ChatGPT

QA Agent for Browser-Generated Web Games – Product Requirements Document

Background & Motivation

DreamUp lets children generate their own browser-based games. To continuously improve the game-building agent, DreamUp needs a QA pipeline that can run autonomously on any single-player HTML5 game and report whether the game loads, controls work, and the experience completes without crashes. Manual QA cannot scale to the volume of games created by users, so an agent-driven system is essential.

Browser automation frameworks such as Stagehand provide a reliable and AI-native way to control a browser. Stagehand is built for the AI era, blending deterministic code with AI assistance [1]. Stagehand v3 is more extensible, reliable and AI-ready than previous versions, with native support for Chrome DevTools Protocol and 44 % faster execution across iframes and shadow-DOM interactions [2]. By combining Stagehand with Browserbase, we obtain cloud-hosted browsers, session replay and prompt observability. Vercel's AI SDK provides the infrastructure and API to call large language models (LLMs) and to build agents that can observe, decide and act [3]. These technologies enable the QA agent to simulate gameplay, evaluate playability with heuristics and LLMs, and output structured results.

Goals & Objectives

1. **Headless gameplay testing.** Launch a headless (or headed fallback) browser session via Stagehand to load a game URL. Handle page loads with retry and timeout logic; detect common UI patterns such as start buttons, menus and game-over overlays.
2. **Config-driven inputs.** Accept a JSON configuration describing both the ordered list of actions/axes and the control layout. Map high-level actions (e.g., **Jump**, **MoveHorizontal**, **Move**) to one or more physical keys, virtual buttons or axes. Allow repeated key presses or held keys to emulate analog input.
3. **Simulate gameplay.** Use Stagehand's natural-language `page.act` API and direct key events to perform actions defined in the config. Provide safeguards such as per-action timeouts, a maximum number of actions and a total execution timeout to avoid infinite loops.
4. **Evidence capture.** Save 3–5 timestamped screenshots per test session, along with console logs and error messages, to a structured results directory. When games use multiple scenes or overlays, ensure the UI layer is visible in captured images. Stretch goal: capture GIFs or videos.
5. **Playability evaluation.** Compute a heuristic score based on the ratio of issues to actions and call an LLM via the Vercel AI SDK to answer targeted questions about the run. Combine the heuristic and the model's confidence to produce a final `playability_score`.
6. **Structured reporting.** Output a JSON report containing status, playability score, list of issues, screenshot metadata and a timestamp. Provide CLI output and an optional path for downstream dashboards or feedback loops.

7. **Robust error handling.** Enforce a maximum runtime (e.g., 5 minutes), per-action timeouts, and a configurable number of page-load retries. Gracefully handle failures (missing elements, screenshot errors, LLM timeouts, headless incompatibility) and record issues without aborting prematurely.
8. **Extensibility & generality.** Design the system as modular components so that new action types (drag, scroll) or input devices can be added easily. Ensure compatibility with future Stagehand updates, and provide optional dashboards and CI integration in later phases.

Users & Stakeholders

Role	Needs
DreamUp Game-Building Agent	Requires automated feedback on the quality and playability of games generated by users to improve its model.
Internal QA engineers	Need a reproducible, scalable way to run game tests and capture evidence without manual effort.
Children and end-users	Indirect beneficiaries; improved games lead to a better user experience.
Developers/ Maintainers	Need clear modular architecture, TypeScript types and docs to extend the system with new actions, game genres or evaluation strategies.

In-Scope Functionality

- **CLI entry point.** Provide a command `qa-agent test <game-url> [--config <file>]` that accepts the game URL and an optional JSON configuration. The CLI must support Bun/Node and run locally or in CI.
- **Configuration parser.** Validate and parse the config using `zod` schemas. Required fields include:
- `actions[]` – ordered list of test steps, each containing either a natural-language prompt or a key/axis reference.
- `controls` – mapping of high-level actions and axes to physical keys, arrow keys, WASD, joysticks or virtual D-pads.
- `timeouts` – load, action and total execution timeouts (with sensible defaults).
- `retries` – number of allowed page-load retries (default 3).
- **Browser management.** Use Stagehand to initialise a headless browser session. If the game fails to load in headless mode, fall back to a headed browser. Manage resource cleanup after each run.
- **Interaction engine.** Simulate test steps:
- For **Act** instructions, call `page.act(prompt)` to click or fill DOM elements using natural language – Stagehand's APIs make this intuitive and self-healing [4] .
- For **keypress** instructions, send repeated key events (e.g., press 'w' for jump; hold `ArrowLeft` for a negative axis). Clamp the number of repeats to avoid runaway loops.
- For **axis** inputs, alternate or hold keys to simulate continuous movement with smoothing; respect axis configuration in the schema.
- Provide per-action timeouts and a global maximum number of actions.
- **Evidence capture.** During execution, capture timestamped screenshots (at least three). Use Stagehand's screenshot method to ensure the UI is visible. Collect console logs and error messages. Store artifacts under a session-specific directory such as `results/<sessionId>/`.

- **Evaluation engine.** Calculate heuristic metrics: `playability_score = 1 – (#issues / max(actions, 1))`. Issues include load timeouts, missing elements, action failures, screenshot errors and axis misbehaviour. Use Vercel's AI SDK to ask questions like "Did the game load successfully?", "Were controls responsive?" and "Did the game finish without crashes?" and combine the model's confidence with the heuristic result. Fall back to heuristics if the LLM call fails or times out.
- **Result serialiser.** Produce a structured JSON report containing at minimum the following fields: `status`, `playability_score`, `issues[]`, `screenshots[]` (with filenames and timestamps) and `timestamp`. Include additional fields such as test duration, configuration summary, LLM responses and logs when needed.
- **Error handling & safety.**
- **Max execution time.** Cap the total run time (e.g., 5 minutes) to prevent infinite loops.
- **Per-action timeouts.** If an action exceeds its timeout (default 10 s), record an `action_timeout` issue and proceed.
- **Retries.** Retry page loads up to a configurable number of times before giving up.
- **Graceful degradation.** If screenshot capture or log collection fails, record a `screenshot_failed` or `log_failed` issue and continue.
- **Headless fallback.** Automatically switch to a headed browser if a game fails to load in headless mode.
- **Axis safety.** Limit the number of simulated key events per axis to avoid runaway loops.
- **LLM fallback.** If the AI call errors out or times out, revert to the heuristic score.

Out-of-Scope

- **Multiplayer or network-dependent games.** The system will not handle multiplayer synchronisation or network latency.
- **Mobile/touch-only games.** Games requiring touch gestures, accelerometers or mobile-only features are excluded in the first release.
- **Deep behavioural analysis.** The QA agent measures playability (loading, control responsiveness, completion), but does not assess fun, fairness or user satisfaction.
- **Real-time chat or social features.** Chat or user-generated content moderation is handled elsewhere.
- **Runtime modification of game code.** The agent interacts only through the DOM and input events; it does not modify the game itself.

Success Metrics

- **Run completion rate.** $\geq$ 95 % of single-player games with valid configs complete within the maximum runtime and produce reports.
- **Evidence coverage.** $\geq$ 3 screenshots and complete logs captured for each run; no more than 5 % of runs should record `screenshot_failed` or `log_failed` issues.
- **Score consistency.** Playability scores should vary by less than 5 % across repeated runs of the same game/config (indicating deterministic execution).
- **Error handling.** $\leq$ 2 % of runs should terminate unexpectedly (uncaught exceptions). All other failures should surface as recorded issues.

Data Structures

Configuration Schema (input)

Field	Type	Notes
`actions`	Array	Ordered list of test steps. Each entry contains either a `prompt` (natural-language description of what to click/fill) or a `key` / `axis` reference.
`controls`	Object	Maps high-level action names (e.g., `Jump`, `MoveHorizontal`, `Move`) and axes (1D or 2D) to arrays of physical keys/virtual buttons. Multiple bindings per action are allowed.
`timeouts`	Object	Configurable `load`, `action` and `total` timeouts (ms). Defaults: load = 30 s, action = 10 s, total = 5 min.
`retries`	Integer	Number of page-load retries before declaring a failure (default = 3).
`metadata`	Optional	Additional data such as game genre, expected outcome or notes for the evaluation prompt.

Output Schema

Field	Type	Notes
`status`	String	Overall outcome: `success`, `load_failed`, `action_failed`, `timeout`, etc.
`playability_score`	Float (0–1)	Combined heuristic and LLM evaluation score; higher is better.
`issues`	Array	List of issues encountered during the run; each issue has a `code`, `description`, and `timestamp`.
`screenshots`	Array	Metadata for each screenshot (filename, captured at step index, timestamp). Actual files are stored in the results directory.
`timestamp`	ISO-8601	Timestamp when the run completed.
`logs`	Optional	Captured console logs and error messages.
`llm_responses`	Optional	Answers returned by the AI model for targeted questions. Included only when LLM evaluation is enabled.

Non-Functional Requirements

- **Determinism & repeatability.** Stagehand's deterministic and repeatable automation ensures that scripts behave the same way across runs [5].

- **Performance.** Stagehand v3 interacts directly with the Chrome DevTools Protocol and is 44 % faster in complex environments [2]. The QA pipeline must complete each test within the configured maximum execution time.
- **Modularity.** Organise the code into separate modules (CLI, config parser, browser manager, interaction engine, capture manager, evaluation engine, result serializer). Modules should communicate via clear interfaces and be replaceable.
- **Scalability.** Support running multiple tests in sequence or parallel (subject to hardware limits). Provide caching of LLM responses to reduce API costs.
- **Observability & debugging.** Leverage Browserbase features such as session replay and prompt observability to diagnose failures. Provide detailed logs and a unique `sessionId` for each run.
- **Security & privacy.** Do not expose sensitive information from console logs or game content. Config files and results should be sandboxed per test session. Handle API keys (Browserbase, Vercel AI) securely.

Implementation Considerations

- **TypeScript & Bun.** Implement the CLI and modules in TypeScript, targeting Bun for improved performance and built-in TypeScript support. Provide `package.json` scripts for `bun run qa.ts` and `bun test`.
- **Stagehand integration.** Use Stagehand's natural-language and low-level APIs to interact with the game. Stagehand is compatible with Playwright but Stagehand v3 removes the Playwright dependency and introduces a modular driver system [6], allowing us to run on Puppeteer or CDP if needed.
- **LLM evaluation prompts.** Design prompts for the AI model to answer single-player QA questions. Include context such as the number of actions taken, the final screenshot and any error messages. Keep prompts concise to reduce token usage, and cache identical prompts/responses.
- **Token & cost management.** Vercel's AI SDK allows selecting different models and using an AI Gateway for unified access [7]. Use cheaper models for preliminary evaluations and reserve higher-end models for final runs. Cache responses when the same game and config are tested repeatedly.
- **Fallback strategies.** When the AI call fails or model budgets are exhausted, rely on heuristic scoring. Provide configuration flags to disable LLM evaluation.
- **Future extensibility.** Keep the architecture flexible to add support for drag, scroll, multi-touch inputs, new evaluation metrics or additional output formats. Plan for a web dashboard and CI integration in later phases.

Future & Stretch Goals

- **Expanded input types.** Add support for drag-and-drop actions, scroll events and pointer gestures to cover more complex games.
- **Evidence as video or GIF.** Capture short videos or animated GIFs to better visualise gameplay and UI transitions. Consider using Browserbase's session replay or external tools.
- **Game-specific heuristics.** For different genres (puzzle, platformer, clicker), introduce heuristics tailored to typical gameplay loops (e.g., number of moves, scoring systems).
- **Dashboard & CI integration.** Build a simple web dashboard to display results over time, with filters and session replay. Provide Dockerfiles and GitHub Actions workflows to run tests automatically on game submissions.

- **Multi-language support.** Internationalise prompts and evaluation questions to support games with non-English interfaces.

---

1 4 5 Stagehand: A browser automation SDK built for developers and LLMs.
https://www.stagehand.dev/

2 6 Launching Stagehand v3, the best automation framework
https://www.browserbase.com/blog/stagehand-v3

3 7 How to build AI Agents with Vercel and the AI SDK
https://vercel.com/guides/how-to-build-ai-agents-with-vercel-and-the-ai-sdk