**2048 development**

GAME ENVIRONMENT DEVELOPMENT

Pseudocode:

INITIALIZE:

board = 4x4 grid of zeros

score = 0

ADD two random tiles (2 or 4) to empty positions

GAME\_LOOP:

DISPLAY board and score

GET user input (UP, DOWN, LEFT, RIGHT)

old\_board = copy of current board

EXECUTE\_MOVE(direction):

FOR each row/column in direction:

COMPRESS tiles (remove zeros)

MERGE adjacent identical tiles

COMPRESS again

UPDATE score with merged values

IF board changed:

ADD new random tile (90% chance of 2, 10% chance of 4)

CHECK win condition (2048 tile exists)

CHECK lose condition (no valid moves possible)

REPEAT until win or lose

## **Key Game Mechanics**

Tile Movement Logic:

1. Compression: Move all non-zero tiles toward the chosen direction, removing gaps
2. Merging: Combine adjacent tiles with same values (only once per move)
3. Final Compression: Remove gaps created by merging

## 

## **Core Functions**

Move Function:

FUNCTION move\_left(board):

moved = false

FOR each row:

new\_row = compress(row)

new\_row, points = merge(new\_row)

new\_row = compress(new\_row)

IF new\_row != original\_row:

moved = true

score += points

RETURN moved, score\_increase

Merge Function:

text

FUNCTION merge(row):

points = 0

FOR i = 0 to length-2:

IF row[i] == row[i+1] AND row[i] != 0:

row[i] \*= 2

points += row[i]

row[i+1] = 0

skip next iteration // prevent double merge

RETURN row, points

## **Game State Management**

Valid Move Check:

* Board has empty cells, OR
* Adjacent tiles can be merged (horizontally or vertically)

Win/Lose Conditions:

* Win: Any tile reaches 2048[4](https://en.wikipedia.org/wiki/2048_(video_game))
* Lose: No valid moves possible (board full + no merges available)

## **Strategic Elements**

The optimal algorithm for 2048 uses Expectimax - a decision tree that considers both player moves and random tile placement[3](https://www.educative.io/answers/what-is-the-optimal-algorithm-for-2048-game). Key strategies include:

* Keep largest tile in a corner[4](https://en.wikipedia.org/wiki/2048_(video_game))
* Build monotonic sequences (decreasing values from corner)
* Maintain empty spaces for maneuvering
* Avoid separating small tiles with large ones

## **DQN Core Pseudocode**

text

INITIALIZE:

Q\_network = neural network (state → Q-values for 4 actions)

target\_network = copy of Q\_network

replay\_buffer = empty memory buffer

epsilon = 1.0 (exploration rate)

TRAINING\_LOOP:

state = encode\_board\_state(game.board) // 4x4 → 16-dim vector

ACTION\_SELECTION:

IF random() < epsilon:

action = random\_action() // exploration

ELSE:

Q\_values = Q\_network.predict(state)

action = argmax(Q\_values) // exploitation

EXECUTE\_STEP:

next\_state, reward, done = game.step(action)

STORE\_EXPERIENCE:

replay\_buffer.add(state, action, reward, next\_state, done)

TRAIN\_NETWORK:

IF replay\_buffer.size() > batch\_size:

batch = sample\_random\_batch(replay\_buffer)

train\_on\_batch(batch)

UPDATE\_TARGET:

IF episode % target\_update\_freq == 0:

target\_network = copy(Q\_network)

DECAY\_EXPLORATION:

epsilon = max(epsilon\_min, epsilon \* decay\_rate)

## **State Representation for 2048**

Board Encoding:

text

FUNCTION encode\_state(board):

flat\_board = flatten(board) // 4x4 → 16 elements

encoded = zeros(16)

FOR i in range(16):

IF flat\_board[i] > 0:

encoded[i] = log2(flat\_board[i]) // 2→1, 4→2, 8→3, etc.

RETURN encoded

Why log encoding?

* Normalizes exponential tile values (2, 4, 8, 16, ..., 2048)
* Makes learning more stable for neural networks

## **Q-Learning Update Rule**

Target Calculation:

text

FUNCTION calculate\_targets(batch):

FOR each experience in batch:

IF done:

target = reward

ELSE:

next\_Q\_values = target\_network.predict(next\_state)

target = reward + gamma \* max(next\_Q\_values)

current\_Q\_values = Q\_network.predict(state)

current\_Q\_values[action] = target

train\_Q\_network(state, current\_Q\_values)

## **Neural Network Architecture**

Network Structure:

text

INPUT: 16 neurons (encoded board state)

HIDDEN: [512, 512, 256, 128] neurons with ReLU activation

OUTPUT: 4 neurons (Q-values for LEFT, UP, RIGHT, DOWN)

FUNCTION forward\_pass(state):

x = ReLU(dense1(state))

x = ReLU(dense2(x))

x = ReLU(dense3(x))

Q\_values = dense4(x) // no activation

RETURN Q\_values

## **Reward Engineering for 2048**

Reward Function:

text

FUNCTION calculate\_reward(moved, score\_increase, max\_tile\_increase):

IF not moved:

RETURN -1.0 // penalty for invalid moves

reward = 0

reward += score\_increase \* 0.01 // score improvement

reward += max\_tile\_increase \* 50 // reaching higher tiles

reward += corner\_strategy\_bonus() // keeping large tiles in corners

reward += monotonicity\_bonus() // ordered sequences

reward += empty\_cells\_bonus() // maintaining space

RETURN reward

## **Key DQN Challenges in 2048**

Sparse Rewards:

* Most moves give small/zero rewards
* Major rewards only when merging or reaching new tiles
* Solution: Shaped rewards for strategic play

Large State Space:

* 2^16 possible board configurations
* Neural network must generalize across similar patterns

Action Masking:

text

FUNCTION get\_valid\_actions(board):

valid = []

FOR action in [LEFT, UP, RIGHT, DOWN]:

test\_board = simulate\_move(board, action)

IF test\_board != board:

valid.append(action)

RETURN valid

## **Training Process**

Experience Replay:

* Store (state, action, reward, next\_state, done) tuples
* Sample random batches to break correlation
* Stabilizes learning by reusing past experiences

Target Network:

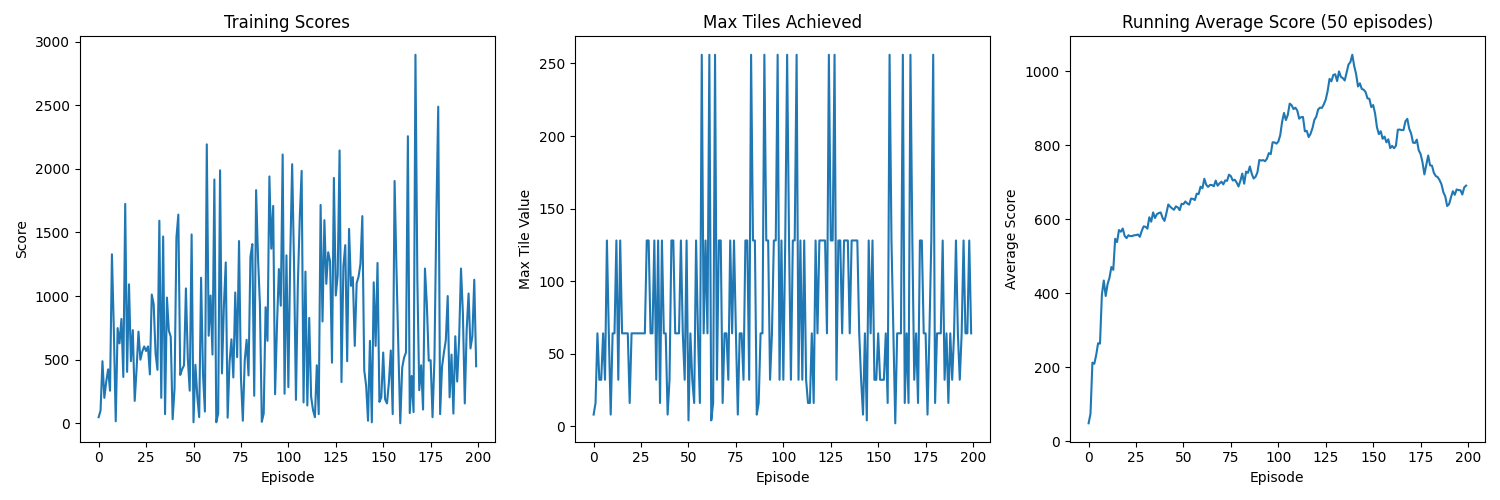
* Separate network for calculating target Q-values
* Updated less frequently to reduce moving target problem
* Prevents instability in Q-learning updates

Exploration Strategy:

* ε-greedy: Start with random actions (ε=1.0)
* Gradually reduce to exploitation (ε=0.1)
* Balances learning new strategies vs using known good ones

This DQN implementation transforms 2048 from a puzzle game into a Markov Decision Process, where the agent learns optimal tile-moving strategies through trial and error, eventually discovering advanced techniques like corner strategies and monotonic building patterns that human players use

Some Graphs on the training:



## **Performance Analysis Pseudocode**

text

EVALUATION\_RESULTS:

games\_played = 10

games\_won = 0 // (reaching 2048 tile)

win\_rate = 0%

average\_score = 1175

best\_score = 1500

max\_tile\_distribution = {128: 70%, 64: 30%}

PERFORMANCE\_ASSESSMENT:

IF win\_rate == 0 AND average\_score < 5000:

status = "UNDERPERFORMING"

ELIF win\_rate < 10% AND average\_score < 10000:

status = "LEARNING\_IN\_PROGRESS"

ELSE:

status = "PERFORMING\_WELL"

## **Conclusion Analysis**

Current Agent Performance Level:

text

FUNCTION analyze\_performance(results):

benchmark\_comparison = {

"random\_play": 200-500 average score,

"basic\_strategy": 2000-5000 average score,

"good\_rl\_agent": 8000-15000 average score,

"expert\_level": 20000+ average score

}

current\_level = "BETWEEN\_RANDOM\_AND\_BASIC"

improvement\_needed = "SIGNIFICANT"

RETURN performance\_diagnosis

Key Findings:

* Learning Stage: Early/intermediate learning phase
* Strategy Development: Agent has learned basic tile movement but lacks advanced strategy
* Tile Progression: Consistently reaches 64-128 tiles (good foundation)
* Strategic Gaps: Missing corner strategy, monotonicity, and long-term planning

## **Recommended Next Steps**

Training Improvements:

text

FUNCTION improve\_training():

increase\_training\_steps(current=1M, target=3-5M)

enhance\_reward\_function(corner\_bonus \*= 4, strategy\_rewards += new\_components)

adjust\_exploration(final\_epsilon = 0.2, exploration\_fraction = 0.5)

monitor\_progress(checkpoint\_every=100k\_steps)

Expected Progression Timeline:

text

TRAINING\_MILESTONES:

500k\_steps: "Should reach 256 tiles regularly"

1M\_steps: "Should occasionally reach 512 tiles"

2M\_steps: "Should start reaching 1024 tiles"

3M+\_steps: "Should achieve occasional 2048s (10-20% win rate)"

## **Final Verdict**

Current Status: Promising but needs more training

* Agent has successfully learned basic game mechanics
* Shows consistent performance (not random behavior)
* Foundation is solid for further improvement

Success Indicators Achieved:

* ✅ Learned valid move selection
* ✅ Consistent tile merging
* ✅ Basic score accumulation

Missing Components:

* ❌ Advanced strategic thinking
* ❌ Corner/edge tile management
* ❌ Long-term planning capabilities

The results indicate your DQN implementation is working correctly but requires extended training time and enhanced reward engineering to reach competitive 2048 performance levels. This is typical for complex strategy games where optimal play requires learning sophisticated multi-step strategies.