What Plan? Virtual Plan Visualization with PDSim

Emanuele De Pellegrin, Ronald P. A. Petrick,

Edinburgh Centre for Robotics
Heriot-Watt University
Edinburgh, Scotland, United Kingdom
ed50@hw.ac.uk, R.Petrick@hw.ac.uk

Abstract

Modelling and verifying planning solutions is a challenging problem, especially in real-world domains. This paper presents an update on the development of the Planning Domain Simulation (PDSim) project, an asset for the Unity game engine to simulate plans in a 2D or 3D environment with custom animations and graphics effects. PDSim aims to provide an intuitive tool for users to define animations without the need to learn a new scripting language, in order to quickly evaluate the validity of planning models. Due to the scarcity of similar systems and tools, PDSim fills an important gap in the area of planning simulation and validation: simulating a planning problem using 3D graphics and animation techniques can help the user to quickly evaluate the quality of a plan and improve the design of the planning domain and problem. This paper presents an update of PDSim, including its aims as a system for automated planning, the current state of development, and future plans for the project.

Introduction

The task of modelling planning domains and verifying plan solutions can be a challenging problem, especially for real world scenarios. While representation languages like PDDL (McDermott et al. 1998) provide a standard way of representing planning models supported by a wide range of planners, it can be difficult to catch modelling errors due to the complexity of the knowledge that needs to be specified (e.g., definitions of state properties, actions, and objects) and the level of abstraction that is often required for ensuring the generation of tractable solutions.

Although several tools do exist to aid in the validation of planning domain models (e.g., VAL (Howey and Long 2003)), and formal plan verification methods are a growing area of research (Bensalem, Havelund, and Orlandini 2014; Cimatti, Micheli, and Roveri 2017; Hill, Komendantskaya, and Petrick 2020), approaches based on visualisation methods and visual feedback can also play an important role in addressing the problem of correctly modelling planning domains. Visual tools provide a natural and interactive environment for displaying, inspecting, and simulating aspects of the planning process, which can aid in plan explainability for human users (Fox, Long, and Magazzeni 2017).

PDSim (De Pellegrin 2020; De Pellegrin and Petrick 2021) introduced a system to visualise and simulate plans for classical planning problems defined in PDDL. Although planning problem visualisation has been previously investigated (Vrakas and Vlahavas 2005; Vaquero et al. 2007; Chen et al. 2020; Tapia, San Segundo, and Artieda 2015; Le Bras et al. 2020), PDSim approaches the problem by building a graphical environment for plan visualisation and simulation within the Unity game engine (Unity Technologies 2020). PDDL is used to define the planning domain knowledge and problem formulation, including planner requirements, language models used in the domain, and standard definitions of the domain and problem. These components are used by a planner to establish that a solution exists and to generate a plan for the specified goals. Using the plan, PDSim interprets state properties and action effects as 2D or 3D animations and graphics effects to deliver a visual explanation of the world and its actions during plan execution. In this paper, we report on recent developments to PDSim, notably integration with the Unified Planning Framework (UPF) and several enhancements to PDSim's animation system.

The rest of the paper is organised as follows. First, we review related work and provide an overview of the PDSim system. We then describe recent additions to PDSim with an overview of the components and examples of their use. We conclude with a discussion of future PDSim development.

Related Work

PDSim (De Pellegrin 2020; De Pellegrin and Petrick 2021) is part of the small (but growing) ecosystem of automated planning simulators that use visual cues and animations to translate the output of a plan into a visual environment. The closest approach to ours is Planimation (Chen et al. 2020) which uses Unity as the front-end system to display and animate objects while following a given plan. Animations are defined using an ad-hoc language (an animation profile) similar to PDDL. PDSim removes this additional step with its animation system and provides a more intuitive system for users (see Figure 2, described in more detail below).

Other systems like vPlanSim (Roberts et al. 2021) and JEDAI (Shah et al. 2021) are similar applications that also aim to provide a 3D visualization of a plan, but with a number of important differences. For instance, while vPlanSim offers a simple and fast custom graphical environment for

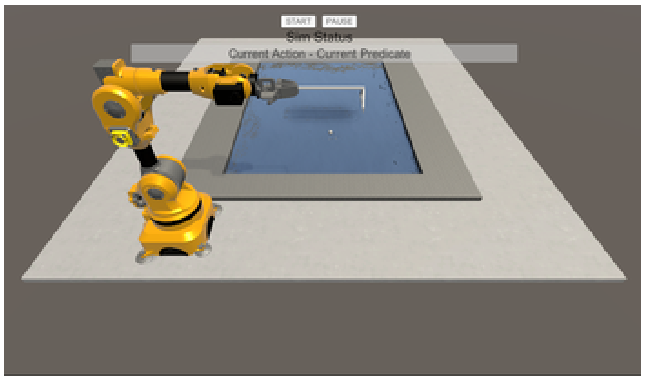

Figure 1: Robotarm PDSim simulation provided by Heemsekerk Innovative Technology[1].

creating plan simulations with few dependencies, PDSim is built using a popular game engine like Unity to offer the user industry standard tools to create real scenarios. PDSim offers a language agnostic tool to set up simulations which is paramount for users that are not familiar with PDDL and Unity. The approach used by JEDAI is also similar to that used by PDSim. JEDAI uses a block-based plan creation approach to let the user focus on the simulation; however, the simulations are restricted to the robot planning use case, while PDSim offers a more abstracted environment so the user can simulate problems of different types and complexity. For instance, Figure 1 shows a robotics simulation that uses PDSim as the main component to display a plan that involves a robot arm. The simulation visualises the plan generated for the robotic arm, involving the removal of solid impurities on the surface of a liquid metal bath.

The Logic Planning Simulator (LPS) (Tapia, San Segundo, and Artieda 2015) also provides a planning simulation system that represents PDDL objects with 3D models in a user-customisable environment. The approach is integrated with a SAT-based planner and a user interface that enables the execution of a plan to be simulated while visualising updates to the state of the world and individual PDDL properties. Unlike PDSim and Planimation, LPS is not based on Unity but provides its own interface for plan visualisation. Several user-specified files are also required to define 3D object meshes, the relationship between PDDL elements and 3D objects, and specific animation effect to be produced.

Several systems also exist to help formalise planning domains and problems through user-friendly interfaces. For example, GIPO (Simpson, Kitchin, and McCluskey 2007), It-Simple (Vaquero et al. 2007), and VIZ (Vodrázka and Chrpa 2010) use graphical illustrations of the domain and problem, removing the need for PDDL knowledge to help new users approach planning domain modelling for the first time. Other software such as Web Planner (Magnaguagno et al. 2017) and Planning.Domains (Muise 2016) use Gantt charts or tree-like visualisation methods to illustrate the generated plan and the state space searched by a particular planning algorithm. PlanCurves (Le Bras et al. 2020) uses a novel

[1]https://heemskerk-innovative.nl/

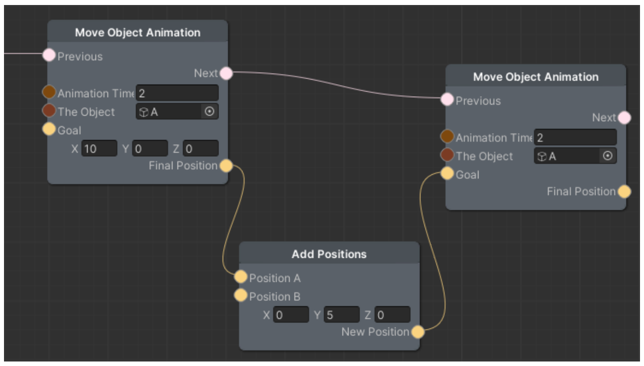

Figure 2: PDSim animation system.

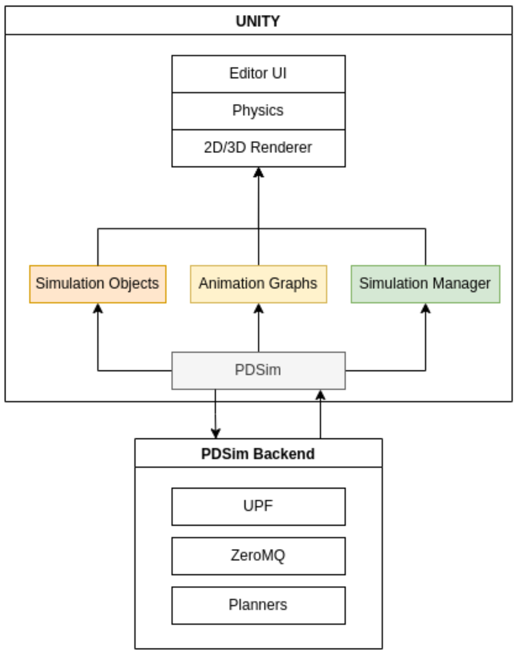

Figure 3: High-level PDSim system architecture.

interface based on time curves (Bach et al. 2015) to display multi-agent temporal plans distorted in timelines to illustrate similarity between states. All of these tools attempt to help users understand how a plan is generated and detect potential errors in the modelling process.

Simulators using a game engine such as MORSE (Echeverria et al. 2011) or Drone Sim Lab (Ganoni and Mukundan 2017) are also prevalent in robotics applications. A game engine offers benefits like multiple cameras to follow the simulation, a physics engine and realistic post-processing effects with no need to implement them from scratch (Ganoni and Mukundan 2017). PDSim is built by extending the Unity game engine editor (Unity Technologies 2020) and uses components offered by the engine, such as a path planner, a lighting system, and scene management, among others.

PDSim System Architecture

The PDSim system can be imported into Unity3D as a common asset. The Unity editor interface is used to interact with

PDSim components such as setting the simulation scene, creating animations, or importing 3D or 2D models. PDSim relies on its Python back-end implementation that is used to parse the PDDL files and generate plans. The high-level structure of the PDSim system is shown in Figure 3.

A simulation is initialized and handled by the back-end server running the Unified Planning Framework (UPF; see below) which is responsible for parsing and building a JSON representation of the planning model. UPF also handles calls to external planners to generate plans. UPF is a planner-agnostic framework for Python, which increases PDSim's modularity and lets users select their preferred planner implementation, separating it from the simulation stage itself which comes later in the process.

Several PDDL components are key to simulating a planning problem, including: predicates, actions, types, and initial values. Unity uses those components differently to represent PDDL in a graphical environment. The domain file is used to build the core components and the animations for the simulation. The types and objects define *SimulationObjects*, the visual aspect of the simulation: 3D models or 2D sprites. Predicates are used to define the *AnimationGraphs*, internal visual scripting language to define common transformation operations, path planning, audio emission, particle effects, etc. For instance, Figure 2 shows an animation definition for a predicate in the form of a stacking of 2 translation animations where the same object is translated along the x axis first and the y axis subsequently. Action effects are the animated components, where every predicate in the effects list that has an associated animation graph will execute an animation at simulation time. Finally, the initial values are used during simulation time to set up the scene. Similar to the animation effects, all the grounded values from UPF are animated if the predicates are associated with an animation.

PDDL files are translated into a JSON map of the components needed for simulation. PDSim uses components of the domain to set up the core simulation in Unity. Figure 4 shows the JSON code for the logistics domain, used to establish the internal definitions of actions, types, and predicates. The problem components of PDDL are used to set up a Unity level or scene, as in Figure 5. Later in the Unity editor, the user can configure multiple problems for the same domain and, thus, multiple simulations for different plans.

In Unity, the user can set 2D or 3D models for the constants defined in the planning problem, create animations for predicates and use Unity's internal components such as the physics engine, planning system, etc. PDSim communicates with UPF using the ZeroMQ library[2], in particular the Python implementation package pyzmq[3] on the server side and the C# implementation netMQ[4] on the Unity side.

Figure 6 shows the workflow executed by the system when the user wants to create a new simulation. The user interacts with PDSim,using the editor user interface to create a form in which the domain and problem files are expected. Unity tries to connect, using a separate thread, with

[2]https://zeromq.org/

[3]https://pypi.org/project/pyzmq/

[4]https://github.com/zeromq/netmq/

```
{ 'predicates':
  {'in-city':
      {'args': ['place', 'city'],
       'arity':2}, ... },
 'actions':
  {'load-truck':
     {'effects':[
        {'args'['pkg', 'loc'],
         'fluent': 'at',
         'negated': true} ... ],
      'params':{'pkg':'package',
               'truck':'truck',
               'loc':'place'} } ... }
 'types':
  {'object':['city','place','physobj'],
   'place':['airport','location'] ...}
}
```

Figure 4: Example domain representation in JSON.

```
{'objects':
  {'apn1' : 'airplane'
   'city1' : 'city', ... },
 'init':
  {'at': [
     {'args': ['obj13', 'pos1']
      'value': true},
     {'args': ['obj23', 'pos2']
      'value': true}, ... ] ... }
}
```

Figure 5: Example problem representation in JSON.

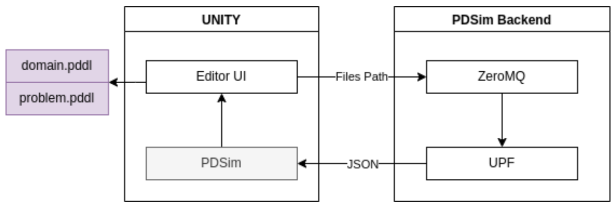

Figure 6: New simulation workflow in PDSim.

the back-end server by submitting a request using the PDDL domain and problem file paths. The request is sent with the "init" header to tell the server that there is a request to parse the PDDL and create a representation of the planning problem on the server to be used for later requests. If there are no parsing, syntax, or general server-side errors, Unity instantiates a new simulation scene. After scene creation, a second request is sent to get the JSON PDDL representation from the server, and all the PDSim components are initialized. Figure 7 shows the plan request interaction to generate a plan with the domain and the current simulation problem.

Unified Planning Framework (UPF)

One of the main extensions of PDSim is the wrapping of the Unified Planning Framework (UPF) project as the driver for handling and solving planning problems in PDSim. UPF is a Python library that aims to simplify the adoption of auto-

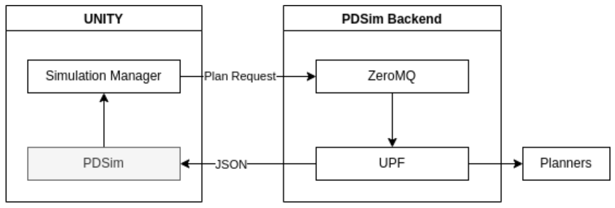

Figure 7: Plan generation workflow in PDSim.

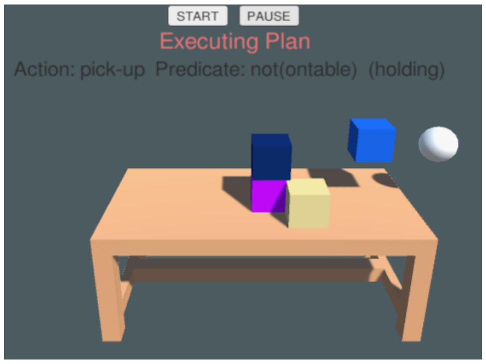

Figure 8: PDSim Blocks World simulation.

mated planning as AI technology, as part of the AIPlan4EU project[5]. The project aims to standardize the techniques used to solve a planning problem, making it accessible to users of any level of expertise. It offers a well-developed and maintained parser and a standard interface for communicating with external planners. Integration with UPF enables the PDSim system to take advantage of these features and any future updates that UPF may provide.

Examples

PDSim has been tested using the published benchmark domains for the International Planning Competition (IPC)[6]. We illustrate the complexity of the simulated planning problems here using Blocks World, Logistics, and Sokoban.

Blocks World: Blocks World (IPC 2000) is one of the simplest domains: blocks can be stacked on top of each other, and only one block can be picked, moved, and dropped at a time. The goal is achieved when the specified stack sequence is reproduced. Figure 8 shows an example of the Blocks World simulation in PDSim.

Logistics: Logistics (IPC 2000) describes a problem involving packages that need to be transported between cities using a airplane and within cities using trucks. This domain steps up the complexity of the simulation environment while keeping simple definitions of predicates and actions (e.g., the predicates *InCity*, *At*, *In* are used to respectively describe if a location is inside a city, if an object is in a location, and if a package is in a vehicle). Figure 9 shows the Logistics simulation, highlighting the animation of boxes and airplanes.

[5]https://www.aiplan4eu-project.eu/

[6]https://github.com/potassco/pddl-instances

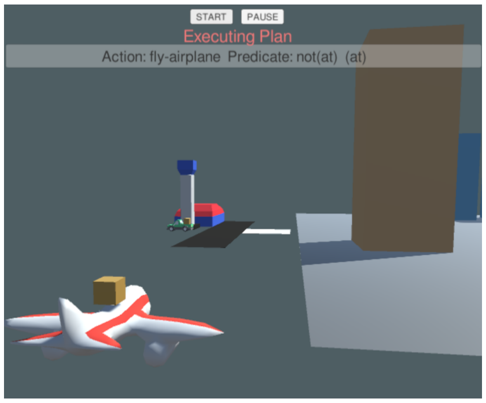

Figure 9: PDSim Logistics simulation.

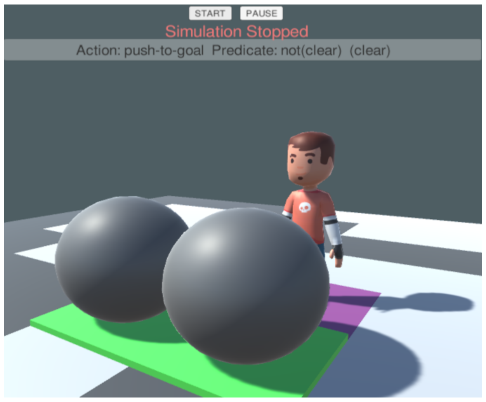

Figure 10: PDSim Sokoban simulation.

Sokoban: Sokoban (IPC 2008) describes the Sokoban game problem[7], where a player needs to move an object to a predefined goal on a grid. Figure 10 illustrates a typical problem level for Sokoban with a player and stone that needs to be moved. This domain adds to the complexity of the previous example, illustrating the functionality of this simulation in Unity, and its ability to rapidly provide an in-game AI agent.

Conclusion and Future Work

This paper presented an update on the structure and operation of PDSim, a simulation system for animating PDDL domains and plans. This project supports classical automated planning, however, current work is extending PDSim to support temporal planning through an intuitive visualization system for timed actions and deadlines. Future work on the project will consider support for partial plans defined by the user and other simulation features such as following simulation actions, replaying previous actions, modifying the simulation speed, and displaying partial animations to show the outcome of animation while defining the structure. An important direction for PDSim will also be to include extensions for visualising the current state of an agent's knowledge and beliefs to support epistemic planning.

[7]https://en.wikipedia.org/wiki/Sokoban

Acknowledgements

Thanks to Robertico Gustina, Delft University of Technology student who used PDSim for his Master's dissertation, and Heemsekerk Innovative Technology.

References

Bach, B.; Shi, C.; Heulot, N.; Madhyastha, T.; Grabowski, T.; and Dragicevic, P. 2015. Time curves: Folding time to visualize patterns of temporal evolution in data. *IEEE transactions on visualization and computer graphics*, 22(1): 559–568.

Bensalem, S.; Havelund, K.; and Orlandini, A. 2014. Verification and validation meet planning and scheduling. *International Journal on Software Tools for Technology Transfer*, 16: 1–12.

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 Nir, L. 2020. Planimation.

Cimatti, A.; Micheli, A.; and Roveri, M. 2017. Validating domains and plans for temporal planning via encoding into infinite-state linear temporal logic. In *Proceedings of AAAI*, 3547–3554.

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

De Pellegrin, E.; and Petrick, R. P. 2021. Automated Planning and Robotics Simulation with PDSim. In *Proceedings of the ICAPS Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*.

Echeverria, G.; Lassabe, N.; Degroote, A.; and Lemaignan, S. 2011. Modular open robots simulation engine: Morse. In *2011 IEEE International Conference on Robotics and Automation*, 46–51. IEEE.

Fox, M.; Long, D.; and Magazzeni, D. 2017. Explainable Planning. In *Proceedings of the IJCAI Workshop on Explainable AI*.

Ganoni, O.; and Mukundan, R. 2017. A framework for visually realistic multi-robot simulation in natural environment. *arXiv preprint arXiv:1708.01938*.

Hill, A.; Komendantskaya, E.; and Petrick, R. P. A. 2020. Proof-Carrying Plans: A Resource Logic for AI Planning. In *International Symposium on Principles and Practice of Declarative Programming (PPDP)*, 1–13.

Howey, R.; and Long, D. 2003. VAL's Progress: The Automatic Validation Tool for PDDL2.1 used in the International Planning Competition. In *Proceedings of the ICAPS Workshop on The Competition: Impact, Organization, Evaluation, Benchmarks*.

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 *Proceedings of the 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 (UISP)*.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL—The planning domain definition language. Technical Report CVC TR-98-003/DCS TR-1165, Yale Center for Computational Vision and Control.

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

Roberts, J. O.; Mastorakis, G.; Lazaruk, B.; Franco, S.; Stokes, A. A.; and Bernardini, S. 2021. vPlanSim: An Open Source Graphical Interface for the Visualisation and Simulation of AI Systems. In *Proceedings of the International Conference on Automated Planning and Scheduling*, volume 31, 486–490.

Shah, N.; Verma, P.; Angle, T.; and Srivastava, S. 2021. JEDAI: A System for Skill-Aligned Explainable Robot Planning. *arXiv e-prints*, arXiv–2111.

Simpson, R. M.; Kitchin, D. E.; and McCluskey, T. L. 2007. Planning domain definition using GIPO. *The Knowledge Engineering Review*, 22(2): 117–134.

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.

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.

Vodrázka, J.; and Chrpa, L. 2010. Visual design of planning domains. In *Proceedings of ICAPS Workshop on Knowledge Engineering for Planning and Scheduling (KEPS)*, 68–69.

Vrakas, D.; and Vlahavas, I. 2005. A Visualization Environment for Planning. *International Journal of Artificial Intelligence Tools*, 14(6): 975–998.