periodically for better exploration behavior
   - Based on novelty

Experiments and Results

Experiment Setup: 500 episodes, 5 seeds





PPO: After approx. 50 episodes the agent reaches the total reward of 4 and has a robust overall training performance

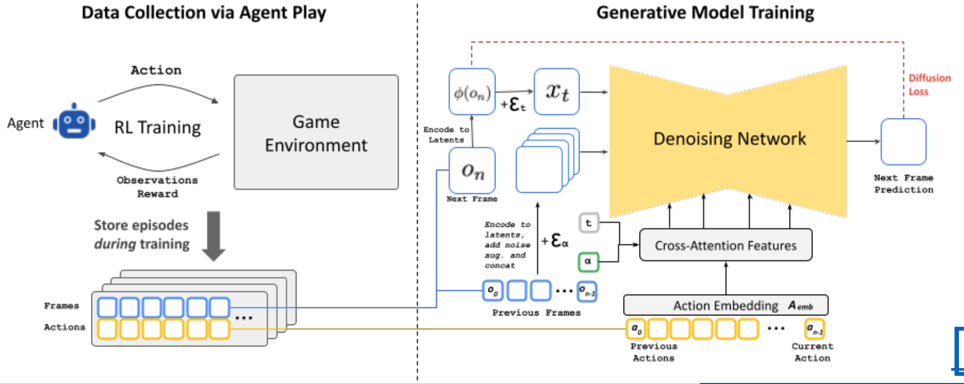
PPO + RND: Agent reaches score of 4 faster, but can not keep the performance, hence it is a less robust performance and has a higher variance in achieved milestones.

Discussion

- PPO has overall the better performance, due to robustness and mean score
- PPO + RND shows that intrinsic rewards can lead to faster rewards
  - But instead of exploiting that, the agent explores in the following episodes
  - The configuration of the hyperparameters has a major impact on the result, but some lack in stability
- **Possible improvements:**
  - Find a better configuration for hyperparameters
  - Re-balance the weights of each advantage
  - Use RND with a time limit (set coefficient to zero after n-steps)

Future Work

- Inspect each milestone separately to see how each method impacts
- It can be tested on similar games like Craftax [2], which has less sparse rewards and a rogue-like game structure
  - Which makes exploration more impactful
- If enough compute is available, test both methods on Minecraft
- Test each method in GameNGen [8] to see impact of exploration on real-time generated game content to measure playability
  - Is PPO with RND a better benchmark for human replay then PPO?



[8]

Questions?

References

- [1] Hafner, D. (2021). Benchmarking the Spectrum of Agent Capabilities. arXiv preprint:2109.06780
- [2] Matthews, M. et al. (2024). Craftax: A Lightning-Fast Benchmark for Open-End Reinforcement Learning. International Conference on Machine Learning (ICML).
- [3] Marek Wydmuch, Michał Kempka, and Wojciech Jas´kowski. ViZDoom Competitions: Playing Doom from Pixels. *IEEE Transactions on Games*, 11(3):248–259, 2019. doi: 10.1109/TG.2018.2877047. The 2022 IEEE Transactions on Games Outstanding Paper Award.
- [4] Bradbury, J. et al (2018). JAX: composable transformations of {P}ython+{N}um{P}y programs. http://github.com/google/jax.
- [5] John Schulman, Filip Wolski, Prafulla Dhariwal, Alec Radford, and Oleg Klimov. Proximal policy optimization algorithms. arXiv preprint arXiv:1707.06347, 2017.
- [6] Nugroho, W. (2021). Reinforcement Learning PPO RND [Source code]. GitHub. https://github.com/wisnunugroho21/reinforcement_learning_ppo_rnd
- [7] Schulman et al. (2015). Trust Region Policy Optimization. arXiv:1502:05477.
- [8] Dani Valevski, Yaniv Leviathan, Moab Arar, and Shlomi Fruchter. Diffusion models are real-time game engines, 2024. URL https://arxiv.org/abs/2408.14837.