PDSim: Simulating Classical Planning Domains with the Unity Game Engine

Emanuele De Pellegrin, Ronald P. A. Petrick
Edinburgh Centre for Robotics
Heriot-Watt University
Edinburgh, Scotland, United Kingdom
{ed50,R.Petrick}@hw.ac.uk

Abstract

The solution of a classical planning problem consists of a sequence of actions mapping the initial state to the goal state. The output of a plan is often provided as raw text, which can be difficult to follow and interpret, especially with large plans. Simulating a plan generated by an automated planner using visual feedback, such as animations of 3D models and environments, can be an important tool for quickly evaluating the quality of a plan and improving the design of planning domains and problems. In this system demonstration, we present the latest version of PDSim, an external tool that can be installed on the Unity game engine adding support for the simulation of classical plans using 3D animations and visualisation methods that can be defined by the user.

Introduction

Modelling planning domains and verifying plan correctness can be challenging tasks, especially for real-world problems. Although a plan could be valid, relying solely on planner output won't always help the user catch domain modelling errors that may be apparent when displayed in a more intuitive form such as a 2D or 3D visualisation (Chen et al. 2020). While systems that use such methods to illustrate the generated plan do exist (Vrakas and Vlahavas 2005; Vaquero et al. 2007; Chen et al. 2020; Tapia, San Segundo, and Artieda 2015; Muise 2016; Magnaguagno et al. 2017; Le Bras et al. 2020), PDSim (De Pellegrin 2020) uses the infrastructure and components of the Unity game engine (Unity Technologies 2020) to deliver a visualisation of the planning problem, that can help spotting errors or wrong definitions of the planning domain. With PDSim, the user can customise the model that represents the objects of the planning problem, as well as define the animations. This may include moving an object onto another object, letting the agent follow a path between two points on a map, playing a sound every time a particular condition is met, spawning a particle system, etc. The user is able to create real-world scenes that reflect the execution environment of the planning problem, exploiting the functionalities of the Unity game engine to be used as an automated planning tool.

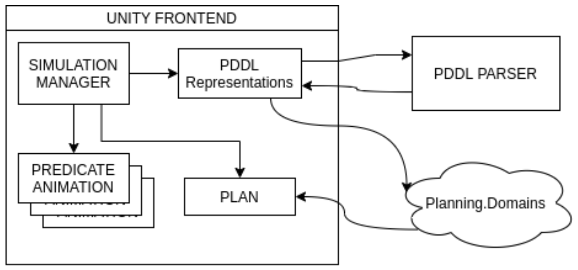

Figure 1: PDSim system architecture.

PDSim System

PDSim (De Pellegrin 2020) has been developed as an external module for the Unity game engine, a 2D and 3D game environment widely used in the video game industry. Unity offers an intuitive Graphical User Interface (GUI) editor and many available components such as a built-in physics engine, realistic shaders and materials, and a path planning library. Thanks to its modularity, it is possible to extend the interface to fit user needs, for instance by defining custom animations for PDDL objects or by extending existing animations to reuse models from different simulations without creating them from scratch every time. A general overview of the system architecture is given in Figure 1, showing the individual core components and the communication between them. In Example, the simulation manager is the main component that handles the animations and serialization of the PDDL problem. The Predicate Animations components defines the specific animations for each PDDL predicate the user wants to animate. The Unity front-end (editor) makes use of the Planning.Domains (Muise 2016) web services to generate the plan with the domain and problem files the user provides.

Plan Visualisation

The text output from an automated planner is converted into 3D animations, as illustrated in Figures 2-5. Figure 2 shows the Blocks World domain and the stacking animation of blocks during plan execution. Figure 3 shows the plan

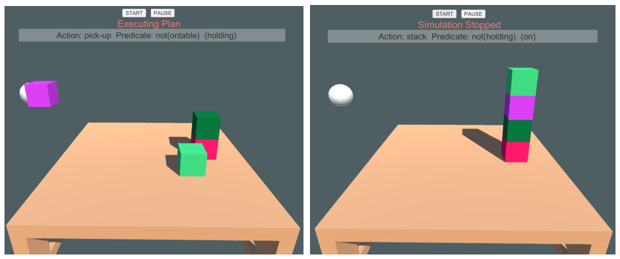
START PAUSE
Executing Plan
Action: pick-up Predicate: not(ontable) (holding)
START PAUSE
Simulation Stopped
Action: stack Predicate: not(holding) (on)

Figure 2: Blocks World plan execution

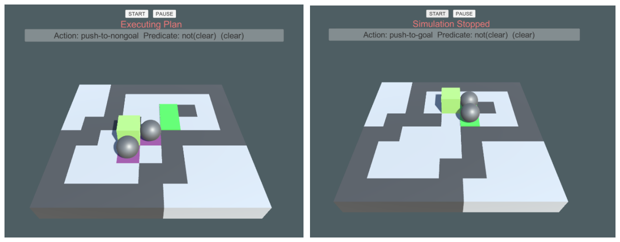
START PAUSE
Executing Plan
Action: push-to-nongoal Predicate: not(clear) (clear)
START PAUSE
Simulation Stopped
Action: push-to-goal Predicate: not(clear) (clear)

Figure 3: Sokoban plan execution

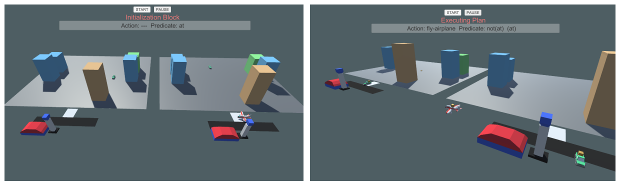

Figure 4: Logistics plan execution

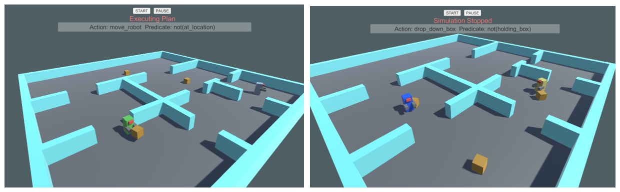
START PAUSE
Executing Plan
Action: move_robot Predicate: not(at_location)
START PAUSE
Simulation Stopped
Action: drop_down_box Predicate: not(holding_box)

Figure 5: Robots and Boxes plan execution

animation for the Sokoban domain, where the green cube represents the player and the grey spheres the stones being pushed to the goal tiles. Figure 4 shows the plan animation for the Logistics domain with custom models of cities, airports, trucks and aeroplanes. Figure 5 shows a custom domain called Robots and Boxes where robots need to pick up and drop boxes in specific rooms defined in the problem. This domain was created to demonstrate the application of Unity's path planning system for animating the movements of the robots. Finally, Figure 6 shows the importance of planning simulation to discover errors in the domain model, even though the planner can generate a plan. Here, the Blocks World domain has been purposely modified to introduce errors in the stack action. A plan is found but it is not valid: at the end of plan execution a block is floating in mid-air, suggesting a possible error in the domain definition.

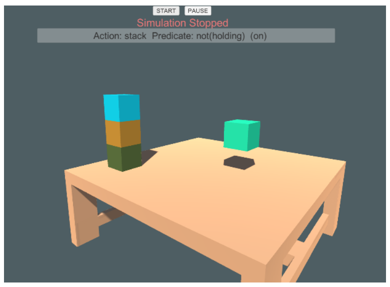

Figure 6: Plan validation check in PDSim.

Conclusion

We described the core features of PDSim, an extension to the Unity game engine for simulating plans. PDSim focuses on user interaction to specify the animations and models relative to a given PDDL domain and problem. In this system demonstration, we will present the main PDSim front-end, consisting of the Unity interface navigation, and the important components of the system, such as creating custom animations from scratch and the key aspects of the Simulation Manager. The demo will also cover the future work in plan for the system and the main changes such as: the temporal planning support, a new parser and a new workflow for creating animation definitions. The system demonstration video is hosted on YouTube[1].

References

Chen, G.; Ding, Y.; Edwards, H.; Chau, C. H.; Hou, S.; Johnson, G.; Sharukh Syed, M.; Tang, H.; Wu, Y.; Yan, Y.; Gil, T.; and Lipovetzky, N. 2020. Planimation. doi:10.5281/zenodo.3773027.

De Pellegrin, E. 2020. PDSim: Planning Domain Simulation with the Unity Game Engine. In *Proc. ICAPS Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*.

Le Bras, P.; Carreno, Y.; Lindsay, A.; Petrick, R. P. A.; and Chantler, M. J. 2020. PlanCurves: An Interface for End-Users to Visualise Multi-Agent Temporal Plans. In *Proc. ICAPS Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*.

Magnaguagno, M. C.; Fraga Pereira, R.; Móre, M. D.; and Meneguzzi, F. R. 2017. Web planner: A tool to develop classical planning domains and visualize heuristic state-space search. In *ICAPS Workshop on User Interfaces and Scheduling and Planning*.

Muise, C. 2016. Planning.domains. ICAPS System Demonstration.

Tapia, C.; San Segundo, P.; and Artieda, J. 2015. A PDDL-based simulation system. In *Proceedings of the IADIS International Conference Intelligent Systems and Agents*.

Unity Technologies. 2020. Unity. URL https://unity.com.

Vaquero, T. S.; Romero, V.; Tonidandel, F.; and Silva, J. R. 2007. itSIMPLE2.0: An Integrated Tool for Designing Planning Domains. In *Proceedings of ICAPS*, 336–343.

Vrakas, D.; and Vlahavas, I. 2005. A Visualization Environment for Planning. *Int. J. Artif. Intell. Tools* 14(6): 975–998.

[1]PDSim demonstration video: https://youtu.be/5CRYWIcFIw4