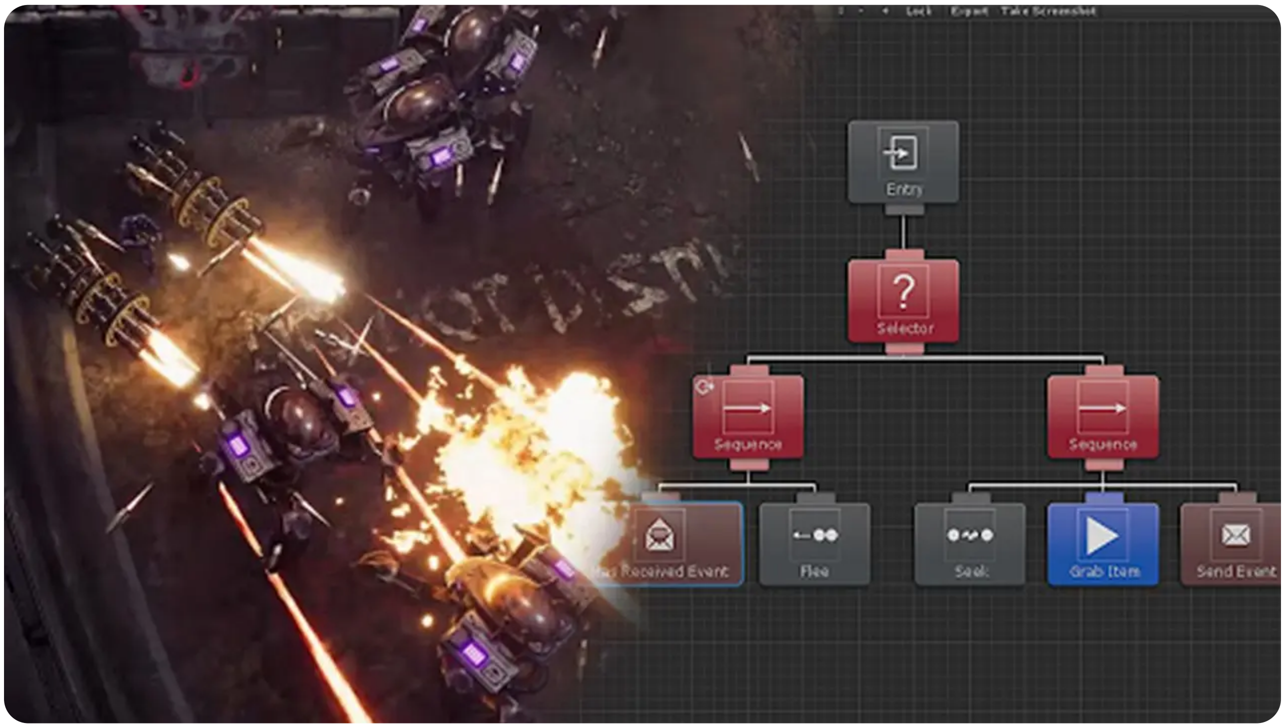
GAMES

Automate your playtesting: Create virtual players for game simulation

DYLAN SCANDINARO /
Anonymous

DEC 11, 2020 | 6 MIN



It's easy to automate playtesting by creating a Virtual Player (a game-playing agent), then using Game Simulation to run automated playtests at scale. Read on to discover three case studies describing how iLLOGIKA, Furyion, and Ritz Deli created Virtual Players – offloading nearly 40,000 hours (~4.5 years) of automated playtesting to Game

Games are challenging to test for the same reason that they're fun – players have the freedom to shape their own experience. Thus, games have a huge surface area for bugs and design flaws to appear. Developers must test frequently and with extensive coverage to resolve issues and reliably meet deadlines.

In the past, developers chose between low-coverage tests with high frequency (unit tests in a CI pipeline) and high-coverage tests with low frequency (playtests before a major release).

We built Unity Game Simulation to help developers test with the coverage of playtests *and* the frequency of unit tests. Unity Game Simulation enables developers to run automated playtests in the cloud. To use Unity Game Simulation:

Create a Virtual Player (a game-playing agent).

Use the Game Simulation package to instrument your game for simulation.
- Implement remotely configurable parameters to simulate different variations of your game.
- Implement metrics to record the data needed to answer design questions.

Use the Game Simulation package to create and upload a build of your game to our servers.

Run your game thousands of times from the Game Simulation user interface (UI).

This blog post focuses on step 1 – creating a Virtual Player for automated testing. Steps 2–4 are straightforward and covered in the Game Simulation documentation. You can try Game Simulation for free now.

Creating a Virtual Player for Unity Game Simulation is easy

A Virtual Player emulates the input of a real player to test some aspect of your game. For simple tests, such as validating that your game can run for 60 minutes without triggering an exception, a Virtual Player can be as simple as a C# script with a few lines of code that launches a scene and performs random actions.

For more complex tests, such as verifying that all weapons are roughly equal in strength or that each level can be completed, a Virtual Player can be created with the same methods commonly used to create non-player characters (NPCs). These include:

- Heuristic scripts: A script with a very simple rule or algorithm

- Behavior trees: - A visual representation of a plan consisting of conditions and tasks
- Finite state machines: A script with a few states that the Virtual Player alternates between, for example, seek and attack
- Unity AI Planner: A visual planning framework with an intuitive Unity Editor UI
- Reinforcement learning and imitation learning using the Unity ML-Agents Toolkit: Check out how we created a Virtual Player for Jam City's *Snoopy Pop* using ML-Agents.

Below we highlight how three studios created Virtual Players, collectively offloading nearly 40,000 hours of playtesting to Unity Game Simulation. What's particularly noteworthy is that all three studios were able to gain immense value with Game Simulation while relying on relatively simple approaches for creating their Virtual Player.

Ritz Deli: Heuristic (greedy algorithm)

Indie studio Ritz Deli developed *Eraser Blast*, a linker-style puzzle game featuring over 50 characters, each with unique gameplay characteristics. Ritz Deli used Unity Game Simulation to run hundreds of simulations to ensure that each character generates increasing scores and coin counts as their XP level increases.

Ritz Deli's CTO and tech lead Eric Jordan needed to create a Virtual Player capable of solving linker-style puzzles. He implemented a Virtual Player with C# script based on a simple heuristic greedy algorithm. For *Eraser Blast*, the algorithm matches the longest possible chain of bubbles of the same type:

Create the set of all possible single bubble selections.

Select the bubble from the set of valid bubbles that has the greatest number of available matches.

Repeat steps 1 and 2 until no available bubbles are valid matches.

Update the metrics for the total score and coins rewarded.

iLLOGIKA: Scripted behavior tree

iLLOGIKA is the studio behind *Rogue Racers*, a player-versus-player (PvP) runner. Players create decks of cards that contain powerups that the player uses during a race. iLLOGIKA used Game Simulation to test every combination of cards to ensure that no card or deck is too powerful.

The developers at iLLOGIKA created a Virtual Player using a C# script:

Enable the Virtual Player to successfully navigate to the end of the race by performing raycasts to find upcoming obstacles and then avoiding them by switching lanes, ducking, or jumping.

Add a series of rules describing when to use cards based on the state of the game, including the player's current health, relative positions of the other players, their card abilities, etc.

For each action described in steps 1 and 2, choose an incorrect but possible action to account for the unpredictability of a real player.

Furyion: Behavior Designer behavior tree

Furyion is the developer of *Death Carnival*, a top-down shooter with a unique weapon socket system that enables the player to choose from over one hundred thousand possible combinations of weapon, ammo, and weapon module – each combination defining a unique gameplay experience.

Herbert Yung, founder and director at Furyion, used a behavior tree creation tool called Behavior Designer to create a Virtual Player to estimate the average time of level completion for each combination of weapon, ammo, and weapon module. Herbert then ran thousands of simulations with Unity Game Simulation to test each weapon socket combination, saving more than 600 hours of playthrough.

Herbert leveraged the intuitive Behavior Designer UI and many off-the-shelf tasks in Behavior Designer to create a Virtual Player:

If an enemy is in range, attack that enemy.

If no enemy is in range, move towards the exit until an enemy appears within range.

Repeat steps 1 and 2 until no enemies remain.

Navigate to the gate at the end of the level and once the level officially ends, call Application.quit().

For more information on how to create a bot with Behavior Designer, see the documentation on Behavior Designer’s Asset Store page.

Early access preview: Unity Game Simulation for QA testing

The Unity Game Simulation team is committed to helping you create Virtual Players for automated testing, starting with Virtual Players for QA testing. Reach out to us if you’d like to be among the first to try our new tooling and features to create Virtual Players for QA testing.

Get started

Find out more about getting started with Unity Game Simulation – you can even try it for free. And please reach out to the Game Simulation team with any questions.

한국어

Purchase

- Products
- Unity Ads
- Subscription
- Unity Asset Store
- Resellers

Education

- Students
- Educators
- Institutions
- Certification
- Learn
- Center of Excellence

Download

- Get Unity
- Download Archive
- Beta Program

Unity Labs

- Labs
- Publications

Resources

- Learn platform
- Community
- Documentation
- Unity QA
- FAQ
- Services Status
- Case Studies
- Made with Unity

Unity

- Our Company
- Newsletter
- Blog
- Events
- Careers
- Help
- Press
- Partners
- Investors
- Affiliates
- Security
- Social Impact
- Inclusion & Diversity
