Planning-based Toolchain for Automated Regression Testing of Video Games

Tomáš Balyo[1], Roman Barták[1,2], Tomáš Bílý[1,2], Lukáš Chrpa[1,3], Martin Čapek[1], Michal Červenka[1], Filip Dvořák[1], Stephan Gocht[4], Lukáš Lipčák[1], Viktor Macek[1], Dominik Roháček[1], Josef Ryzí[1], Martin Suda[1,3], Dominik Šafránek[1], Slavomír Švancár[1], G. Michael Youngblood[1]

[1]Filuta AI, Inc., 1606 Headway Cir STE 9145, Austin, TX 78754, United States
[2]Faculty of Mathematics and Physics, Charles University, Prague, Czechia
[3]Czech Technical University in Prague, Prague, Czechia
[4]Stephan Gocht - AI Software Engineering, Meißen, Germany
tomas@filuta.ai, batak@ktiml.mff.cuni.cz, tbily@filuta.ai, lukas.chrpa@cvut.cz, {mcapek, mcervenka, filip}@filuta.ai, stephan@drgocht.com, {llipcak, viktor, drohacek, josef, msuda, dsafranek, slavo, michael}@filuta.ai

Abstract

Regression video game testing, in a nutshell, deals with testing whether the game mechanics implemented in the game maintain their functionality even after updating the game code. Usually, regression testing is performed using test scripts. While effective, they require manual creation and frequent updates throughout development, making the process labor-intensive and error-prone. Automated planning can mitigate this burden by automatically generating and maintaining test scripts, as game mechanics can be specified in a planning domain model using the Planning Domain Definition Language (PDDL). Individual tests then need to specify initial states and goals. This demonstration presents a toolchain that allows for easy integration of planning into a game engine, executing and evaluating specified tests, and collecting detailed logs, telemetry data, and video recordings, allowing users to review test results efficiently.

Introduction

Automated testing with AI has been a rising research focus more recently with work that has focused on agent-based approaches that include navigation mesh path-finding (Shirzadehhajimahmood et al. 2021), reinforcement learning agents for finding design and environmental defects (Ariyurek, Betin-Can, and Surer 2019; Ferdous et al. 2022), reinforcement learning for load testing (Tufano et al. 2022), modeling of user interaction for boundary testing (Owen, Anton, and Baker 2016), search for test case generation (Ferdous et al. 2021b), and search for automated play testing (Ferdous et al. 2021a).

Video game testing, in general, provides a wide range of challenges. Regression testing, in particular, is crucial for maintaining the correct functionality of individual game mechanics throughout the development process. For human testers, this involves thousands of hours of repetitive work that is error-prone and costly. Test scripts are a usual way of automating regression tests, yet the scripts still require a lot of maintenance throughout the game development process.

As evidenced by Bram Ridder's (AI Programmer for Rebellion) keynote talk at the 2021 AIIDE Conference on "Improved Automated Game Testing Using Domain-Independent AI Planning" (Ridder 2021) and his 2021 GDC AI Summit talk "Automated Game Testing Using a Numeric Domain Independent AI Planner," planning techniques for game testing are beginning to be used in the games industry mixed in with calls for more AI automation of testing (Fray 2023). Volokh and Halfond (2023) proposed an automated approach for determining actions when conducting automated exploration for games. It is based on program analysis (slicing) of the game code. Although they are not using the usual planning formalism (like PDDL), they work with a symbolic representation of states and actions and rely on SMT (satisfiability modulo theories) solvers to determine the set of applicable actions in a given state.

In a nutshell, automated planning formalism (such as PDDL) provides a useful machinery that can automate regression game testing and mitigate the need for time-consuming and costly maintenance. In principle, game mechanics can be formulated as lifted actions in the PDDL model, and the game environment can be abstracted into a state space represented by predicates and numeric fluents. Each individual test is then specified by an initial state of the game (which is often shared for a whole batch of tests) and a goal that needs to be achieved in the test. The planner then generates a plan (if it exists) that, in fact, can be treated similarly to a test script. On top of that, the planning technology can provide a certain level of robustness to changes in the game that might disrupt the plan while the test is executed by automatically generating a new plan (if possible).

Filuta AI's Gaming QA Toolchain

Our toolchain is designed in a modular way in order to reduce the effort for instrumenting the planning-based automated testing system into the game. The toolchain supports *Unity* and *Unreal* game engines that are widely used in the gaming industry and can be customised (with an additional effort) to custom gaming engines. The main components of our toolchain are described in the following paragraphs.

The *Planning Agent* component is responsible for generating plans, executing these plans, and then reporting the outcome of their execution (e.g. passed or failed) as well as telemetry data. The planning agent component requires a domain model that captures the mechanics of the game, in the PDDL language (Ghallab et al. 1998), that has to be engineered by an expert. Notably, a single model can be used for

a (large) class of tests, and for simpler games, a single model should be able to capture all tested game mechanics. We currently support STRIPS, conditional effects, quantified preconditions and effects, and numeric fluents. The planning agent utilizes the Unified Planning framework (Micheli et al. 2025) to interface with state-of-the-art planners to generate plans. The initial state of the game is obtained directly from the game (via a middle layer), and test goals, specified by a user, are obtained from the dashboard. The planning agent can also work in a *random walk* mode that allows for (random) exploration of the scenario (without specifying goals) to find less common bugs that might arise from a non-standard (but feasible) combination of actions.

The *dashboard* component provides the main user interface in which human test engineers can specify and evaluate tests for their games. Through the dashboard, the user can specify new tests by selecting a domain model and specifying which predicate instances need to be true or false at the end of the test. When a specified test is executed, the dashboard monitors the execution and provides information on which actions in the plan were executed, which action is currently being executed, and what actions are yet to be executed. After a test is completed, it is either marked as successful if all actions were executed and the goal was reached, or as failed if an issue occurs (e.g. an action cannot be executed). Completed tests can be investigated in more detail by looking at the reasons for test failure (if any), watching the in-game video from the test, and exploring a range of telemetrics (e.g. CPU usage, memory consumption, FPS) arranged in a timeline of the plan execution (e.g. that a specific action led to high CPU consumption).

To connect the planning agent to the game, we use the *middle-layer* component. The middle layer component can access the current game data and is responsible for mapping it to planning states (that are specified via the PDDL model). Then, the middle layer is responsible for executing the planning actions (specified in the PDDL model) in the game. Often, each predicate as well as each lifted action requires a single block of code in the middle-layer. The practical aspect of the middle layer is that it does not require changes in the game code as well as in the planning agent code, which, in consequence, makes the instrumentation easier.

Onboarding our Toolchain

When a new game is onboarding for our toolchain, there are several steps that need to be followed. Initially, it is important to define a *testing scope* that, in a nutshell, consists of mechanics of the game that will be tested and types of goals for regression tests. For example, one might want to test whether it is possible to build a unit in a strategy game. Besides building a unit, additional game mechanics such as building buildings and collecting resources might be involved (as they are essential for building a unit). Testing scope has to be also defined for manual or script-based testing and hence we believe that our toolchain does not introduce considerable overheads for this step for game developers.

The second step involves designing and developing a *planning domain model* that captures the game mechanics specified within the testing scope. It is an iterative process between game developers and planning experts that refine the testing scope into a symbolic specification of the states of the environment and actions. The challenging part in the domain design is to properly abstract game states into symbolic ones (e.g. by specifying predicates and numeric fluents) and to properly specify actions such that they can be (correctly) executed in the game and their outcome can be (correctly) evaluated. This step indeed introduces overheads for game developers as they need to interact with planning experts. Arguably, knowledge engineering in planning is still somehow a "black art" and the effectiveness of the process relies on planning experts (McCluskey, Vaquero, and Vallati 2017). On the other hand, often a single planning domain model can handle hundreds of test cases (that are run each time the game code is updated), and is usually resilient to a range of smaller changes in the game (e.g. different level design, new units).

The development of the middle layer can be done by game developers (with the help of planning experts). It might not introduce large overheads since the knowledge gathered during the design and development of the domain model can be leveraged in this case.

Regression tests are run frequently, and unless there is some major change in the game, their setup does not require maintenance. Although onboarding our toolchain introduces some overheads, in the long run, the toolchain saves a lot of manual effort in regression game testing.

Demo Video

The video describing our toolchain can be found at: https://youtu.be/VrN-Fv56fo4. The video provides an overview of the features of our toolchain and demonstrates its functionality on two games: FPS game Lyra (Epic Games 2025), a sample tutorial project developed alongside Unreal Engine, and RTS game Silica by Bohemia Interactive (Bohemia Interactive 2025), developed on the Unity engine.

Future Work

Our plans for future work involve formal verification of plan execution that would go beyond our current plan execution monitoring (e.g., verifying desired behavior of game mechanics), and integration of other methods (e.g., Monte-Carlo Tree Search, Reinforcement Learning) that can better capture game mechanics such as combat.

References

Ariyurek, S.; Betin-Can, A.; and Surer, E. 2019. Automated video game testing using synthetic and humanlike agents. *IEEE Transactions on Games*, 13(1): 50–67.

Bohemia Interactive. 2025. Silica Game. https://silicagame.com/. Accessed: 2025-05-15.

Epic Games. 2025. Lyra Starter Game - Unreal Engine Marketplace. https://www.unrealengine.com/marketplace/learn/lyra. Accessed: 2025-05-15.

Ferdous, R.; Kifetew, F.; Prandi, D.; Prasetya, I.; Shirzadehhajimahmood, S.; and Susi, A. 2021a. Search-based automated play testing of computer games: A model-based approach. In *International Symposium on Search Based Software Engineering*, 56–71. Springer.

Ferdous, R.; Kifetew, F.; Prandi, D.; Prasetya, I. S. W. B.; Shirzadehhajimahmood, S.; and Susi, A. 2021b. Search-Based Automated Play Testing of Computer Games: A Model-Based Approach. In *Search-Based Software Engineering: 13th International Symposium, SSBSE 2021*, 56–71. Springer-Verlag.

Ferdous, R.; Kifetew, F.; Prandi, D.; and Susi, A. 2022. Towards Agent-Based Testing of 3D Games Using Reinforcement Learning. In *Proceedings of the 37th IEEE/ACM International Conference on Automated Software Engineering*, 1–8.

Fray, A. 2023. Automated Testing Roundtables GDC 2023. https://autotestingroundtable.com/. (Accessed on 12/12/2023).

Ghallab, M.; Howe, A.; Knoblock, C.; Mcdermott, D.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL—The Planning Domain Definition Language.

McCluskey, T. L.; Vaquero, T. S.; and Vallati, M. 2017. Engineering Knowledge for Automated Planning: Towards a Notion of Quality. In Corcho, Ó.; Janowicz, K.; Rizzo, G.; Tiddi, I.; and Garijo, D., eds., *Proceedings of the Knowledge Capture Conference, K-CAP 2017, Austin, TX, USA, December 4-6, 2017*, 14:1–14:8. ACM.

Micheli, A.; Bit-Monnot, A.; Röger, G.; Scala, E.; Valentini, A.; Framba, L.; Rovetta, A.; Trapasso, A.; Bonassi, L.; Gerevini, A. E.; Iocchi, L.; Ingrand, F.; Köckemann, U.; Patrizi, F.; Saetti, A.; Serina, I.; and Stock, S. 2025. Unified Planning: Modeling, manipulating and solving AI planning problems in Python. *SoftwareX*, 29: 102012.

Owen, V. E.; Anton, G.; and Baker, R. 2016. Modeling user exploration and boundary testing in digital learning games. In *Proceedings of the 2016 conference on user modeling adaptation and personalization*, 301–302.

Ridder, B. 2021. Improve Automated Game Testing Using Domain Independent AI Planning - YouTube. https://www.youtube.com/watch?v=2KXmxuCjjCw. (Accessed on 12/12/2023).

Shirzadehhajimahmood, S.; Prasetya, I.; Dignum, F.; Dastani, M.; and Keller, G. 2021. Using an agent-based approach for robust automated testing of computer games. In *Proceedings of the 12th International Workshop on Automating TEST Case Design, Selection, and Evaluation*, 1–8.

Tufano, R.; Scalabrino, S.; Pascarella, L.; Aghajani, E.; Oliveto, R.; and Bavota, G. 2022. Using reinforcement learning for load testing of video games. In *Proceedings of the 44th International Conference on Software Engineering*, 2303–2314.

Volokh, S.; and Halfond, W. G. 2023. Automatically Defining Game Action Spaces for Exploration Using Program Analysis. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 19(1): 145–154.