PDSim: Planning Domain Simulation with the Unity Game Engine

Emanuele De Pellegrin
Heriot-Watt University
Edinburgh, Scotland
ed50@hw.ac.uk

Abstract

PDSim is an extension for the Unity game engine that adds support for simulating a classical plan through the visualization of 3D animations of user-defined models and objects. With PDSim it is possible to define models for PDDL types and animations for actions and predicates in a 3D environment. This paper will present the main features of the PDSim system as well as its long-term goals.

Introduction

PDDL (McDermott et al. 1998) is used to define the structure of the domain of a planning problem, including the predicates, the possible actions and effects and the object's types in the environment. PDDL also formalizes the problem to solve along with the objects, their initial states and the final goal. These elements are used by automated planners to search for a solution. The solution of a classical planning problem consists of a sequence of actions, delivered in text form, mapping the initial state to the goal state. The output of a plan is provided as raw text, which can be difficult to follow and to interpret especially with large plans. Although the plan could be valid, relying only on textual feedback doesn't help the user catch domain modelling errors that may be more apparent when displayed visually (Chen et al. 2019). However there is a lack of visualizations tools able to validate a plan using graphical cues such as the simulators applied in robotics. Simulating a plan generated by an automated planner using visual feedback such as animation of 3D models and environments is important to quickly evaluate the quality of a plan and improve the design of planning domains and problems a user has in mind.

This paper presents the Planning Domain Simulation system (PDSim), an extension to the Unity game engine (Unity Technologies 2020) that allows the execution of a classical plan to be simulated from the PDDL definitions of the domain and problem. The system attempts to provide additional feedback by using a 3D environment to represent the PDDL elements. The action text from the plan is interpreted in 3D animations and effects with the help of the Unity game engine. Approaching planning and PDDL for the first time can be challenging especially for non-experts. PDSim is meant to improve the understanding of a generated plan rather than simply looking to the text output which can result difficult to track especially with a large number of actions and objects.

In the rest of this paper we present related work on the high-level formalization of PDDL and planning simulators, the technical overview of the system, and future work.

Related Work

Systems that help users formalize planning domains and problems through user-friendly interfaces are available. Systems like GIPO (Simpson, Kitchin, and McCluskey 2007), ItSimple (Vaquero et al. 2007) and VIZ (Vodrázka and Chrpa 2010) use graphical illustrations of the domain and problem elements, removing the requirement of the language knowledge to help new users approaching PDDL 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 visualization methods to illustrate the generated plan and the state space searched by a particular planning algorithm. These tools assist both new and experienced users to understand how a plan is generated and help to monitor semantic errors while improving the efficiency of domain development process (Magnaguagno et al. 2017).

Planimation (Chen et al. 2019) and LPS (Tapia, San Segundo, and Artieda 2015) are simulation systems that attempt to animate, using visual cues, a plan from PDDL specifications. Planimation uses a 2D canvas to represent PDDL objects and animate their position in space, visually explaining the plan to the user (Chen et al. 2019). LPS instead represents PDDL objects with 3D models in a user-customizable environment (Tapia, San Segundo, and Artieda 2015). While LPS use a 3D environment and model to simulate a plan that the user need to define in a separate file and Planimation offers a friendly user interface to customize the animations, they both don't offer a complete UI editor and tools that a game engine can offers, and the feasibility to use them in the simulation of a plan. An editor can help in represent and model the PDDL elements and abstract on a high-level the definition of animations for the models, and tools like path planning, particles, sounds and lights systems might help in enhance the realism and the understanding of a simulation.

Simulators using a game engine such as MORSE (Echeverria et al. 2011) or Drone Sim Lab (Ganoni and Mukun-

dan 2017) are prevalent in robotic applications. A game engine offers benefits like multiple cameras outputs, a built-in physics engine and realistic visual effects without the need to implement them from scratch (Ganoni and Mukundan 2017). PDSim is built on top of the Unity game engine (Unity Technologies 2020) to extend the capabilities of the existing planning simulators. The Unity game engine's editor was extend to integrate the PDDL paradigms, and to offer the user-friendly interface for setting up the simulation. Similarly to Planimation the user can set up the animation using an high-level UI, but it's possible to use the unity tools formerly introduced. Likewise, LPS offers a simulation in a 3D world but doesn't incorporate an editor to easily set up the 3D models for the PDDL objects and the simulation scene. The Unity editor, offers a 3D scene building window where the user can fully customize the planning environment.

Technical overview

Unity is a 2D and 3D game engine mainly use to develop interactive video games. Unity offers a fully customizable editor and many different tools such as a built-in physic engine (nVidia PhysX), support to shaders and textures for models, a path planning system, to rapidly develop an idea without focusing on the core implementation of them. This approach is also enhanced by the presence of a user-friendly scripting language such as C# which can be used to define custom behaviours for every object in the unity scene or editor. C# is a popular object-oriented programming language developed and maintained by Microsoft. Every object in the scene or editor can be scripted, meaning its possible to specify custom behaviours for the object and custom user interface (UI) layouts and components for the editor. The passive benefits Unity also offers are:

- Documentation: The full engine API is available online with several working examples.
- Easy Learning Curve
- Active community of developers
- Low Cost: Unity is free for personal use and for students.

PDSim has been developed as a custom module for the Unity game engine. Thanks to its modular nature it is possible to extend the framework to fit the user needs, such as to define custom animations for PDDL object by extending the existing ones or reusing models and animations into different simulations without creating them from scratch every time. Unity has facilitated the implementation of modules like the physics simulation, models rendering, the agent path planning which are available in-engine.

To achieve modularity the simulator uses Unity's scriptable objects[1] component to set up the simulation. A scriptable object serves as a data and code container, and it's specifically used to reduce memory usage of the system as it can behave like a small scale database.

In PDSim the GenericObject class represent each PDDL type in the 3D scene. It holds information about the object transformation in the 3D space (position, rotation and scale), the mesh renderer and a set of methods responsible for the animation. The user can customize each generic object, for example, to let it use the path planning component for the movement animation or having a random or custom colour. PDSim finally creates the representation of the PDDL objects using the user definitions of the respective generic object. Each object created is labelled to map the name of the object in the problem file and stored in a global dictionary to be used at run-time.

The predicates defined in the domain file are responsible for the animation itself. The class *CommandBase* store settings and code to animate the PDDL predicates in the simulation. When the user wants to define his custom predicate animation it can override the functions:

- PreActivate: that will run before the animation
- Positive: that will run when the predicate represents a positive fact
- Negative: that will run when the predicate represents a negative fact
- PostActivate: that will run before the animation completes

It is possible to define behaviours based on the number of attributes of a specific action. To help the user 3 main classes are available to extend from: *OneAttributeCommand*, *TwoAttributeCommand* and *ThreeAttributeCommand*. These classes expose respectively the objects X, Y and Z representing the attribute of the predicate, which are of type GenericObject that have the appropriate methods to perform transformations and animations. The predicate behaviours available for the user are actions for moving the object to a point in space (three-dimensional vector), moving an object to another object by specifying the target alignment or changing the defined model colour. The alignment the user can choose are 6 points representing the top, bottom, left, right, front and back vectors of the target object. The user can use these built-in methods to rapidly animate a movement behaviour and tuning the controls such as moving to another object using the path planning system (for objects like cars or humanoids), moving to a random point near to the target or change the colour of the object based on the predicate being true or false.

System Architecture

The general system architecture is illustrated in Figure 1. The Unity Editor represents the UI of Unity which is where the user interacts with the simulation. The UI has been extended to help the creation of simulations and the simulation objects formerly described. From the Unity editor, the user can inspect the domain and problem file using the PDDL editor component, which is a browser window showing the WebPlanner web app (Magnaguagno et al. 2017).

Every new simulation creates a Unity scene[2], an essential part of the editor where all objects and component are stored and managed. For each scene created, the PDDL Parser component is used to instantiate a simulation manager object

[1] https://docs.unity3d.com/Manual/class-ScriptableObject.html

[2] https://docs.unity3d.com/Manual/CreatingScenes.html

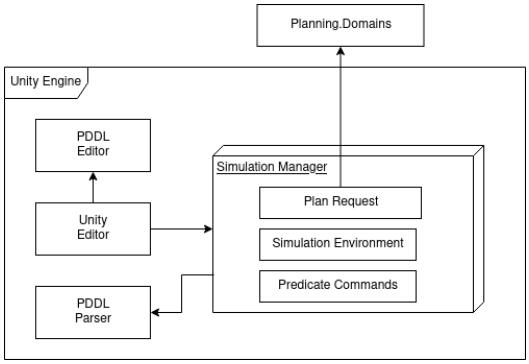

Figure 1: System Architecture

using the domain file, and a simulation environment object using the problem file.

The simulation manager holds the information about the actions, the predicates and the types. The simulation manager controls the whole simulation from the generation of the scene to the plan execution. When the Unity editor is in play mode, the simulation manager initializes the PDDL elements previously parsed and runs 3 main blocks:

- Plan Generation
- Initialization
- Plan Execution

During the plan generation, the simulation manager looks in the simulation environment object if a plan is already available. If the plan is not saved locally, the simulation manager will use the cloud planning service Planning.Domains (Muise 2016) to generate a new plan using the domain and problem file provided. The generated plan is parsed, and the actions are stored in a list. During the initialization, if a predicate in the list of the initialization blocks predicates has a mapped behaviour the animation will run in the scene. The same procedure occurs during plan execution, where the predicates checked are stored in the list of the plan action's effects.

The simulation environment stores the 3D models of the types, the objects, the initialization procedure and the positions of the objects in the scene modified by the user. The user can change the position, rotation or scale of the PDDL objects in the scene and save the environment object to store all the transformations defined by the user, or generate the scene using the models defined. The simulation environment is a modular component, meaning that a user can create several environments with problems that use the same domain, and simply swap them in the simulation manager.

Predicate commands are components that can be created in the Unity editor interface, used to generate animations for the PDDL predicates. Predicate commands can be drag and dropped in the specific predicate field in the simulation manager.

Case Studies

Examples of the plan execution using PDSim are illustrates in Figure 3, 4, 5 and 6. Figure 3 shows the Blocks World

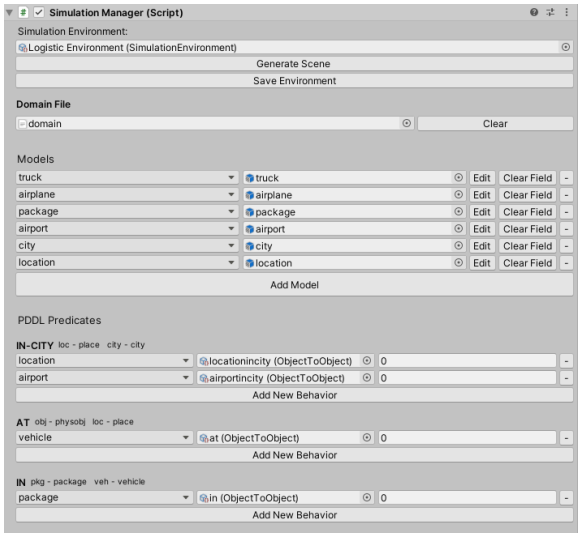

Figure 2: Settings Simulation Manager, Logistic Domain

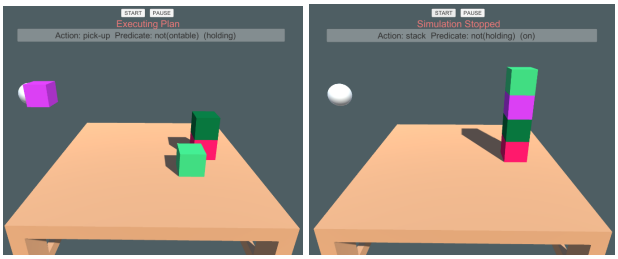

Figure 3: Blocks World plan execution

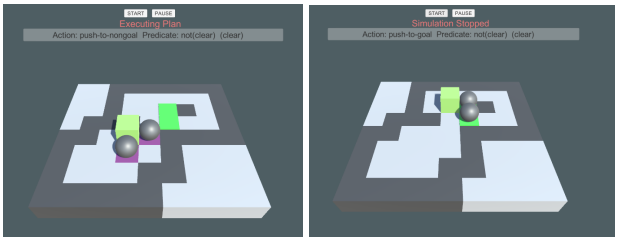

Figure 4: Sokoban plan execution

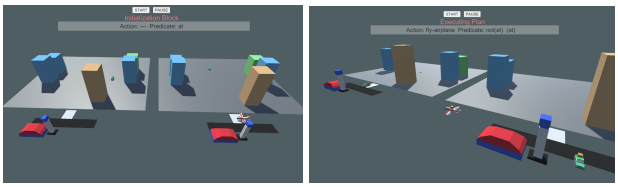

Figure 5: Logistic plan execution

domain, evidencing the stacking animation of blocks during the plan execution. Thanks to the simple structure of the blocks world domain, this has been used as starting reference for building the system. Figure 4 shows the plan animation for the Sokoban domain, where the green cube

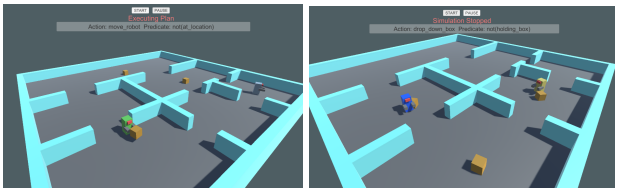

Figure 6: Robots and Boxes plan execution

represents the player and the grey spheres the stones being pushed to the goal tiles. Figure 5 shows the plan animation for the Logistic domain with custom models of cities, airports, trucks and aeroplanes. These components are defined by the user as illustrated in Figure 2 showing the user interface of the simulation manager where the user needs to drag and drop animations for the PDDL predicates and objects models for the PDDL types. These two domains were used to test the different mechanics of PDSim such as the concurrent multiple objects animations (the boxes moving with the van in the logistic and the stones moving with the player in Sokoban), and custom animation definition (the grid creation in Sokoban). Finally, Figure 6 shows a custom domain called Robots and Boxes where robots need to pick up and drop down boxes in specific rooms defined in the problem. This domain was created to demonstrate the application of Unity's path planning system used to animate the movements of the robots.

Conclusion

This paper presented the current development of PDSim, a simulator interface for PDDL as an additional assistant to PDDL domain modelling and as a supplementary tool for educational purposes. PDSim has been developed as part of a master dissertation project, it will be available as an open-source repository for the planning community. At this point of development, PDSim works with PDDL types restrictions. Without PDDL types, in PDSim the user can't define the objects and 3D models. For example the type *block* in blocks world is used to generate the unity object that will be cloned when creating the scene from the problem file by checking how many blocks the simulation needs. However, future implementations aim to remove this constraint. Future work on the system involves adding support for temporal and multi-agent planning, multiple simulation settings like the ability to proceed through the plan step by step, cameras that focus on the executed actions, partial animations and integrating with ROS and ROSplan (Cashmore et al. 2015). A parallel version is also in the roadmap to interface PDSim with the planning.domains web-based planner (Muise 2016) as an internal plugin.

Acknowledgements

Thanks to Dr Ron Petrick for his supervision and for providing feedback on this work.

Appendix

The system demonstration video is available at the following URL: PDSIM demonstration video

References

[Cashmore et al. 2015] Cashmore, M.; Fox, M.; Long, D.; Magazzeni, D.; Ridder, B.; Carrera, A.; Palomeras, N.; Hurtos, N.; and Carreras, M. 2015. ROSPlan: Planning in the Robot Operating System. In *Twenty-Fifth International Conference on Automated Planning and Scheduling*.

[Chen et al. 2019] Chen, G.; Ding, Y.; Edwards, H.; Chau, C. H.; Hou, S.; Johnson, G.; Syed, M. S.; Tang, H.; Wu, Y.; Yan, Y.; Tidhar, G.; and Lipovetzky, N. 2019. Planimation. ICAPS system demonstration.

[Echeverria et al. 2011] 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.

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

[Magnaguagno et al. 2017] 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 2017 Workshop on User Interfaces and Scheduling and Planning (UISP)*.

[McDermott et al. 1998] 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 2016] Muise, C. 2016. Planning.domains. ICAPS system demonstration.

[Simpson, Kitchin, and McCluskey 2007] 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, San Segundo, and Artieda 2015] 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 Technologies. 2020. Unity.

[Vaquero et al. 2007] Vaquero, T. S.; Romero, V.; Tonidandel, F.; and Silva, J. R. 2007. itSIMPLE 2.0: An integrated tool for designing planning domains. In *ICAPS*, 336–343.

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