Dynamic Scene Reconstruction for Prototyping Planning-Based Game Environments

Albaraa Ammar Othman[1,2], **Emanuele De Pellegrin**[2], **Ronald P. A. Petrick**[1]

[1]Department of Computer Science, Heriot-Watt University, Edinburgh, Scotland, UK
[2]School of Informatics, University of Edinburgh, Edinburgh, Scotland, UK
ao2000@hw.ac.uk, edepell@ed.ac.uk, R.Petrick@hw.ac.uk

Abstract

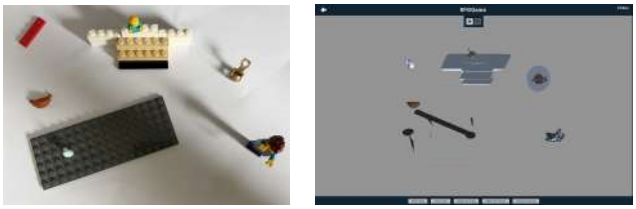
High-fidelity game prototypes are expensive and time-consuming to create which limits the level of user-testing carried out in early stages of game development. However, recent advancements in technologies focussing on the reconstruction of real world environments in a digital space introduce a different approach to these prototypes. This paper presents a human-in-the-loop framework which converts an image of a physical prototype into symbolic attributes populated by camera data and user definitions. These attributes are suitable for constructing a partial PDDL representation for planning and plan visualisation. We describe the approach and evaluate the framework using a series of item collection-based games, demonstrating the pipeline from initial physical prototypes to plan visualisation.

Introduction

Dynamic Scene Reconstruction is the process of creating a simulated version of a real world domain. While such environments are typically used for robotics-related tasks (Bavelos et al. 2025; Gu et al. 2023; Ni et al. 2025), they are also beneficial during the prototyping of games, for instance for tasks like user testing. User testing is invaluable to the success of a video game (Politowski, Petrillo, and Guéhéneuc 2021; Ramadan and Hendradjaya 2014) and such tests should be carried out as early as possible to attain full value. However, the creation of the necessary high-fidelity prototypes are both time-consuming and expensive (Stahlke, Nova, and Mirza-Babaei 2020).

In this paper, we consider the problem of automatically identifying elements of a scene and their respective spatial properties in order to build a symbolic representation that supports automated planning and plan visualisation. The task of segmenting scenes and identifying object properties has a long history in areas like vision, sensing and robotics (Ravi et al. 2024; Redmon et al. 2016; Vidal et al. 2023), with generative AI providing new tools and methods to aid in this process.

Automated planning focusses on finding a plan to reach a required goal state from a prespecifed initial state. Automated planning has previously been deployed for a variety of different tasks such as robotics, logistics, manufacturing and has a long history in games (James 2021; Ramirez and Bulitko 2015; Kelly et al. 2007; Duarte et al. 2020; Balyo et al. 2025a), but the success of these techniques often relies on the correctness of the underlying planning model.



(a) (b)

Figure 1: Example level prototype.

A camera is used to capture a top-down image of an environment and a segmentation model (SAM 2) (Ravi et al. 2024) is used to extract objects from the frame. Spatial properties for each element are determined from the camera data. This representation is then converted into a PDDL domain and model (McDermott et al. 1998), capable of being used with classical planning, providing a pipeline for converting continuous visual data into discrete symbolic data supporting planning model concepts. Given the induced planning model we reconstruct the scene digitally in the Unity Game Engine using the Planning Domain Simulation (PDSim) system (Pellegrin and Petrick 2024), an open source extension for Unity. To test the validity of the reconstructed prototype, plans are generated and executed which simulate level completion. Levels are prototyped primarily with LEGO bricks as seen in Figure 1, with these levels primarily focussing on item collection-based games in the 2D plane.

The rest of this paper is organised as follows. First, we review related work for scene reconstruction, game design, automated planning and plan visualisation. We then present our pipeline for extracting objects and object properties from an image, which is used to induce a partial PDDL planning model. This model is then used to dynamically reconstruct scenes in PDSim, which we then evaluate with a set of experiments. Lastly, we discuss our results and future work.

Background and Related Work

Scene Reconstruction Both manual and automated approaches to scene reconstruction exist. For example, manual

3D modelling may be practical for a scene with few items although as scenes become busier, manual approaches become unsustainable. More automated approaches exist which employ the use of high-end expensive 3D scanners or cheaper but less accurate phone-based applications. Typically these applications use algorithms which involve a blend of Structure from Motion (SfM) and Multi-View Stereo (MVS), which takes overlapping images and converts them into dense points clouds, i.e., an array of points each with a coordinate to recreate a scene (Verykokou and Ioannidis 2023). These approaches create a singular high visual fidelity mesh to represent a scene where all objects act as a singular background (Downs et al. 2022). An emerging technology, 3D Gaussian Splatting (3DGS) is a high-quality and real time rendering technique used to build 3D representations of objects and environments (Kerbl et al. 2023). Scenes are modelled as collections of 3D Gaussian ellipsoids, with each ellipsoid's colour, shape, position, and opacity optimised to reconstruct the scene across multiple camera viewpoints.

Environments typically take a graphical structure and are commonly represented with Scene Graphs (SG) consisting of nodes and edges, where each node represents an object and each edge a relationship between nodes (Rosinol et al. 2020). These graphs allow for rich semantic descriptions as they can represent object properties, descriptor values (e.g.,colour, material), and parent-child relationships (e.g., spatial relationships between objects) (Gu et al. 2023). In dynamic or unknown environments where scenes change in real time, the scene graphs can be updated with network or foundation models (Rosinol et al. 2020; Gu et al. 2023) that give rise to Dynamic Scene Graphs (DSGs).

Object Identification Object identification is the process of providing labels to objects in a scene, a task which has a long history in vision and sensing research (Dalal and Triggs 2005; Girshick 2015; Redmon et al. 2016). In small datasets, approaches that assign similarity scores between objects can be used, such as feature matching which identifies matching points between different images and can be undertaken using techniques like Scale-Invariant Feature Transform (SIFT), Speed up Robust Feature (SURF), and Oriented Fast and Rotated Brief (ORB) (Karami, Prasad, and Shehata 2017). For larger datasets, Machine learning (ML) techniques such as neural networks or clustering algorithms can be used to sort objects into distinct classes. ML approaches are trained on labelled data which allows them to estimate classes for new data but are generally successful in environments where all objects are previously known.

The vast landscape of generative tools such as Large Language Models (LLMs) and their visual counterpart, Vision Language Models (VLMs), introduce a new method for labelling objects in unknown environments. These models are trained on vast amounts of unlabelled, unsupervised data (Naik, Naik, and Naik 2024; Ferrara 2023; Zhiheng, Rui, and Tao 2023) offering the possibility of exceptional performance on the labelling task as the model can generalise and infer details about previously unknown or obscured objects (Raschka 2024). LLMs and VLMs also have the inherent ability to interpret and generate human-coherent content (Berrios et al. 2023), with LLMs specialising in text and VLMs able to perceive and generate images or video frames.

In busy scenes, the labelling tool can fail to identify all objects, so a method to extract meaningful regions, such as segmentation can simplify the labelling task (Othman, De Pellegrin, and Petrick 2026). In simple cases, segmentation can be completed by extracting colour and depth regions, or areas within a detected edge. For a more sophisticated approach in unknown environments, foundation models which specialise in segmentation such as Meta's Segment Anything Model (SAM) (Ravi et al. 2024) can be used.

Automated Planning Automated planning research addresses the problem of reasoning about action sequences to transform an initial state into a new state that achieves a set of goal conditions (Ghallab, Nau, and Traverso 2016). A planning problem can be defined as a tuple $\Pi = \langle P, A, I, G \rangle$, where P is a set of properties that define a state space (including a set of objects), A is a set of actions, I is a set of initial state properties, and G is the set of goal conditions. This definition can be considered in the context of a state transition system, where a state captures all the properties of P that are true, and actions in A transform states to new states. A solution to the planning problem is a sequence of actions (a plan) that when applied to the initial state I transforms it to a state where the properties of G are true.

Traditionally, languages such as the Planning Domain Definition Language (PDDL) (McDermott et al. 1998) are used to model planning problems, providing a standard, planner-independent representation language supported by a wide variety of planning engines. The Unified Planning Library (UPL) (Micheli et al. 2025) provides an alternative to standard PDDL specification through an API that enables external planners to be integrated in a Python-based wrapper. Tools like UPL also attempt to address problems of accessibility, by offering a traditional programming environment as an attempt to encourage the wider adoption of planning tools in research and industrial applications.

Plan Visualisation Plan visualisation is an important area of research exploring the development of tools for tasks like plan simulation and planning domain debugging. While still an under-explored area, tools like LPS (Tapia, San Segundo, and Artieda 2015), Planimation (Chen et al. 2020), and vPlanSim (Roberts et al. 2021) aid in the interpretation of planner outputs, allowing interaction and online inspection of objects and states during plan execution for debugging, analysis, and refinement of planning models (Gerevini and Saetti 2020).

Our work uses the Planning Domain Simulation (PDSim) system (De Pellegrin and Petrick 2024), an open source Unity-based extension for visualising planning domains. PDSim is an external plug-in for the Unity game engine, providing a user interface to 2D/3D graphics and animations, and a connection to external planners and PDDL language parsers using UPL. PDSim leverages Unity's underlying physics and animation system to create animations for PDDL-defined objects and properties. These animations can be used to set up a scene in Unity, allowing plans to be executed in the simulated virtual environment.

Game Design Game Engines inherently provide support for high-quality physics, rendering and integrability (Collins et al. 2021; Kargar et al. 2024; Wang, Han, and Tiwari 2021). For example, tools such as Unity (Unity Technologies 2025) and NVIDIA's PhysX (NVIDIA Corporation 2025) provide realistic graphics and accurate physics properties within a robust framework. These tools have been used to make a long list of award winning games.

The level of realism achieved within game engines has resulted in them being used in the field of robotics as they allow for safe, cost effective and scalable environments for robot development (Collins et al. 2021; Kargar et al. 2024). These tools are also used as digital twins—digital recreations of real-world environments and systems (e.g., robots)—which allow for digital counterparts of physical systems to be tested in realistic environments. For instance in tasks such as autonomous decision making and action execution (Verdouw et al. 2021).

Digital counterparts can also be used as preliminary prototypes for user-testing games. Traditionally, the construction and modification of these high-fidelity prototypes can be costly and time consuming (Stahlke, Nova, and Mirza-Babaei 2020). User testing is crucial to ensure a positive response from the public on first impression by resolving issues early in development and iteratively improving the game to reach satisfaction from the end user (Politowski, Petrillo, and Guéhéneuc 2021; Stahlke, Nova, and Mirza-Babaei 2020). In general, games full of bugs do not make it to the release stage resulting in a large financial loss for a company (Ramadan and Hendradjaya 2014). To evaluate game quality, game testers assess the 'fun factor' and search for bugs whereas software testers write code to automate source-code tests to verify multiple aspects such as whether a level is achievable (Ramadan and Hendradjaya 2014; Stahlke, Nova, and Mirza-Babaei 2020).

Planning in Games Video games are examples of rapidly changing dynamic environments where automated planning has been successfully applied in a number of occasions (Techland 2015; Studios 2012; Productions 2005; Games 2004; Entertainment 2011) to control non-player characters (NPCs) by replacing or in addition to traditional behaviour trees. In these environments, actions can have different results depending on the game state. Several goals can be active simultaneously and multiple agents can be using the same planner at any given time. Several approaches have been proposed to address these challenges such as learning heuristics of actions during runtime, putting goals in a hierarchy to prioritise certain goals over others, and using multiple planning domains to represent different aspects of a character. To do this, these games use different types of planners such as Goal Oriented Action Planning (GOAP) or Hierarchical Task Network (HTN) (Neufeld et al. 2017).

Planning has applications in other aspects of the video game design process. These include level design, where it can be used to create alternate combinations in levels at varying difficulties (James 2021). As mentioned earlier, game testing is crucial in the design process. Automated planning can expand the testing process by testing the validity of level completion through regression tests where game objects are encoded as planning model goal states (Balyo et al. 2025b).

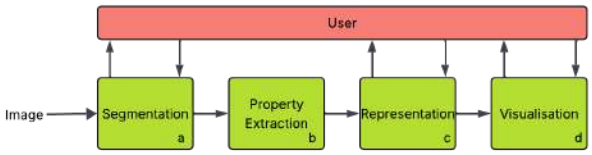


Figure 2: Level construction pipeline

Level Reconstruction Pipeline

We now present our pipeline for dynamically reconstructing physical prototypes in the Unity Game Engine as illustrated in Figure 2. The pipeline converts physical level prototypes into a symbolic representation and then into a digital prototype. This digital prototype can be used for level testing with automated planing or as a quick high-fidelity user test. This approach is intended for users with an understanding of traditional game design concepts such as game states, objects and actions. The knowledge of more technical planning related terms is not necessary as the questions are high-level.

To start, the user creates a physical prototype of a level. This can be created any collection of objects though our running examples will use LEGO bricks due to their versatility. The LEGO elements are used as spatial placeholders and do not need to have any resemblance to their true game item.

The game being prototyped in this example is a simple item collection game. The player must traverse the level to look after the baby by picking it up along with the bottle, carrier and food. Along the journey the player must also clean the environment by using the oil cleaner to clean the oil spill.

Data Capture The pipeline starts by receiving an image of the physical scene. The image must be taken as perpendicular as possible to the prototype. The prototype should be created on a solid white background to simplify the segmentation task. We capture the image with the rear camera of an iPhone 17 Pro Max.

Segmentation This stage of the pipeline (Figure 2a) extracts objects from the physical prototype. This is done through segmentation by using Meta's SAM 2 foundation model (Ravi et al. 2024). The image undergoes two iterations of segmentation. On the first pass, SAM 2 segments the raw image and produces a set of masks. However, some masks will fixate on unnecessary details and some elements are missed entirely. The system presents the user with each identified segment and the option to remove segments they deem irrelevant. The user then draws bounding boxes around each missed element. SAM-2 uses these boxes as prompts and generates masks for these new elements. To finish, the user annotates each segment with a descriptor label. In this example 36 masks were filtered to six. The user then prompted the creation of five masks.

Figure 3 shows this full process: **a** is the segmentation

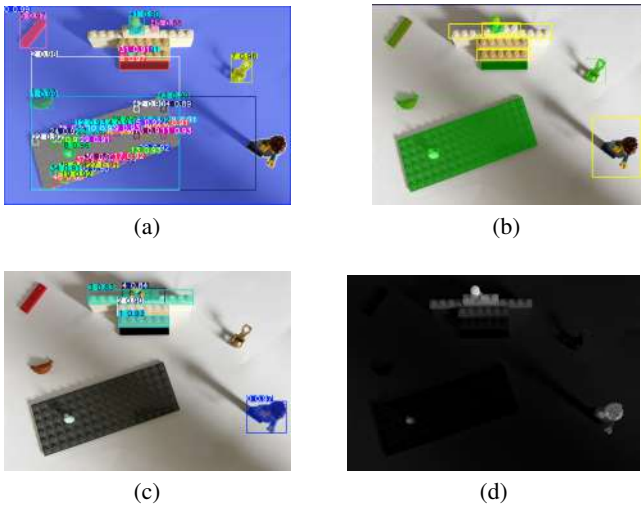


Figure 3: Visualisation of the segmentation process and depth map estimation.

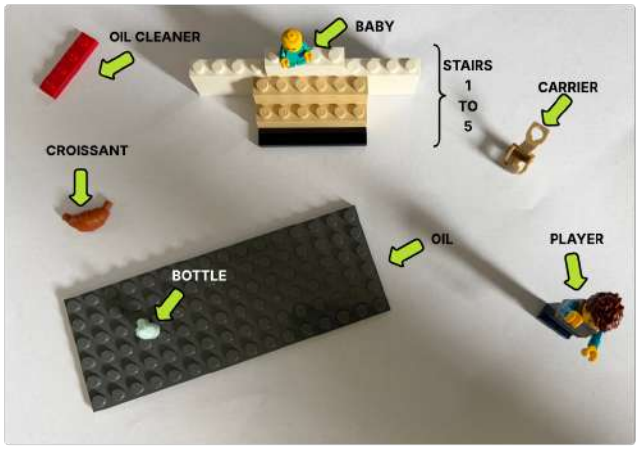


Figure 4: User labels.

ID	Questions
Q1	`Initially you have the types {initialTypes}. What other types of Objects will you have?`
Q2	`The system has automatically created states for applying dimensions,colours and labels but also for assigning objects to locations. What other types of States will you have?`
Q3	`What types of Actions will you have?`
Q4	`Which objects will be involved in the{state} state? Your choices are {objects}`
Q5	`Which objects {objects} will be involved in the{act} action?`
Q6	`Your keys are{parameters} for action{act}. Your choice of states are{states}. What states should be involved in the preconditions? Separate answers with commas. Type n to stop.`
Q7	`Which arguments will you have for the{precondition} precondition? Your choices are:{parameters}`
Q8	`Should the precondition be true? (y/n)`
Q9	`Your keys are{parameters} for action{act}. Your choice of states are{states}. What states should be involved in the effects? Separate answers with commas. Type n to stop.`
Q10	`Should the effect be true? (y/n)`
Q11	`Which arguments will you have for the{effect} effect? Your choices are:{parameters}`
Q12	`Do you want to add any more initialisation? Enter the state first, followed by parameters, finishing with boolean True of False.`
Q13	`Do you want to add any more goals? Please enter a state then each parameter.`

Table 1: System questions.

on first pass, **b** shows the saved segments in green and the five boxes drawn by the user in yellow. Lastly, **c** shows the generated masks from the hints. Figure 4 shows the labels assigned by the user.

Property Extraction This stage of the pipeline (Figure 2b) extracts meaningful data from the image to help improve the final visualised scene. As we are using RGB images we start by using a tool for depth map estimation (Ranftl et al. 2022). For our image the estimated depth map is seen in Figure 3d. For each segment we extract: an average RGB colour, a Cartesian location and a 3D scale (width, height, depth) in pixels.

Representation The next stage of the pipeline (Figure 2c) asks the user a range of questions regarding the domain to convert the induced labels and object properties into an automated planning representation. A flowchart modelling the user interaction is shown in Figure 5. The gathered data is converted into its PDDL equivalent and then into Python for use with the Unified Planning Library. We consider each element of a planning problem $\Pi = \langle P, A, I, G \rangle$, below.

Properties (P): Some PDDL properties are automatically defined as they are required for any game environment. These are `location`, `colour`, `dimension` and `label`, where each `Location` is a 3D Cartesian coordinate of the form `loc_x_y_z`, each `Colour` is an RGB colour of the form `col_r_g_b`, each `Dimension` is the calculated 3D scale of the form `dim_width_height_depth` and `Label` is the user specified annotation as seen in Figure 4.

The system then requests a list of task-specific objects (properties) from the user (Table 1 Q1) for this example the provided objects were: `player`, `static`, `cleaner` and `moveable`. Where `player` is the main character, `static` is any immovable objects, `cleaner` are the items used to clean oil spills and `moveable` is any regular item the player can interact with. The system then automati-

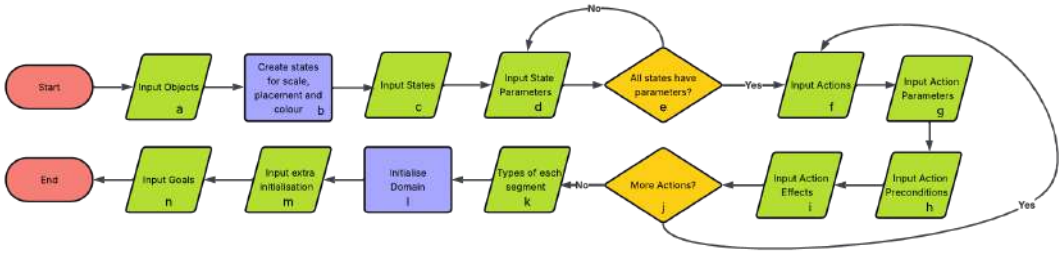


Figure 5: Flowchart for representation builder.

cally creates predicates that assign `Location, Colour, Dimension` and `Label` to each of the new properties.

Next, the system requests a list of domain-specific predicates from the user (1 Q2), including: `oiled` which marks the oil spill region, `holding` to model players holding moveable objects, `hascleaner` to equip players with the cleaning equipment and `cleaned` to mark cleaned oil spill regions. To finish, for each specified predicate the user is asked for applicable parameters.

The full list of predicates, both automated and user defined, are seen in Table 2. Table entries referring to a generic 'object' exist as four distinct predicates in our domain. For example, `objectAt` in our table is `staticAt, moveableAt, playerAt, cleanerAt` in our domain.

Actions (A): The system now asks the user for domain-specific actions (1 Q3) and their respective parameters, preconditions and effects. These questions are seen in Table 1 Q6-11 and are asked iteratively for each action. The actions for this example can be seen in Table 3. The `move` action allows a `player` to move between `locations`. `pickup and place` allow the `player` to equip and de-equip items. `pickup_cleaner` allows a `player` to equip a `cleaner` object and lastly `clean` allows a `player` to clean a spill at a `location`.

Initial State (I): The system asks the user to assign a property to each extracted segment. This response coupled with the data from the property extraction stage automatically builds the initial state which correctly assigns a location, colour, dimension and label to each extracted segment. The user is then asked to enter any other task specific initialisations with 1 Q12. In this case, we assign the `oiled` predicate to the `location` of the oil spill.

Goal State (G): Lastly, the system asks the user a series of questions to build the goal states (4 Q13). In this example, the goal states mimic the level completion and are seen in Table 4.

Before the Visualisation stage a planner is executed in the Python script to find a plan that reaches the user's specified goal states.

Visualisation The domain is visualised in Unity using PDSim (Figure 2d). To do this, the PDSim extension is first installed in Unity and the PDSim back-end server launched in the Python code. Once executed, the server will automatically be identified by PDSim and the domain and objects will be created. Upon creation, all objects are cube bodies placed at the same coordinate. To visualise movements the animations for predicates must be manually defined using Unity Script Graphs, where each node is an operation within Unity that can be customised using the Unity or PDSim libraries. These graphs are then executed whenever a predicate is referenced in the plan.

PDSim automatically creates prefabs (an object template in Unity) for each type of object in our scene. To prepare the environment for scene reconstruction these prefabs should be improved to allow for the correct placement of objects. First, the edges of the object prefab is marked as seen in Figure 6a. These labels help to align objects during placement to avoid overlapping. To improve scene explanation we add a text object to the prefab which will show the user defined label (Figure 6b) during reconstruction.

The Unity Script Graphs for each predicate animation can now be created as described below. The objectAt, objectDimension, objectColour and objectLabel predicates exist in four different forms relating to either the static, moveable, player or cleaner properties. For example the objectAt predicate described below relates to `staticAt, moveableAt, playerAt and cleanerAt`.

- **objectAt:** These animations deal with moving an object to it's desired `location`. To do this we make use of PDSim's *Translate to Point* node which takes three inputs; the moving object, the moving offset (the labelled bottom of the object) and the Cartesian coordinates of the desired `location`. As our `locations` are structured as `loc_x_y_z` the script graph extracts the Cartesian coordinate from the string and creates a Unity transform object (a position). The transform is then fed to the *Translate to Point* node to allow for the object's movement.
- **objectDimension:** These animations should scale an ob-

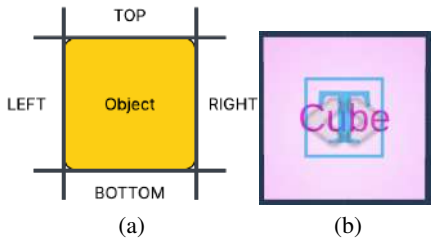


Figure 6: Environment preparation.

Predicate	Description	Automatically defined
`(objectAt ?o ?l)`	Maps object `?o` to location `?l`.	✓
`(objectSize ?s ?si)`	Assigns scale `?si` to object `?o`.	✓
`(objectColour ?o ?co)`	Assigns colour `?co` to object `?o`.	✓
`(objectLabel ?o ?l)`	Assigns label `?l` to object `?o`.	
`(oiled ?l)`	Marks location `?l` as an area with an oil spill.	✓
`(cleaned ?l)`	Marks location `?l` as an area which is now cleaned after previously being `oiled`.	✓
`(holding ?p ?m)`	Marks moveable object `?m` as being held by player **?p**.	✗
`(hascleaner ?p ?c)`	Marks cleaner object `?c` as being held by player **?p**.	✗

Table 2: Example environment predicates.

Action	Description
`(move ?p ?l1 ?l2)`	Move player `?p` from location `?l1` to location `?l2`
`(place ?p ?m ?l)`	Player `?p` places a moveable object `?m` at location `?l`
`(pickup ?p ?m ?l)`	Player `?p` picks up a moveable object `?m` from location`?l`
`(pickupCleaner ?p ?c ?l)`	Player `?p` picks up a cleaner object `?m` from location`?l`
`(clean ?p ?c ?l)`	Player `?p` cleans location `?l` using `?c`

Table 3: Example environment actions.

Goals
`(holding player bottle)`
`(holding player baby)`
`(holding player croissant)`
`(holding player carrier)`
`(cleaned loc_oil_spill)`

Table 4: Example environment goals.

ject to the `dimension` extracted in the early stages. This graph parses the `dimension dim_x_y_z` and again creates a Unity Transform which this time represents scale. We apply the scale to the object using various inbuilt Unity nodes.

- **objectColour:** These animations colour the object to match the determined `colour`. Similar to the above, the script graph parses the **rgb_r_g_b** colour, creates a new Unity colour property and applies the new colour to the object's mesh.
- **objectLabel:** These animations attach the user specified label to each object. The script graph extracts the label and edits the initial 'Cube' (Figure 6b) to match the annotation.
- **holding, hasCleaner:** These predicates are user specified and deal with the player carrying different objects. The script graph animates this by setting the object meshes of items to be invisible when picked up by the player.
- **cleaned** This predicate is triggered when an oil spill is cleaned. This animation simply turns the object to be white to mimic cleanliness.

To improve the visuals of our prototype we manually added 3D models taken from an online dataset to create the scene shown in Figure 1b. At this stage our scenes are reconstructed to visualise the initial state. Figure 7a shows the scene midway through plan execution, at this point the player has picked up the bottle, baby and carrier. Figure 7b shows the scene after plan execution with the remaining objects picked up and the oil spill cleaned.

We can apply this same pipeline to a range of games, for example the stacking game in Figure 8 where the prototype, reconstruction and executed plan are shown.

Experiments and Discussion

The pipeline was implemented using Python 3.12 and Unity version 2022.3.62f1. The hardware used for the pipeline was a desktop PC with a 6 core CPU, 1,920 CUDA core GPU and 48GB of memory, running Windows 11.

Playing Reconstructed Scene So far, the paper has demonstrated the planning aspect and level testing benefit to reconstructing our scenes using this approach. The other benefit we highlighted was to reconstruct a high fidelity prototype for user testing that is playable. In this experiment we take the scene and reconstruction found in Figure 8a and b respectively and produce a game where the user has to clean the scene.

To add functionality to this game we create a simple C# script to handle player movement and assign the 'c' key as our clean button. When pressed, the closest item within a radius is removed. Figure 9 shows the game being played in the reconstructed scene and items in the hierarchy have been greyed out after cleaning.

Platformer Experiment In this experiment we went for a Mario-Style platformer. The prototype, reconstruction and animations are seen in Figure 10 respectively. The prototype for this scene was mainly drawn and the pipeline still worked as usual.

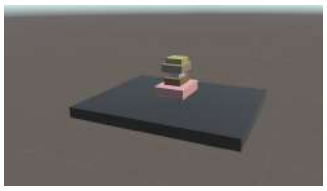
The final scene was successfully created in Unity and required one small change to the pipeline. Initially, properties

(a)

(b)

Figure 7: Example execution.

(a) (b)



(c)

Figure 8: Stacking Game.

Figure 9: Cleaning Game Execution.

Description
This image is a prototype of a level for a game. The game involves a player that collects items (baby, bottle, carrier, food) and cleans an oil spill using oil cleaner.

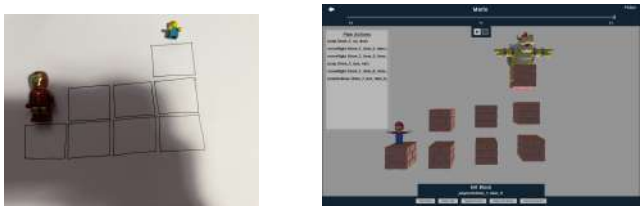
Table 5: Example environment goals.

were defined by asking users what type of object each segmented element was. In this game we require a `level` a property to track heights of different blocks for the jumping mechanic. This property does not refer to a physically drawn element so it was missed in the questionnaire. We added an extra question asking if there were any other properties the user wanted to define.

Full Automation Experiment The representation stage in the main example took a total of 8 minutes and 40 seconds. This is significantly faster than manually typing out the entire domain but could be automated further. This experiment uses Anthropic's Claude VLM, in particular the `claude-sonnet-4-6` model with default reasoning settings. The model was asked to perform the role of the user in aspects of the reconstruction as seen in Figure11.

We start by providing the VLM with instructions on its role in this system. We also give the VLM the image in Figure 1a, a description of how the game should function (Table 5) and a copy of the code to be aware of the upcoming questions. Lastly, we instruct the VLM to keep responses conscience and avoid the use of punctuation unless specified in the question.

This experiment was run multiple times and failed each time. When an issue arose and was 'fixed' another issue would appear. When working with users, the questions do not have to be explicit. For example we ask the user which parameters should be used in the `move` action, the user knows to reply with `player,location,location` as we need both the initial and destination locations. The VLM did not and the question had to be altered to explicitly suggest that the same property can be listed more than once. We tried to constrain the VLM's responses by giving it multiple choice questions when choosing preconditions and parameters but even then the VLM sometimes responded outwith



(a) (b)

(c) (d)

Figure 10: Platformer experiment step through.

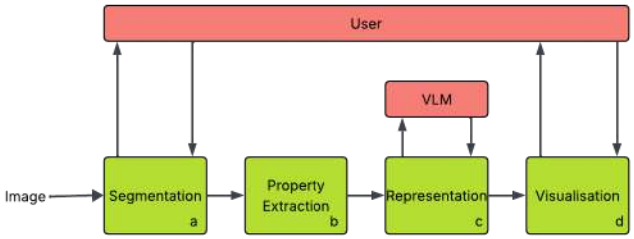


Figure 11: More automated pipeline.

the domain. Lastly, the VLM sometimes would randomly include punctuation even though it was instructed not to in the prompt.

Observations These experiments highlight the modularity of the presented approach. The human-in-the-loop approach allows for a generalised method to fit different styles of games. During experimentation several observations were made which are detailed below.

To refresh, the approach asks user's questions regarding the domain so it can be constructed. A run through of the pipeline from start to finish took around 12 minutes for the base example and 19 minutes for the platformer. Aspects of the approach can be streamlined or automated to quicken level reconstruction as described below:

Currently, questions are asked in textbox pop-ups. Sometimes this is a requirement but some questions such as indicating preconditions, effects and their respective parameters should be asked through graphical interfaces. Clicking buttons will be significantly quicker and easier to follow than the text approach. By constraining the inputs with buttons this approach could also improve the VLM integration as well as overall user experience. Another aspect to automate will be the Unity Script Graph creation which can be repetitive especially in domains with a large number of objects.

The item labels help to distinguish between different objects although it is still relatively difficult. The manually included 3D models help significantly but this could also be automated to pull models from online free to use datasets using the user defined annotations for the search.

The automation experiment highlights the unpredictability behind VLMs and their difficulty in reasoning over long horizons. When presented with the entire code base and description of the planning task it struggled to determine which actions were needed. In the examples where the VLM conformed to the rules of the domain and syntactically made it through the questionnaire the final plan failed. Often even if logical issues were corrected there were many aspects of the game that were forgotten. The model we used was an off the shelf VLM. Perhaps with tuning and a more task specific VLM this could be automated.

To further encourage the use of our approach for a quick prototype, improvements can be made in the visualised scene. The arrangement and colours were relatively correct but still many improvements can be made. In our approach we scale objects to a 3D Cartesian scale, this scale works great for rectangular shaped elements but not for circular shapes. Rotations are also ignored in our simulation leading to a different placement.

Conclusions and Future Work

This paper investigates the use of Dynamic Scene Reconstruction for planning-based game environments and builds the foundations for using this method for early stage prototyping and level testing. Our implementation in its current form builds a suitable prototype but it can be improved further in future work by including more accurate dimensional representations of elements and including the attainment of 3D models in the primary pipeline. Future work will also involve improving the user interface to be more user friendly, automating setup in Unity and testing other generative tools to automate aspects of the pipeline. Other than improvements to the pipeline we also look to undertake a user study to compare this method of constructing a prototype to traditional design methods.

References

Balyo, T.; Barták, R.; Chrpa, L.; Červenka, M.; Dvořák, F.; Gocht, S.; Lipčák, L.; Macek, V.; Roháček, D.; Ryzí, J.; Suda, M.; Šafránek, D.; Švancar, S.; and Youngblood, G. M. 2025a. Using Planning for Automated Testing of Video Games. In *Proceedings of the International Joint Conference on Artificial Intelligence (IJCAI) Demo Track*, 11004–11008.

Balyo, T.; Barták, R.; Chrpa, L.; Červenka, M.; Dvořák, F.; Gocht, S.; Lipčák, L.; Macek, V.; Roháček, D.; Ryzí, J.; Suda, M.; Šafránek, D.; Švancár, S.; and Youngblood, G. M. 2025b. Using planning for automated testing of video games. In *Proceedings of the Thirty-Fourth International Joint Conference on Artificial Intelligence*, IJCAI '25. ISBN 978-1-956792-06-5.

Bavelos, A. C.; Anastasiou, E.; Dimitropoulos, N.; Michalos, G.; and Makris, S. 2025. Virtual reality-based dynamic scene recreation and robot teleoperation for hazardous environments. *Computer-Aided Civ. and Infrastructure Eng.*, 40(3): 392–408.

Berrios, W.; Mittal, G.; Thrush, T.; Kiela, D.; and Singh, A. 2023. Towards Language Models That Can See: Computer Vision Through the LENS of Natural Language. arXiv:2306.16410.

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. 2020. Planimation. arXiv:2008.04600.

Collins, J.; Chand, S.; Vanderkop, A.; and Howard, D. 2021. A Review of Physics Simulators for Robotic Applications. *IEEE Access*, 9: 51416–51431.

Dalal, N.; and Triggs, B. 2005. Histograms of oriented gradients for human detection. In *Proc. CVPR*, 886–893.

De Pellegrin, E.; and Petrick, R. P. 2024. Planning domain simulation: an interactive system for plan visualisation. In *Proc. ICAPS*, 133–141.

Downs, L.; Francis, A.; Koenig, N.; Kinman, B.; Hickman, R.; Reymann, K.; McHugh, T. B.; and Vanhoucke, V. 2022. Google Scanned Objects: A High-Quality Dataset of 3D Scanned Household Items. In *Proc. ICRA*, 2553–2560.

Duarte, F. F.; Lau, N.; Pereira, A.; and Reis, L. P. 2020. A Survey of Planning and Learning in Games. *Applied Sciences*, 10(13).

Entertainment, R. 2011. Orcs Must Die! Video game.

Ferrara, E. 2023. Should ChatGPT be biased? Challenges and risks of bias in large language models. *First Monday*.

Games, G. 2004. Killzone. Video game.

Gerevini, A. E.; and Saetti, A. 2020. An interactive tool for plan generation, inspection, and visualization. In *Knowledge Engineering Tools and Techniques for AI Planning*, 127–155.

Ghallab, M.; Nau, D. S.; and Traverso, P. 2016. *Automated planning and acting*. Cambridge University Press.

Girshick, R. 2015. Fast R-CNN. In *Proc. ICCV*.

Gu, Q.; Kuwajerwala, A.; Morin, S.; Jatavallabhula, K. M.; Sen, B.; Agarwal, A.; Rivera, C.; Paul, W.; Ellis, K.; Chellappa, R.; Gan, C.; de Melo, C. M.; Tenenbaum, J. B.; Torralba, A.; Shkurti, F.; and Paull, L. 2023. ConceptGraphs: Open-Vocabulary 3D Scene Graphs for Perception and Planning. arXiv:2309.16650.

James, L. 2021. Generating video game puzzles through planning. UK PlanSIG 2021 (ONLINE) : The second online Workshop of the UK Planning & Scheduling Special Interest Group ; Conference date: 20-12-2021 Through 20-12-2021.

Karami, E.; Prasad, S.; and Shehata, M. 2017. Image Matching Using SIFT, SURF, BRIEF and ORB: Performance Comparison for Distorted Images. arXiv:1710.02726.

Kargar, S. M.; Yordanov, B.; Harvey, C.; and Asadipour, A. 2024. Emerging Trends in Realistic Robotic Simulations: A Comprehensive Systematic Literature Review. *IEEE Access*, 12: 191264–191287.

Kelly, J.-P.; Botea, A.; Koenig, S.; et al. 2007. Planning with hierarchical task networks in video games. In *Proceedings of the ICAPS-07 Workshop on Planning in Games*, 18.

Kerbl, B.; Kopanas, G.; Leimkühler, T.; and Drettakis, G. 2023. 3D Gaussian splatting for real-time radiance field rendering. *ACM Transactions on Graphics*, 42(4): 1–14.

McDermott, D.; Ghallab, M.; Howe, A.; Knoblock, C.; Ram, A.; Veloso, M.; Weld, D.; and Wilkins, D. 1998. PDDL—The planning domain definition language. Tech. Report, Yale Center for Computational Vision and Control.

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.

Naik, D.; Naik, I.; and Naik, N. 2024. Large Data Begets Large Data: Studying Large Language Models (LLMs) and Its History, Types, Working, Benefits and Limitations. In *Contributions Presented at C3AI*, 293–314.

Neufeld, X.; Mostaghim, S.; Sancho-Pradel, D. L.; and Brand, S. 2017. Building a planner: A survey of planning systems used in commercial video games. *IEEE Transactions on Games*, 11(2): 91–108.

Ni, Z.; Hupkes, J.; Eriksson, P.; Leijonhufvud, G.; Karlsson, M.; and Gong, S. 2025. Parametric Digital Twins for Preserving Historic Buildings: A Case Study at Löfstad Castle in Östergötland, Sweden. *IEEE Access*, 13: 3371–3389.

NVIDIA Corporation. 2025. *PhysX SDK Documentation*.

Othman, A. A.; De Pellegrin, E.; and Petrick, R. P. 2026. Dynamic Scene Reconstruction for Planning Environments. In *Proc. UKPlanSIG 2026*.

Pellegrin, E. D.; and Petrick, R. 2024. Simulating Robotics Planning Domains with PDSim and ROS. In *ICAPS 2024 System's Demonstration track*.

Politowski, C.; Petrillo, F.; and Guéhéneuc, Y.-G. 2021. A Survey of Video Game Testing. In *2021 IEEE/ACM International Conference on Automation of Software Test (AST)*, 90–99.

Productions, M. 2005. F.E.A.R. Video game.

Ramadan, R.; and Hendradjaya, B. 2014. Development of game testing method for measuring game quality. In *2014 International Conference on Data and Software Engineering (ICODSE)*, 1–6.

Ramirez, A.; and Bulitko, V. 2015. Automated Planning and Player Modeling for Interactive Storytelling. *IEEE Transactions on Computational Intelligence and AI in Games*, 7(4): 375–386.

Ranftl, R.; Lasinger, K.; Hafner, D.; Schindler, K.; and Koltun, V. 2022. Towards Robust Monocular Depth Estimation: Mixing Datasets for Zero-Shot Cross-Dataset Transfer. *IEEE Transactions on Pattern Analysis and Machine Intelligence*, 44(3).

Raschka, S. 2024. *Build a Large Language Model (From Scratch)*. Shelter Island, NY: Manning Publications.

Ravi, N.; Gabeur, V.; Hu, Y.-T.; Hu, R.; Ryali, C.; Ma, T.; Khedr, H.; Rädle, R.; Rolland, C.; Gustafson, L.; Mintun, E.; Pan, J.; Alwala, K. V.; Carion, N.; Wu, C.-Y.; Girshick, R.; Dollár, P.; and Feichtenhofer, C. 2024. SAM 2: Segment Anything in Images and Videos. *arXiv preprint arXiv:2408.00714*.

Redmon, J.; Divvala, S.; Girshick, R.; and Farhadi, A. 2016. You Only Look Once: Unified, Real-Time Object Detection. In *Proc. CVPR*.

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 *Proc. ICAPS*, 486–490.

Rosinol, A.; Gupta, A.; Abate, M.; Shi, J.; and Carlone, L. 2020. 3D Dynamic Scene Graphs: Actionable Spatial Perception with Places, Objects, and Humans. arXiv:2002.06289.

Stahlke, S.; Nova, A.; and Mirza-Babaei, P. 2020. Artificial Players in the Design Process: Developing an Automated Testing Tool for Game Level and World Design. CHI PLAY '20, 267–280. New York, NY, USA: Association for Computing Machinery. ISBN 9781450380744.

Studios, H. M. 2012. Transformers: Fall of Cybertron. Video game.

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*, 81–88.

Techland. 2015. Dying Light. Video game.

Unity Technologies. 2025. Unity. Game development platform.

Verdouw, C.; Tekinerdogan, B.; Beulens, A.; and Wolfert, S. 2021. Digital twins in smart farming. *Agricultural Systems*, 189: 103046.

Verykokou, S.; and Ioannidis, C. 2023. An Overview on Image-Based and Scanner-Based 3D Modeling Technologies. *Sensors*, 23(2).

Vidal, J.; Vallicrosa, G.; Martí, R.; and Barnada, M. 2023. Brickognize: Applying Photo-Realistic Image Synthesis for Lego Bricks Recognition with Limited Data. *Sensors*, 23(4): 1898.

Wang, Z.; Han, K.; and Tiwari, P. 2021. Digital Twin Simulation of Connected and Automated Vehicles with the Unity Game Engine. In *Proc. DTPI*, 1–4.

Zhiheng, X.; Rui, Z.; and Tao, G. 2023. Safety and Ethical Concerns of Large Language Models. In Sun, M.; Qin, B.; Qiu, X.; Jiang, J.; and Han, X., eds., *Proc. Chinese Nat. Conf. on Computational Linguistics (Tutorial Abstracts)*, 9–16.