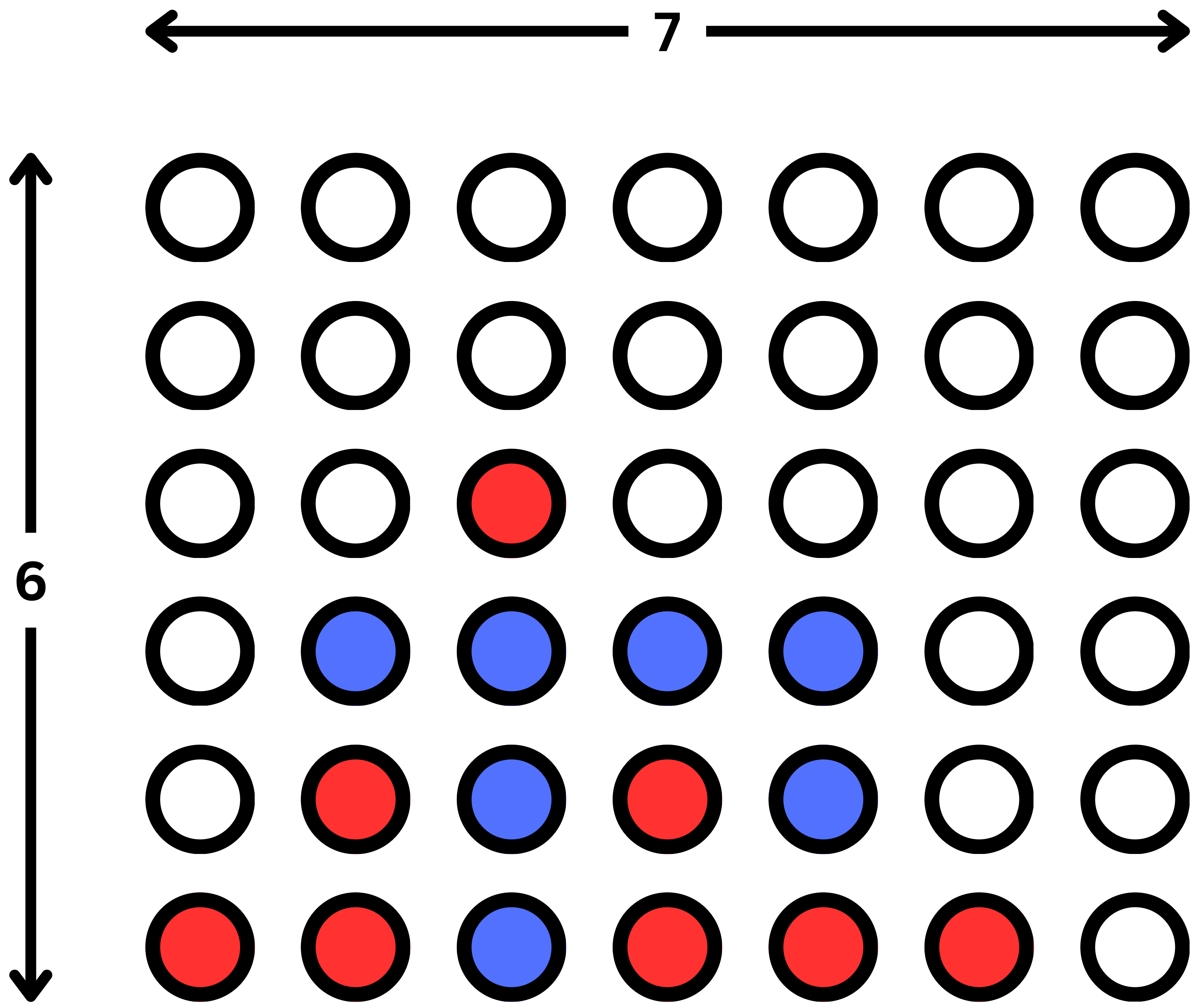
Creating a 4x4 Connect 4 Subgame Agent

Attempting a subgame evaluator approach to a full-game agent.

Aim: **Train a Connect 4-playing agent (using RL techniques)**

Connect 4
Game

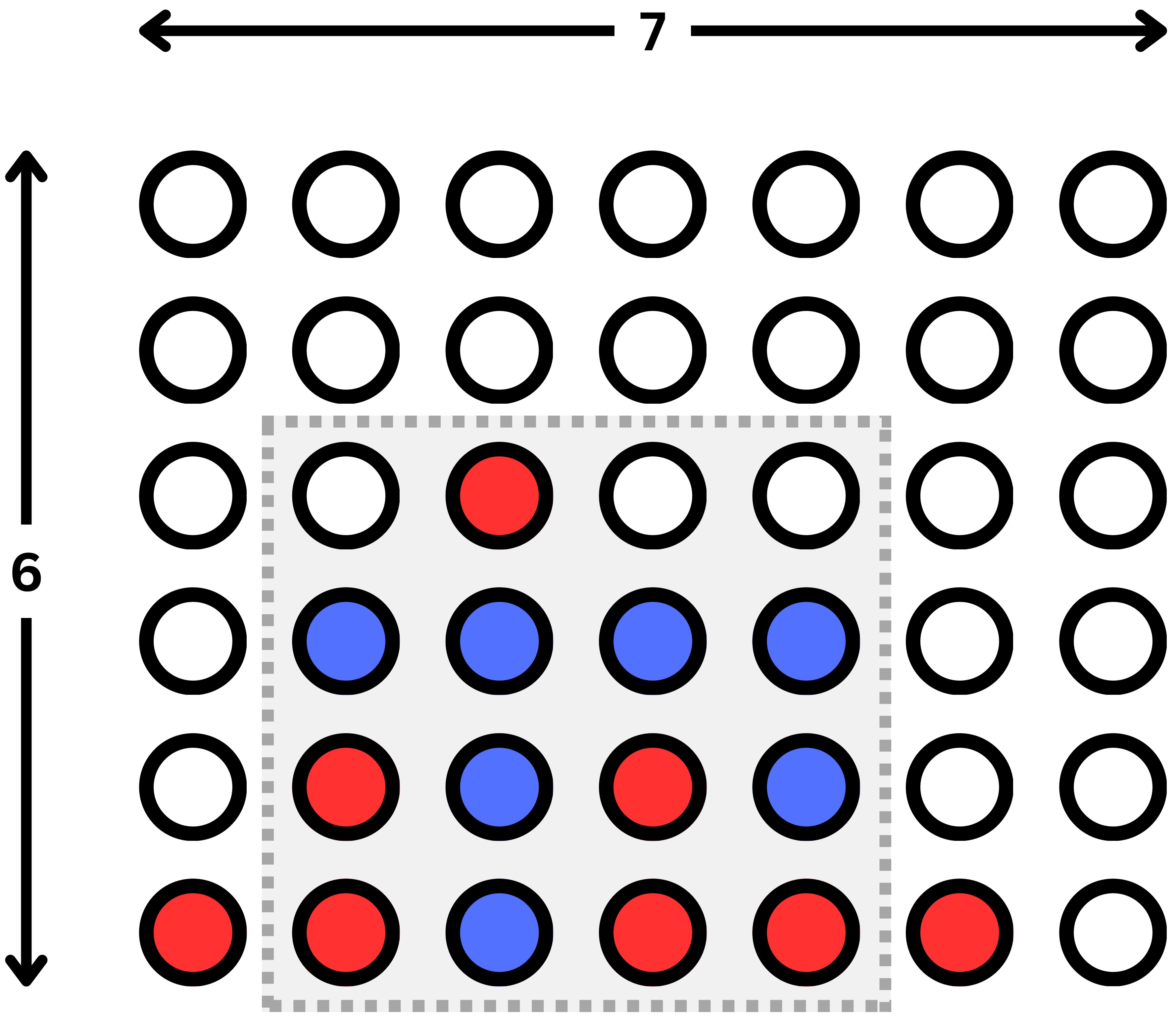
- Objective: **4 in a row** (any direction)
- **Gravity constraint** - pieces dropped down from above, so must stack (column must be filled bottom up)
- Huge state space - approximately **8 trillion valid states**



Connect 4

Subgame

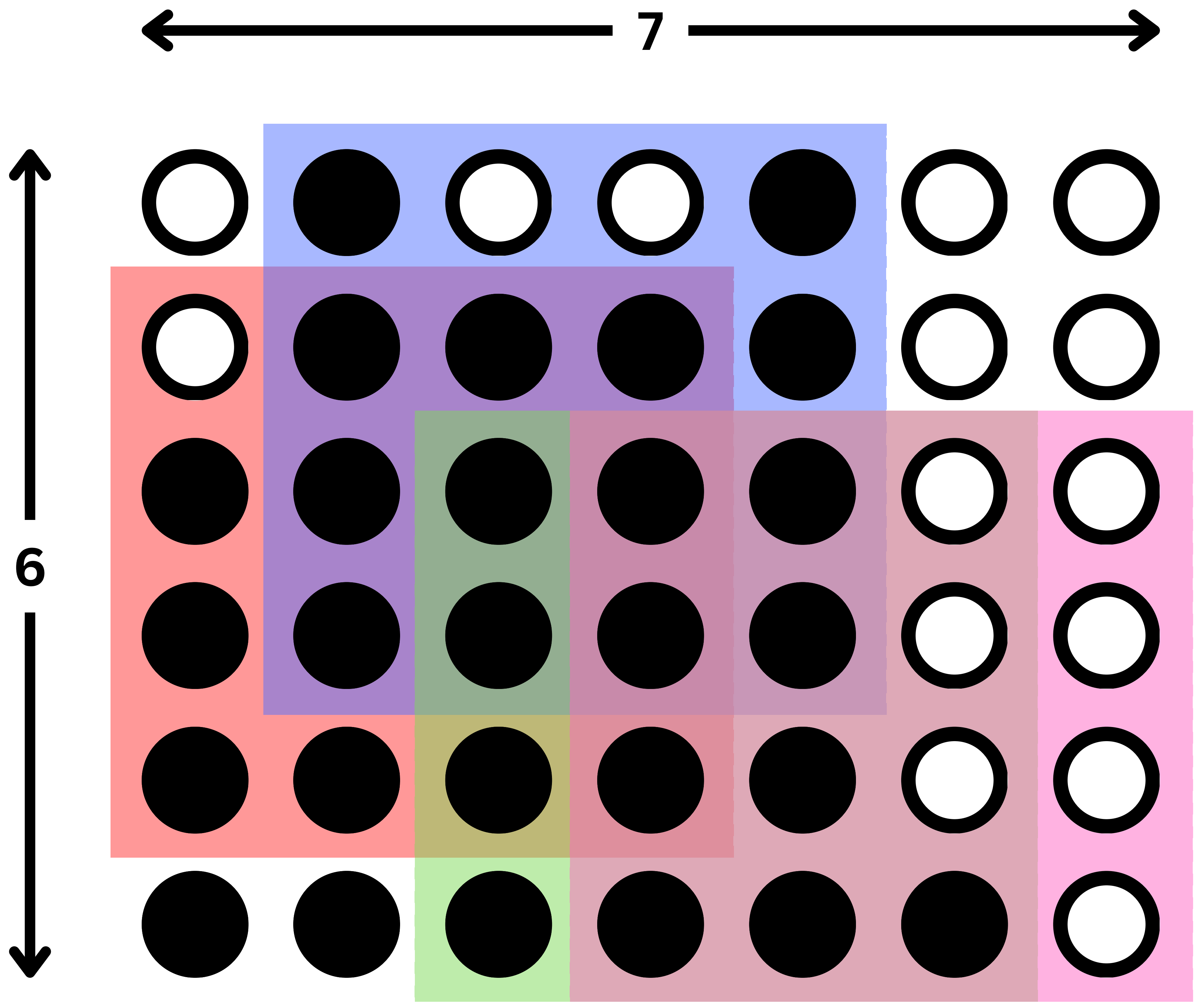
- Winning move always played in a **4x4 context window** (by nature of win condition)
- Thus, immediate **strategic decisions** made in 4x4 window
- Idea - use a 4x4 **subgame evaluator** to decide strategy/policy
- Benefit: vastly smaller state space - less than 5 million gravity-valid states



Windows

Selection Strat.

- Sliding window evaluation, maximum overlap (4-1 = 3)
- **Full horizontal coverage** (7-4 + 1 = 4)
- **Relevance-based vertical coverage** - bottom up first non-trivial window



RL Approaches

Overview

- Given the smaller state space, we can feasibly attempt a tabular Q-learning approach
- **Off-policy learning** - learning without a smart opponent, sample efficiency important in early training phases
  - Faster generalization
  - Stationary environment (see next)

$$Q(s,a) \leftarrow Q(s,a) + \alpha \left[R + \gamma \max_{a'} Q(s_{t+1}, a') - Q(s,a) \right]$$

RL Approaches

Rewards in an Adversarial Environment

- “Next state” for learner is actually two states ahead
- Could choose to update only on learner’s turns, but miss out on learning from opponent’s strategy
- Implicit opponent modelling (causal links)
- **Rewards can be:**
  - Min/Max approach (not compatible with perspective encoding)
  - **Flipped rewards** (from learner’s perspective)

$$\text{Reward} = \begin{cases} 1 & \text{if Player Win} \\ -1 & \text{if Player Win} \\ 0 & \text{if Draw (or other)} \end{cases}$$

(Works because dealing with terminal rewards)

RL Approaches

Reward Shaping

- **Intermediate block rewards** to encourage defensive behavior
- Detection of partial/full and single/double open ended blocks, weighted accordingly
- Reward not too high, to still **push for win-seeking behavior**

$$\text{Reward} = \begin{cases} 1 & \text{if Player Win} \\ -1 & \text{if Player Win} \\ 0.1 & \text{if Learner Blocks} \\ 0 & \text{if Draw (or other)} \end{cases}$$

RL Approaches

Perspective Encoding

- Connect 4 has a significant 1st player bias, indicating an asymmetric strategy - can‘t just invert
- Empirically proven during initial random trials
- Attempts to create a string 2nd move player as well
- Encoding current game state with player perspective - different Q-values depending on who sees it

$$\text{State} = \left(1, \begin{bmatrix} 0 & 0 & 0 & 0 \\ 1 & 0 & 0 & 0 \\ 2 & 2 & 1 & 0 \\ 1 & 2 & 1 & 2 \end{bmatrix} \right)$$

Training
Essentials

- Alternating starting player
- Self-play fraction (mitigate overfitting, more balanced strategy)
- Epsilon-greedy strategy
- Conditional geometric epsilon decay schedule
  - Start high at 0.8
  - Decay geometrically by factor 0.9 every 5000 episodes if improvement in avg. reward
  - Apply slight boost (1.01) if no improvement

$$\epsilon_{i+1} = \begin{cases} 0.9\epsilon_i & \text{if } \bar{R}_{i+1} - \bar{R}_i > -5 \cdot 10^{-3} \\ 1.01\epsilon_i & \text{Otherwise} \end{cases}$$

Training
Phases

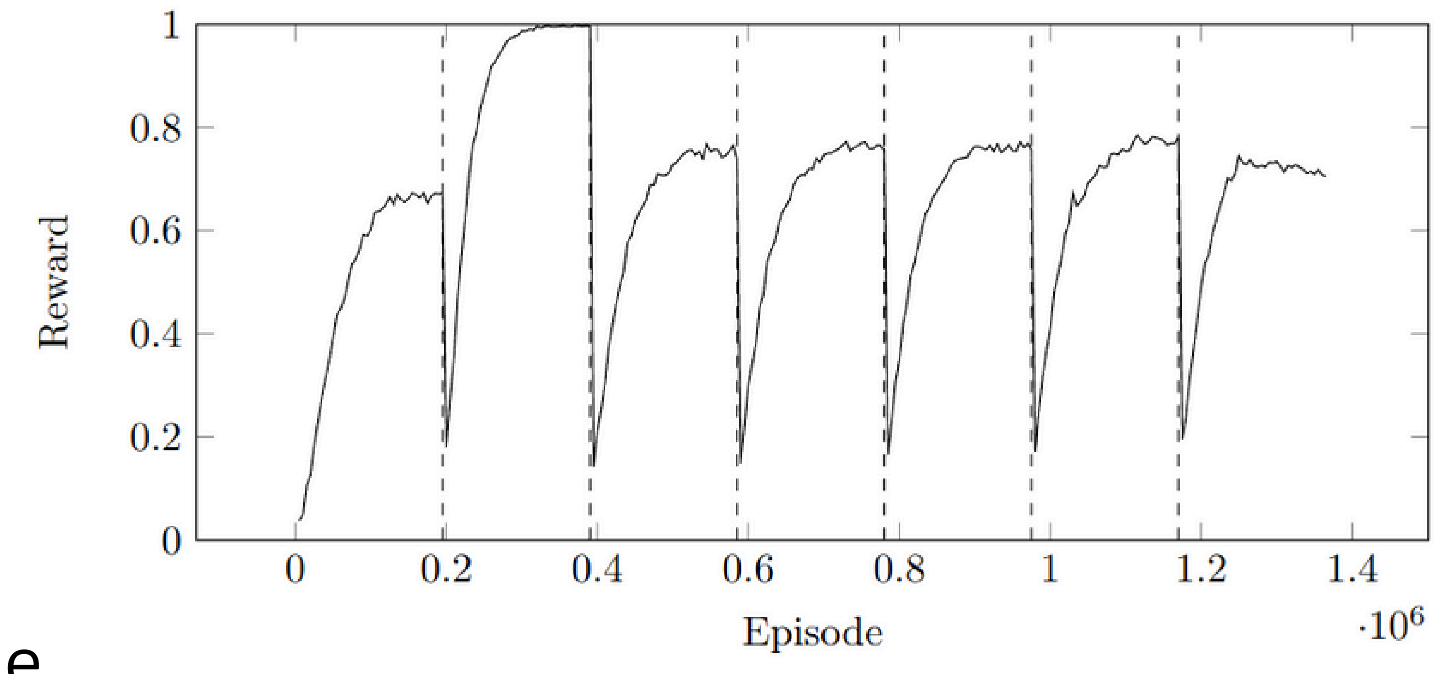


Figure 1: Agent avg. reward time-series data

- **Random Opponent**
  - Creating a baseline strategy, introduction to the game
  - Learning to not lose
- **Pure Self Play**
  - Learning to win
- **Adversarial (Frozen Opponent)**
  - 5 generations of updating opponents frozen on best recent learner
  - Strategy refinement against more intelligent opponent

Training
Results

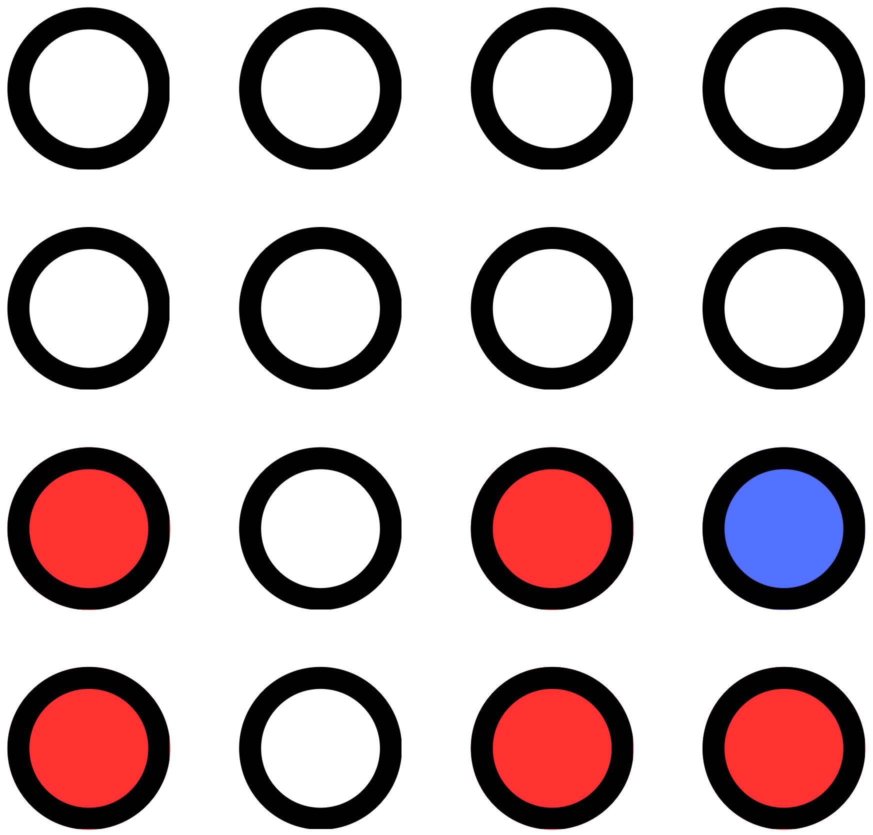
Training Phase	Win Rate	Loss Rate	Draw Rate
After Random Opponent	0.709	0.239	0.052
After Self-Play	0.691	0.056	0.253
After Adversarial (Final)	0.739	0.028	0.233

Table 1: Agent performance (fraction of games) against a random opponent after each training phase.

Issues

In application to the window agent

- Often encounter **unreachable** (by 4x4 normal play), but valid states in full game
- **Solutions**
    - Random (exploring starts) - initialize game with a random gravity-valid state for a fraction of training episodes. Worked, but would need more training
    - Future scope: use DQL (function approximation) to train subgame evaluator to extract strategic information regardless of actual game state



Conclusions
W.R.T Subgame and Full

- Pretty good results for naive subgame evaluator, could do better with smarter phases
- Must pivot approach to function approximation for truly valid application as window evaluator