2048 Deep Reinforcement Learning

Deep Learning Final Project

Tej Shah, Gloria Liu, Max Calero

Outline

I. 2048 Game Introduction
II. Simulation Results: Random, DDQN, and PPO Agents
III. Background I: Value Estimation, DQN, DDQN
IV. Background II: Policy Gradient, TRPO, PPO, GAEs
V. State Representation to Model
VI. 2048 DDQN Implementation Details
VII. 2048 PPO Implementation Details
VIII. Conclusion & Discussion of Results

I. Game Introduction

2048 Game Introduction

Motivation:

- Test Double Deep Q-Network (DDQN) and Proximal Policy Optimization (PPO)
- Game that is simple to understand, yet has a large state space & element of randomness

2048 Game Introduction

Move tiles left, right, up, and down

Current Score: 11284.0

Max Number: 1024.0

1024.0	256.0	32.0	16.0
8.0	8.0	64.0	0.0
2.0	2.0	0.0	0.0
0.0	0.0	4.0	0.0

Last Move: up

Current Score: 11304.0

Max Number: 1024.0

1024.0	256.0	32.0	16.0
0.0	0.0	16.0	64.0
0.0	2.0	0.0	4.0
0.0	0.0	0.0	4.0

Last Move: right

2048 Game Introduction

Current Score: 668.0

Max Number: 64.0

16.0	2.0	8.0	2.0
8.0	16.0	64.0	16.0
4.0	32.0	8.0	4.0
2.0	4.0	16.0	2.0

Last Move: right

Terminal State: Board is full, no action

Current Score: 20260.0

Max Number: 2048.0

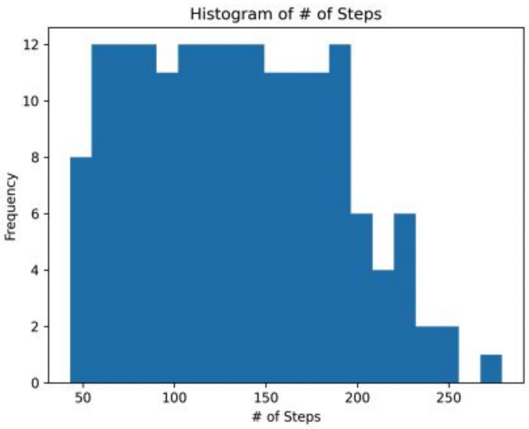
2048	0	0	2
8	16	2	0
2	4	2	0
2	0	0	0

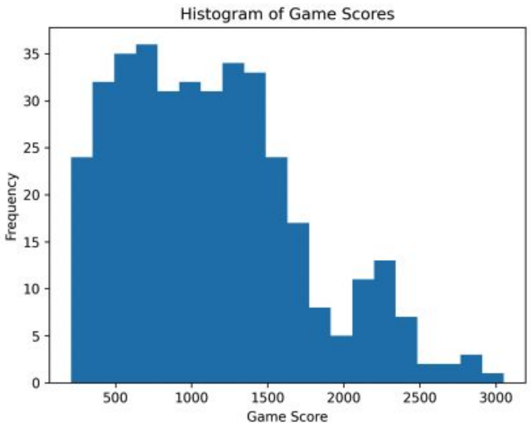
Last Move: left

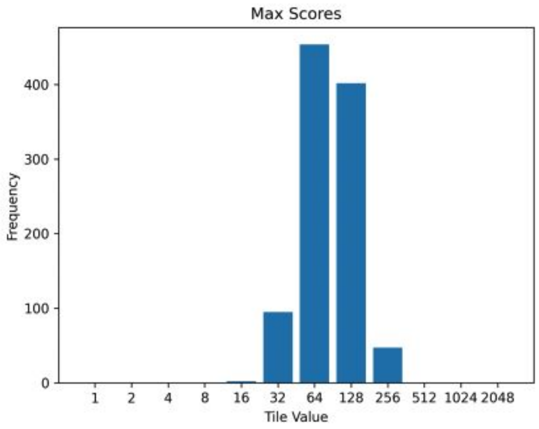
Win, can continue playing

II. Simulation Results - Random, DDQN, PPO Agents

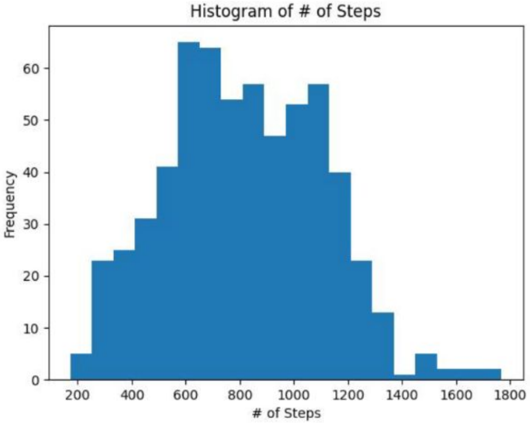
Simulation Results: Random Agent

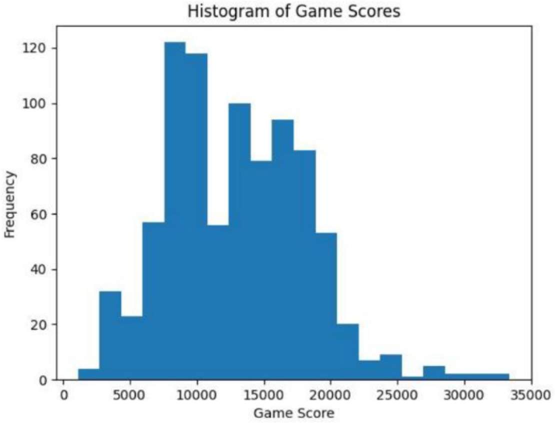


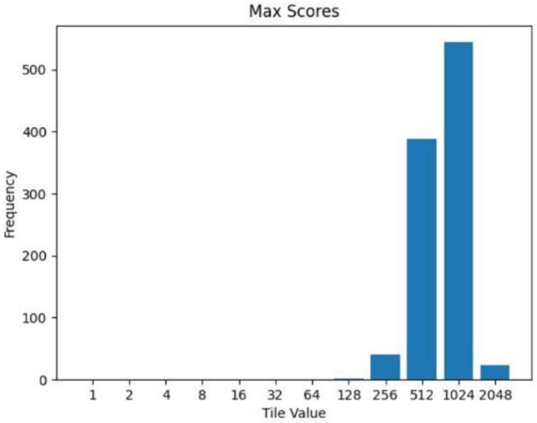




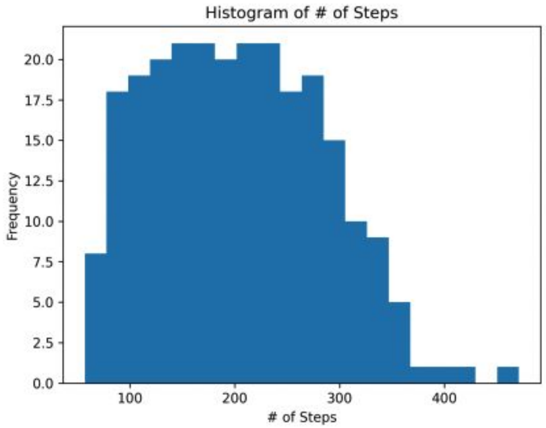
Simulation Results: DDQN

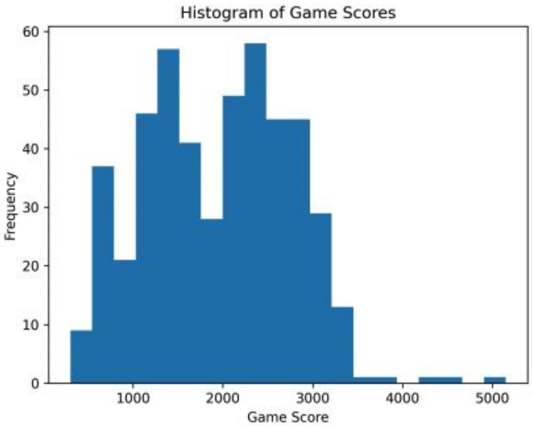


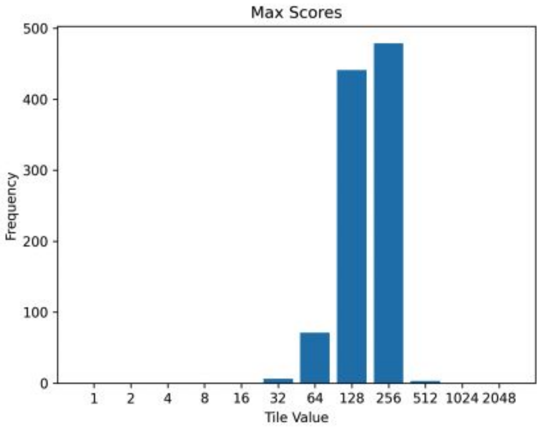




Simulation Results: PPO







III. Background I: Value Estimation, DQN, DDQN

Bellman Equation

$$V^*(s) = \max_a \sum_{s', r} p(s', r | s, a)[r + \gamma V(s')],$$

$$Q^*(s, a) = \sum_{s', r} p(s', r | s, a)[r + \gamma \max_{a'} Q(s', a')],$$

Temporal difference

$$\delta_t = \overbrace{\boxed{r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta^-)}}^{\mathrm{Y}^{\mathrm{DDQN}}} - Q(s, a; \theta)$$

DDQN Loss

Compute the Q-values for the actions of the next state using the target network.
Then get the Q-value of what the online network considers the best action.

Using the online Network the get the best action for the next state.

$$L(\theta) = \mathbb{E}_{(s,a,r,s')\sim U(D)}\left[\left(r + \gamma Q(s', \arg\max_{a'} Q(s', a'; \theta); \theta^{-}) - Q(s, a; \theta)\right)^2\right]$$

The sum of expected discounted future reward.
It contains more info due to it being a state ahead.

Calculate Q-value values and takes the Q-value of the action already taken.
The Q-Value should be the same as everything in blue, but it most likely wont be.
This means there is more to learn.

DDQN key concepts

- Q-Values are an estimation of expected reward for each action in a given state
- Temporal difference determines how much to learn from the sampled mini batch

IV. Background II: Policy Gradient, TRPO, PPO, GAEs

Policy Methods

$$\pi : \mathcal{S} \rightarrow \mathcal{A}$$

Main Goal: Update the parametrized probability distribution pi over actions based on interaction with the environment.

Vanilla Policy Gradients

Update the policy over the trajectory samples in expectation

The advantage of a particular action over the average.

$$\nabla_\theta J(\pi_\theta) = \mathbb{E}_{\tau\sim\pi_\theta}\left[\sum_{t=0}^{T} \nabla_\theta \log \pi_\theta(a_t|s_t) A^{\pi_\theta}(s_t, a_t)\right]$$

The log probability of action selection from policy.

Trust Region Policy Optimization (TRPO)

Relative “importance” ratio

“Trust region”. You don’t want to update your new policy distribution too much from the old distribution.

$$\text{maximize} \quad \mathbb{E}_t\left[\frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\text{old}}}(a_t|s_t)} A^{\pi_{\theta_{\text{old}}}}(s_t, a_t)\right] \quad \text{subject to} \quad \mathbb{E}_t\left[KL(\pi_{\theta_{\text{old}}}(s_t)||\pi_\theta(s_t))\right] \leq \delta$$

User specified bound for the trust region

The advantage of a particular action over the average.

Key Idea: “Constrain” policy updates to a “trust region.” However, it is an expensive computation practically.

Proximal Policy Optimization (PPO) KL Penalty

$$L^{\mathrm{PEN}}(\theta) = \mathbb{E}_t\left[r_t(\theta)A_t - \beta_t KL(\pi_{\theta_{\mathrm{old}}}(s_t)||\pi_\theta(s_t))\right]$$

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\mathrm{old}}}(a_t|s_t)}$$

Key Idea:
Reformulate TRPO objective in a dual form and include KL penalty in the objective function instead of looking at a constrained optimization problem

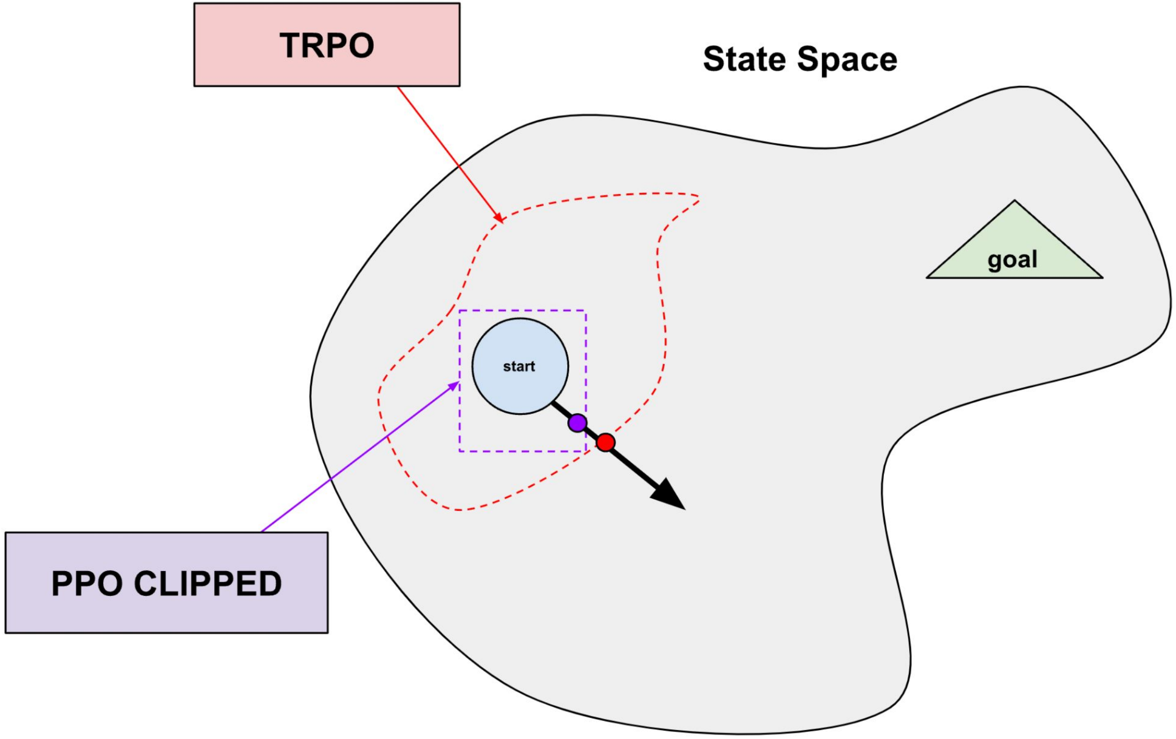
Proximal Policy Optimization (PPO) Clipped

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

$$r_t(\theta) = \frac{\pi_\theta(a_t|s_t)}{\pi_{\theta_{\mathrm{old}}}(a_t|s_t)}$$

Key Idea:
Approximate TRPO and PPO Penalty behavior with pseudo surrogate objective that is easier to compute. We will use PPO clipped for 2048.

A Conceptual Picture: Policy Gradients, TRPO, PPO

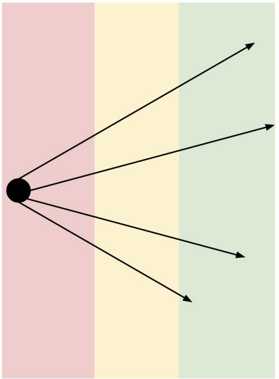


Generalized Advantage Estimation

- We want to score actions as advantageous based on some n-step lookahead, where earlier actions are prioritized more (lambda based exponential average)
- Further, we discount (gamma) future rewards for actions back to present time t

$$\hat{A}_t^{\mathrm{GAE}(\gamma,\lambda)} = \sum_{l=0}^{\infty} (\gamma\lambda)^l \delta_{t+l}^V$$

$$\delta_t^V = r_t + \gamma V(s_{t+1}) - V(s_t)$$



Increasing lambda:

- High bias
- Bias/variance tradeoff
- High variance

V. State Representation to Model

2048: State Representation (DDQN & PPO)

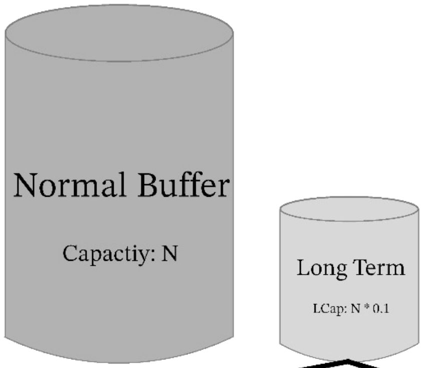
$$\boxed{s} \Leftrightarrow \underset{4 \times 4}{\begin{bmatrix}2 & 8 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 0\\0 & 0 & 0 & 4\end{bmatrix}} \Leftrightarrow \underset{16 \times 18}{\underbrace{\begin{bmatrix}0 & 1 & 0 & \cdots & 0\\0 & 0 & 0 & \cdots & 0\\\vdots & \vdots & \vdots & \ddots & \vdots\\0 & 0 & 1 & \cdots & 0\end{bmatrix}}_{2^0\ \ 2^1\ \ \cdots\ \ 2^{17}}} \Leftrightarrow \begin{bmatrix}0\\1\\\vdots\\0\end{bmatrix}_{288\times 1}$$

VI. 2048 DDQN Implementation Details

2048 DDQN: Implementation Details

Hyperparameters	**Value**
Replay Capacity	50000
Batch size	128
Epsilon start	0.9
Epsilon end	0.01
Epsilon decay	10000
Gamma	0.99
Tau	0.001
Learning Rate	1×10^{-5}
Number of Shared Layers	1
Shared Hidden Layer Size	256
Activation Function	nn.ReLU()
Gradient Clip Value	1000
Episodes	75000

U = DDQN Buffer

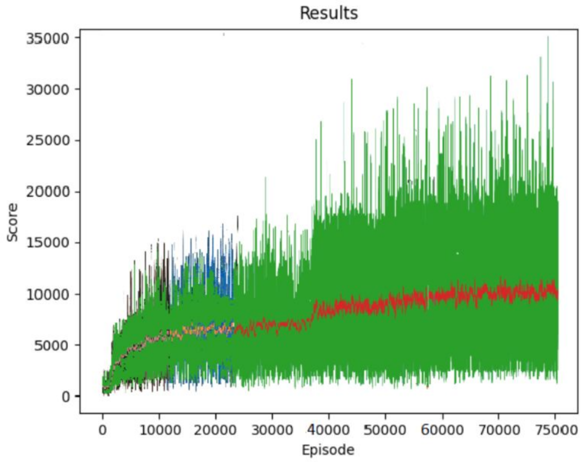


Stores tuples
u_n = (state, action, reward, next state, done)

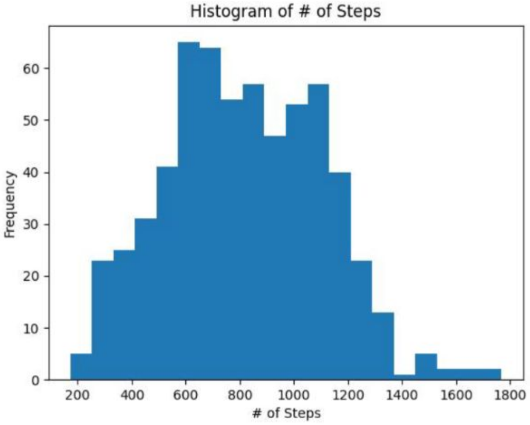
$$\text{Normal\_Buffer} = \{\overbrace{u_1, u_2, u_3, \ldots, u_{LCap * 0.1}}, \ldots, u_{N/2}, \overbrace{u_{N/2+1}, \ldots, u_{N/2 + LCap * 0.1}}, \ldots, u_N, \}$$

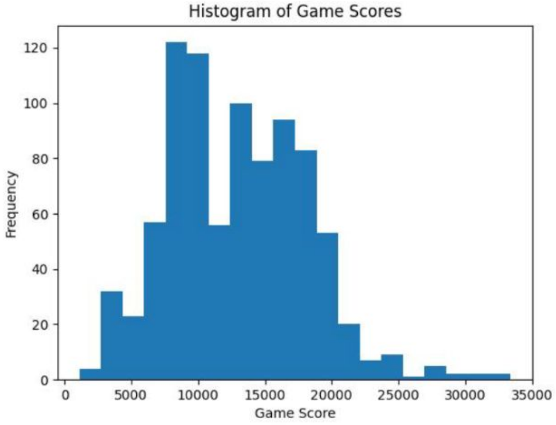
2048 DDQN: Implementation Results

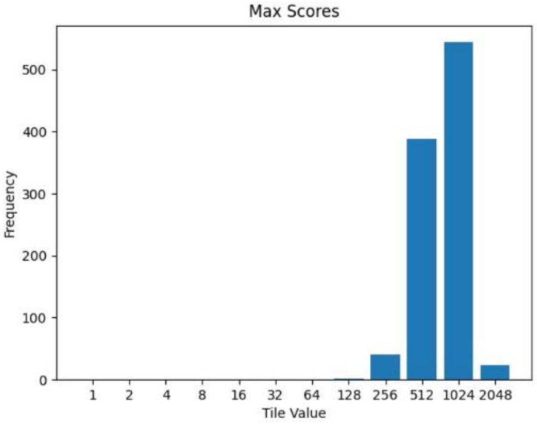
Hyperparameters	Value
Replay Capacity	50000
Batch size	128
Epsilon start	0.9
Epsilon end	0.01
Epsilon decay	10000
Gamma	0.99
Tau	0.001
Learning Rate	1×10^{-5}
Number of Shared Layers	1
Shared Hidden Layer Size	256
Activation Function	nn.ReLU()
Gradient Clip Value	1000
Episodes	75000



2048 DDQN: Results





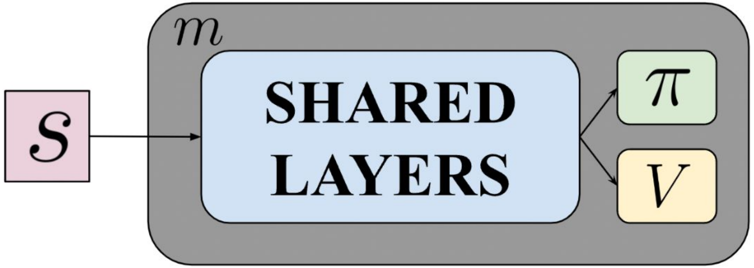


2048 DDQN Behavior

VII. 2048 PPO Implementation Details

2048 PPO: Implementation Details

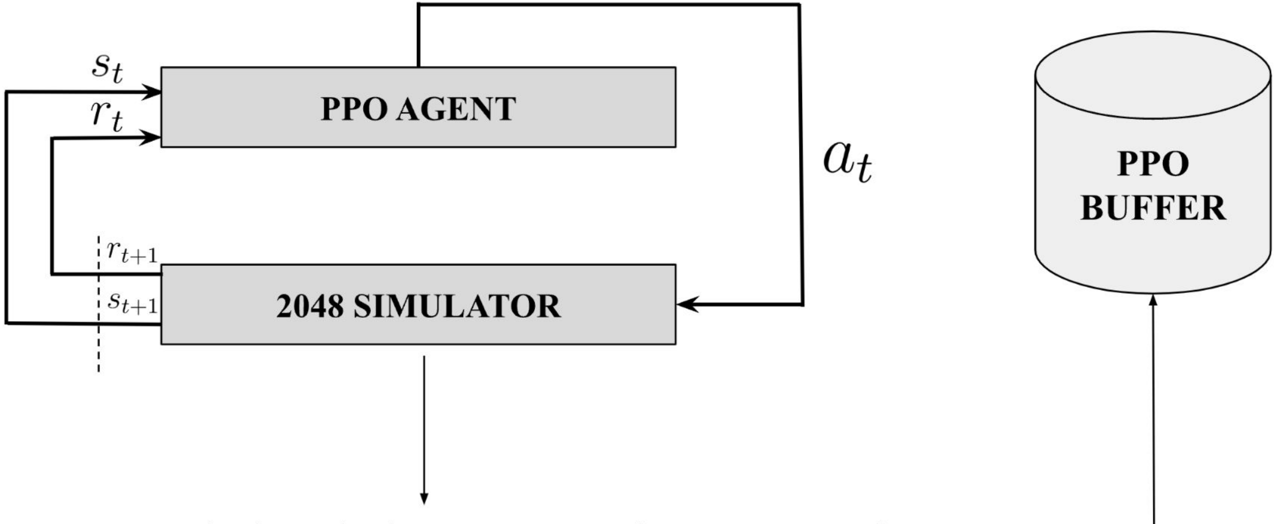
Hyperparameters	**Value**
Shared Hidden Layer Size	256
Number of Shared Layers	1
Activation Function	nn.Tanh()
PPO Clip Value	0.10
PPO Policy Learning Rate	1×10^{-5}
PPO Value Learning Rate	1×10^{-4}
PPO Epochs	60
Value Epochs	60
KL Target	0.02
Number of Rollouts	8/16/64



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

$$L^{\text{Value}} = \frac{1}{N}\sum_{i=1}^{N}(r_i - V(s_i))^2$$

2048 PPO: Implementation Details



$$\begin{bmatrix} s & a & r & \pi(s,a) & V(s,a) & \texttt{computedGAEs}(\lambda=0.95,\gamma=0.99) & \texttt{computedDiscountedRewards}(\gamma=0.99) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \\ s_i & a_i & r_i & \pi(s_i,a_i) & V(s_i,a_i) & \texttt{computedGAEs}_i(\lambda=0.95,\gamma=0.99) & \texttt{computedDiscountedRewards}_i(\gamma=0.99) \\ \vdots & \vdots & \vdots & \vdots & \vdots & \vdots & \vdots \end{bmatrix}$$

2048 PPO: Results

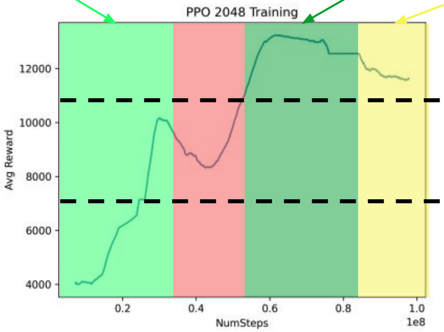
Monotonic improvement early

Changing the overall policy strategy

Converged to suboptimal policy

Changing strategy?

Hyperparameters	**Value**
Shared Hidden Layer Size	256
Number of Shared Layers	1
Activation Function	nn.Tanh()
PPO Clip Value	0.10
PPO Policy Learning Rate	1×10^{-5}
PPO Value Learning Rate	1×10^{-4}
PPO Epochs	60
Value Epochs	60
KL Target	0.02
Number of Rollouts	8/16/64



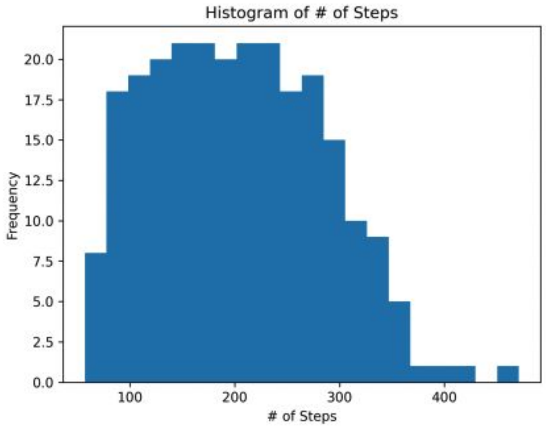
512 Max Tile

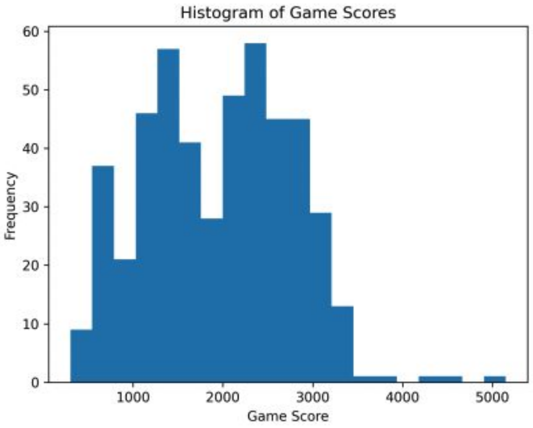
256 Max Tile

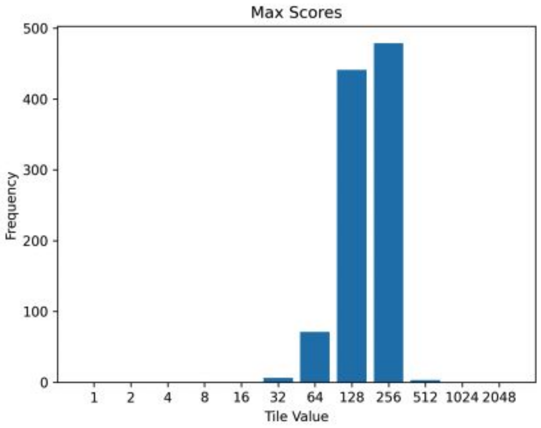
128 Max Tile

Figure 13: $r_{\text{final}} = \texttt{logMaxTile}^{\frac{1}{t}} \cdot \texttt{deltaScore}$

2048 PPO: Results







2048 PPO Behavior

Current Score: 0.0

Max Number: 2.0

0.0	0.0	0.0	0.0
0.0	0.0	0.0	0.0
2.0	0.0	0.0	0.0
0.0	2.0	0.0	0.0

Last Move: no move

VIII. Conclusion & Results Discussion

Conclusion

Results

- Both DDQN and PPO better than random - further study and experimentation with these methods
- DDQN more sample efficient than PPO for 2048

Takeaways

- Importance of reward functions and state representation
- Converging to suboptimal results