Adaptive Game Testing with Deep Q-Networks

Nexus Interactive -- Using DQN Agents to Automate Quality Assurance for Mobile Games

---

Section 1: The Business Problem

Nexus Interactive is a mid-sized mobile gaming studio based in Austin, Texas, with a portfolio of 12 live mobile games generating approximately \$45 million in annual revenue. Their flagship title, *Dungeon Crawlers*, has 8 million monthly active users and receives bi-weekly content updates -- new levels, items, balance changes, and seasonal events.

The QA Bottleneck

Each content update introduces an average of 3 new levels and 15 balance adjustments. Before any update ships, the QA team must verify that:

1. All new levels are completable (no soft-locks or impossible states)
2. Balance changes do not make existing levels trivially easy or impossibly hard
3. No reward exploits exist (infinite loops, unintended reward farming)
4. The difficulty curve remains smooth across the 200+ level progression

The current QA process relies on a team of 18 human testers who manually play through affected levels. Each tester can thoroughly test approximately 5 levels per day. With 200+ levels potentially affected by balance changes, full regression testing takes **8-10 business days** -- consuming most of the two-week release cycle.

The Cost

Metric	Current State
QA team size	18 full-time testers
Annual QA cost	\$2.1 million
Regression test cycle	8-10 business days
Update frequency	Bi-weekly (target: weekly)
Bugs found post-release	3.2 per update (average)
Player churn from bugs	~2% per critical bug
Revenue loss per critical bug	\$180,000 (estimated)

Post-release bugs cost an estimated \$576,000 annually in player churn alone. Worse, the QA bottleneck prevents the studio from shipping updates weekly, which their analytics team estimates would increase player engagement by 23%.

Why Traditional Automation Fails

Nexus has tried scripted test automation. Each level requires hand-authored test scripts specifying the exact sequence of actions. This approach has three fatal problems:

- **Brittle scripts**: When game mechanics change, scripts break. Maintaining 200+ level scripts is a full-time job.
- **No creativity**: Scripts follow predetermined paths. They cannot discover emergent exploits or unexpected soft-locks that arise from novel player behavior.
- **No difficulty assessment**: Scripts can verify that a level is completable but cannot measure whether it is *appropriately difficult* -- a subjective quality that depends on player skill level.

The VP of Engineering, Sarah Chen, proposes a different approach: train DQN agents that learn to play the game, then use their behavior and performance metrics as an automated QA signal.

Section 2: The Technical Solution

Core Idea

Instead of scripting test cases, Nexus trains DQN agents to play each game level. The agents learn from raw game state (simplified pixel observations) and interact with the game through the same action space available to human players. Once trained, the agents provide three automated QA signals:

1. **Completability**: If a DQN agent cannot learn to complete a level within a training budget, the level may contain a soft-lock or impossible state.
2. **Difficulty estimation**: The training curve of the DQN agent -- how many episodes it takes to achieve consistent completion -- correlates with human-perceived difficulty.
3. **Exploit detection**: If a DQN agent discovers a strategy that yields abnormally high rewards, it has likely found a reward exploit.

Architecture

The system uses a modified DQN architecture tailored for the grid-based dungeon crawler mechanics:

State representation: Unlike raw Atari pixels, Dungeon Crawlers uses a tile-based grid. Each level is represented as a multi-channel 2D grid where channels encode: - Channel 0: Terrain (walls, floors, doors, traps) - Channel 1: Items (keys, potions, weapons) - Channel 2: Enemies (type and health) - Channel 3: Player position and health - Channel 4: Fog of war (explored vs unexplored)

Grid size: 32x32 tiles with 5 channels, giving a state tensor of shape $(5, 32, 32)$.

Network: A modified DQN with smaller kernels (appropriate for 32x32 input):

Layer	Filters	Kernel	Stride	Output
Conv1	32	5x5	2	32 x 14 x 14
Conv2	64	3x3	1	64 x 12 x 12
Conv3	64	3x3	1	64 x 10 x 10
FC1	256	-	-	256
Output	8	-	-	8 actions

Actions: move (4 directions), attack, use item, interact, wait.

Training configuration:

Parameter	Value	Rationale
Replay buffer	50,000	Sufficient for single-level training
Batch size	64	Faster convergence on smaller state space
Gamma	0.99	Standard discount factor
Learning rate	5e-4	Faster than Atari (simpler environment)
Target update	Every 500 steps	More frequent (shorter episodes)
Epsilon decay	10,000 steps	Fast exploration (levels take ~200 steps)
Training budget	500 episodes per level	Hard limit for completability test

Double DQN for Stability

The system uses Double DQN to avoid Q-value overestimation, which is critical for the difficulty estimation use case. Overestimated Q-values would make the agent appear to find levels easier than they actually are, corrupting the difficulty signal.

QA Decision Pipeline

After training a DQN agent on a level, the system extracts three metrics:

Metric 1: Completion Rate (CR)

$$CR = \frac{\text{episodes completed in last 100}}{100}$$

If $CR < 0.1$ after 500 training episodes, the level is flagged as potentially impossible. If $CR > 0.95$ after fewer than 50 episodes, the level is flagged as potentially too easy.

Metric 2: Difficulty Score (DS)

$$DS = \frac{\text{episodes to first completion}}{500} \times 10$$

Clamped to $[0, 10]$. This gives a 0-10 difficulty rating. The studio maintains a target difficulty curve across the 200+ levels that should increase monotonically with occasional difficulty resets at new "world" boundaries.

Metric 3: Exploit Score (ES)

$$ES = \frac{\max(\text{episode reward}) - \mu_{\text{reward}}}{\sigma_{\text{reward}}}$$

If $ES > 5$ (reward more than 5 standard deviations above the mean), the agent has likely discovered an exploit strategy. The replay buffer is examined to reconstruct the exploit sequence.

Infrastructure

The training infrastructure runs on a cluster of 8 T4 GPUs in Google Cloud. Each GPU trains agents on 4 levels simultaneously (32 levels in parallel across the cluster). A full regression test of all 200+ levels completes in approximately 3 hours, compared to the previous 8-10 business days.

Section 3: Results and Impact

Pilot Results (3-Month Trial)

Nexus deployed the DQN-based QA system alongside their human testing team for a 3-month pilot covering 6 bi-weekly updates.

Completability Testing

Metric	Before (Human QA)	After (DQN + Human)
Soft-locks detected pre-release	71%	94%
Average detection time	4.2 days	3.1 hours
False positive rate	N/A	8% (4 false flags in 50 levels)

The DQN agents found 3 soft-locks that human testers had missed. In each case, the soft-lock occurred under unusual conditions (e.g., using a specific item combination near a wall boundary) that human testers would not have tried.

Difficulty Estimation

The DQN difficulty scores correlated with human difficulty ratings at $r = 0.82$ (Pearson correlation). This was sufficient for automated difficulty curve validation. In one update, the system flagged that a balance change had made Level 47 (difficulty target: 4/10) effectively a 1/10 difficulty -- the enemy damage reduction was too aggressive. Human testers had not caught this because they tested the level in isolation, not relative to the surrounding difficulty curve.

Exploit Detection

Across the pilot period, the DQN agents discovered 2 previously unknown reward exploits:

1. A gold farming loop on Level 83 where enemies respawned faster than intended near a checkpoint
2. An item duplication exploit on Level 112 triggered by a specific interact-move-interact sequence

Both exploits were confirmed by human testers and fixed before the update shipped.

Business Impact

Metric	Before	After	Change
QA cycle time	8-10 days	1.5 days (DQN) + 2 days (human review)	-65%
Post-release bugs	3.2 per update	0.8 per update	-75%
Update frequency	Bi-weekly	Weekly	2x
QA team reallocation	18 testers on regression	6 on regression, 12 on new content	Creative reallocation
Annual QA cost	\$2.1M	\$1.8M (saved \$300K, reinvested in GPU cluster)	-14%
Revenue impact (weekly updates)	Baseline	+\$4.2M annual (23% engagement increase)	+9.3%

Limitations Observed

1. **Platforming levels**: DQN agents struggled with levels requiring precise timing (jumping puzzles). The frame-skip mechanism, which works well for strategy, was too coarse for timing-dependent actions.

2. **Cooperative mechanics**: Some late-game levels involve managing an AI companion. The DQN agent learned to ignore the companion entirely, finding completion strategies that were valid but not representative of human play.

3. **Narrative puzzles**: Levels with dialog-based puzzles or riddle mechanics could not be meaningfully tested by the DQN agents.

For these categories (approximately 15% of levels), human testers remain essential.

Section 4: Technical Deep Dive

Modified DQN for Game Testing

The standard DQN architecture requires several modifications for the game testing use case.

Reward shaping: In the original game, the only reward is completing the level. This sparse reward makes DQN training extremely slow. The QA system adds intermediate rewards:

$$r_{\text{shaped}} = r_{\text{game}} + 0.1 \cdot \Delta_{\text{exploration}} + 0.05 \cdot \Delta_{\text{items}} - 0.01 \cdot \Delta_{\text{health\_loss}}$$

where $\Delta_{\text{exploration}}$ is the fraction of new tiles explored this step, Δ_{items} is the number of items collected, and $\Delta_{\text{health\_loss}}$ penalizes taking damage.

Curriculum training: Rather than training each level from scratch, the system uses transfer learning. An agent trained on Level N initializes the weights for Level $N+1$. This reduces training time by approximately 40% because adjacent levels share similar mechanics.

Action masking: Invalid actions (moving into walls, using items when none are available) are masked in the Q-value output. The DQN predicts Q-values for all 8 actions, but invalid actions are set to $-\infty$ before the argmax operation.

Parallel level evaluation: The system leverages vectorized environments. Each GPU runs 4 independent level instances simultaneously, sharing the same network weights. This is critical for achieving the 3-hour full regression target.

Implementation Details

```
class GameTestDQN(nn.Module):
    def __init__(self, n_actions=8):
        super().__init__()
        self.conv = nn.Sequential(
            nn.Conv2d(5, 32, kernel_size=5, stride=2, padding=1),
            nn.ReLU(),
            nn.Conv2d(32, 64, kernel_size=3, stride=1),
            nn.ReLU(),
            nn.Conv2d(64, 64, kernel_size=3, stride=1),
            nn.ReLU()
        )
        self.fc = nn.Sequential(
            nn.Linear(64 * 10 * 10, 256),
            nn.ReLU(),
            nn.Linear(256, n_actions)
        )

    def forward(self, x, action_mask=None):
        x = self.conv(x)
        x = x.view(x.size(0), -1)
        q_values = self.fc(x)
        if action_mask is not None:
            q_values = q_values.masked_fill(~action_mask, float('-inf'))
        return q_values
```

Difficulty Calibration

The raw DQN difficulty score must be calibrated against human play data. Nexus collected difficulty ratings from 500 human playtesters across 50 levels. A simple linear regression maps DQN difficulty scores to the 1-10 human scale:

$$DS_{\text{calibrated}} = \alpha \cdot DS_{\text{raw}} + \beta$$

With $\alpha = 0.73$ and $\beta = 1.2$ (fitted on the 50-level calibration set), the calibrated scores achieve $r = 0.82$ correlation with human ratings. This calibration is re-run quarterly as game mechanics evolve.

Cost Analysis

Component	Monthly Cost
8x T4 GPUs (Google Cloud)	\$4,800
Storage (replay buffers, models)	\$200
Engineering maintenance	\$8,000 (0.5 FTE)
Total	**\$13,000/month**

Compared to the \$175,000/month human-only QA cost, the hybrid system (DQN + reduced human team) costs \$163,000/month -- a net savings of \$12,000/month (\$144,000/year) while delivering significantly better coverage and enabling weekly releases.

The real ROI, however, is the \$4.2 million annual revenue increase from weekly updates -- a 29x return on the \$144,000 investment.