

Mission Winning — slide deck

Inside the Game hackathon (Microsoft, 2026) · explainable match intelligence Mission Winning · synthetic football only — the fictional Harbor Invitational

Every number here is measured and every file cited is in the repository. Two known gaps — repository visibility and the Azure account — are stated on the last slide, not buried.

1. The problem

A match gives you numbers. The viewer wants the reason.

- Live football tells you a shot was 0.31 xG and the ribbon moved 12 points.
- It does not tell you *why*, and the broadcaster still has to write that line.
- Two audiences, one match: an analyst wants xG, pass difficulty and xT; a casual fan wants one sentence. Both want it now, in their language.
- And the reason is where generated commentary invents things.

2. What we built

One pipeline, five stages — the shape the challenge asks for.

Stage	What happens
Ingest	Events stream into one shared match state
Interpret	xG, momentum windows, pressure surges, pass difficulty, synthetic xT
Explain	Every win-chance swing gets a factual reason from that state
Render	Timestamped overlay cards + win-chance ribbon, replayed at 30×
Personalize	Fan or analyst · English, Spanish, French · favourite player

3. The five agents, one shared state

Agent	File	Role
Data	itg/agents/data_agent.py	Streams events into shared match state
Analysis	itg/agents/analysis_agent.py	xG, momentum, pressure, pass difficulty
Win-chance	itg/winprob.py	Live probability + why it moved
Story	itg/agents/story_agent.py	Persona, language, favourite player
QA	itg/agents/qa_agent.py	Recomputes every claim; rejects wagering language

- One state, one query surface (itg/facts.py) — which is *why* QA can re-derive any card.
- Handoffs are recorded: each step in the trace names its predecessor.

4. The QA guarantee — the strongest idea here

We do not ask whether the sentence sounds right. We recompute it.

- Every card carries machine-readable claims: high-xG counts, pressure counts, goals and who scored, xG added, momentum windows, pass difficulty, shot identity, clock and score.
- QA re-derives each one **from match state** and compares. Unknown claim kinds are rejected.
- Wagering language is rejected by pattern match.
- The recap never reaches the screen on an unverified claim.

`itg/agents/qa_agent.py` — 8 claim kinds recomputed, not one prompt.

5. Failure is visible, and recovered

A rejected claim is a feature, not a log line.

Verified run, seed 7, `?inject=1`:

```
Story | injected      | drafted a fan recap, then injected a fabricated claim
QA    | rejected        | high-xG count 9 exceeds match state (4)
                                   | pressure count 221 exceeds match state (216)
                                   | win chance swing 99%→1% not in match state
Story | template_fallback | offline template restored
QA    | ok              | template recap accepted
```

- The failure is on the demo screen, in the trace panel, with its reasons.
- Recovery is a designed edge: QA reject → Story on the offline template only → QA re-checks.



6. Calibrated, not vibes

Brier 0.350 vs 0.390 — on held-out matches, not on the demo seed.

- Model: Poisson on remaining minutes; rate blends a pre-match strength prior with observed xG per minute. Two knobs, both visible.
- Fitted by grid search on 300 synthetic matches, scored on **300 held-out** synthetic matches — seven time points each, so 2,100 scored predictions.
- Brier **0.3504** / log-loss **0.5986**
- Score-only baseline (no strength, no xG): **0.3899** / **0.6727**
- Reliability table: predicted vs actual within 7 points in every decile (largest gap 6.8 points). Bin counts are predictions, not matches.
- A 40-match smoke check runs in the test suite and must stay under Brier 0.40.

`itg/winprob.py` · `predictions/winprob-backtest.json` · the method, the reliability table bin by bin, and what this does and does not establish: `docs/CALIBRATION.md`

7. Ask the match — the MCP server

The same state, published to any agent. No key, no cloud, no account.

```
$ python -m itg.mcp --demo

> tools/call verify_claim
  accepted false
  issue      - high-xG count 14 exceeds match state (4)
```

Tool	What it answers
<code>get_match_state</code>	the match: teams, score, events ingested, squad
<code>get_key_moments</code>	the key moments, each with its reason and its claims
<code>get_win_probability</code>	the win chance at a minute, and why it moved
<code>verify_claim</code>	is this sentence about the match true?
<code>verify_overlay_claims</code>	recompute every number on every card
<code>get_agent_trace</code>	the handoffs; what was rejected and recovered
<code>get_player_profile</code>	one player: distance, speed, carries, xT

- JSON-RPC 2.0 over stdio, **seven tools**, and it adds **nothing** to the install a judge runs. The guardrail is a tool an agent can call, not internal plumbing. `itg/mcp.py` · `tests/test_mcp.py`

8. What an explanation actually looks like

A real card from the running demo (seed 7):

*10'→15': Harbor City FC win chance **39%**→**8%**, tilted to Northgate United by a goal by Vidal; 1 high-xG chance under sustained press (10 pressures).*

- The number is computed; the reason is **re-derived from state** before render.
- The same card appears in English, Spanish and French.
- Match intelligence, not a price: no odds, no wagering — and the QA agent blocks that language.

9. Personalize — two audiences, three languages

- **Fan:** the moment and the reason, one line each.
- **Analyst:** adds xG, pass accuracy, mean pass difficulty, xT.
- **EN / ES / FR** — the challenge names multi-language storytelling; this is built into the templates, not bolted on.
- **Favourite player:** distance covered, top speed, carries — from the generated player tracking, not a lookup.
- 22-player team sheet; a player who is not on it says so, instead of inventing a line for them.

10. Synthetic football, honestly

- Our own seeded possession-chain simulator: pass, carry, shot, tackle, pressure, possession change, goal — with metadata, player speed and distance.
- Same seed → same match. A test asserts no real club or competition name appears anywhere in a generated match.
- xT is a location function written for this project, **not** fit on real matches. On-ball fields follow the SPADL shape, credited in `NOTICE` (MIT).
- Fictional Harbor Invitational, Harbor City FC and Northgate United. **No real footage, no real data, no real club.**

11. Microsoft and Azure — wired vs planned

	Status
Microsoft Agent Framework (agent-framework)	Wired, in two places: the orchestration graph in <code>itg/workflow.py</code> — the default orchestrator — and the model backend in <code>itg/llm.py</code> (<code>Agent</code> + <code>FoundryChatClient</code> , selected by <code>FOUNDRY_PROJECT_ENDPOINT</code>)
The multi-agent graph	Wired: five specialised agents as a graph with a QA→Story retry edge. A dependency-free pipeline produces the same answer, and a test asserts they agree
MCP (Model Context Protocol)	Working and running: <code>itg/mcp.py</code> publishes the match state over JSON-RPC 2.0 on stdio, seven tools, no key and no dependency. The guardrail is a tool an agent can call. This is the one hero technology that works without an Azure account
The static demo (no server, no key, no cloud)	Wired: <code>python -m itg.export</code> builds it; the repository's own workflow publishes it
Microsoft Foundry (gpt-4o-mini class)	Planned — no Azure account yet, so the live path has never been executed. The code path is wired, and the credential switches to managed identity inside Azure
Azure Container Apps	Planned , not deployed — <code>docs/AZURE.md</code> has the exact <code>az</code> commands, and the <code>Dockerfile</code> is the image they build
Azure MCP, Fabric, Copilot agent mode	Not used

- We would rather show you the path that runs than the path we wish ran.

12. The demo

```
pip install -r requirements.txt && python -m itg.web      # http://127.0.0.1:8000
python -m itg.mcp --demo                                # the MCP exchange above
```

- Pitch replay at 30x, play / pause / scrub, overlay cards, win-chance ribbon.
- Fan ↔ Analyst, EN ↔ ES ↔ FR, favourite player, and the agent trace with timings.
- **"Inject a fabricated claim"** — QA refuses it and the system recovers, live.
- `python -m pytest -q` → green on Python 3.9, 3.12 and 3.14, offline, no keys.
- The 1:18 video is the static export, recorded — the five stages, the agent trace, the MCP exchange.

The judge's version needs nothing. `python -m itg.export` builds the same demo as a static site — five precomputed variants, inlined, no server, no key — and the repository publishes it on every push. The video is that build.



13. Where it lands, what is missing, what we ask

Where it lands. A timestamped, machine-readable overlay feed — the shape a broadcast renderer consumes — localized, with a guardrail that re-derives every number before it is shown.

What is not done, stated plainly

- **No Azure account exists yet**, so Foundry is wired but unconnected, nothing is deployed on Azure, and the model backend has never run against a live endpoint. The `Dockerfile` has not been built here either, because Docker is not installed in the build environment.
- This repository is the **clean export** of the build: the private working repository's history contained an invitation redemption link, and it is in no commit here.

What we ask

- **Category: Best Multi-Agent Orchestration** — distinct roles, shared state, recorded handoffs, and failure recovery a single agent would not give you. The trace is on the video screen, not in a log file.
- **Grand Prize: First Place** as the overall entry.
- We are **not** entering the Foundry category until we can demonstrate it.

*The one thing to take away: **an explanation that is not recomputed is a caption.** Ours is checked against match state before anyone sees it — and when it fails, you watch it fail and recover. Then any agent can ask the match the same question, over MCP, and get the same checked answer.*