

Constraint-Based Four-Dimensional Spacetime Blending of Learned Game Mechanics Along Orthogonal Planes

Akshar Vandara

Khoury College of Computer Sciences
Northeastern University
Boston, Massachusetts, USA
vandara.a@northeastern.edu

Seth Cooper

Khoury College of Computer Sciences
Northeastern University
Boston, Massachusetts, USA
se.cooper@northeastern.edu

Abstract

Blending mechanics from different games may provide new game-play opportunities. In this work, we blend tile-based game mechanics to generate levels for blended games. Mechanics are learned from example playthroughs of each game’s levels as changes in tile configurations. In particular, we show that mechanics learned from 2D games can be blended along different orthogonal planes to create a new 3D game; that is, the mechanics from one game apply along one plane and the mechanics from another game apply along a different plane. We ensure completability of the generated levels by also using temporal constraints to generate a solution along with the level, resulting in a level with its solution being generated as conceptual 4D spacetime block, which we encode in a 2D “expanded filmstrip” representation. We demonstrate the application to several tile-based puzzle games.

Keywords

procedural content generation, constraint solving, game blending

ACM Reference Format:

Akshar Vandara and Seth Cooper. 2026. Constraint-Based Four-Dimensional Spacetime Blending of Learned Game Mechanics Along Orthogonal Planes. In *Foundations of Digital Games (FDG '26)*, August 10–13, 2026, Copenhagen, Denmark. ACM, New York, NY, USA, 9 pages. <https://doi.org/10.1145/3815598.3815702>

1 Introduction

In the field of procedural content generation [22], some work has looked at approaches to automatically blending games as means to create new games or explore the space of game possibilities [20].

However, existing approaches typically blend games within the *same* dimensions. That is, when multiple *spatially* 2D games are blended, they are blended such that the 2D spaces of the games are aligned, and create a new 2D game. This may limit the blending possibilities available and thus the space of new gameplay possibilities.

Therefore in this work, we explore the possibility of blending games in such a way that their spatial dimensions are not aligned. Using multiple spatially 2D games, we blend their mechanics in such a way that the 2D planes for each game are orthogonal, creating a blended game that is spatially 3D. In this work we use *row*, *column*,

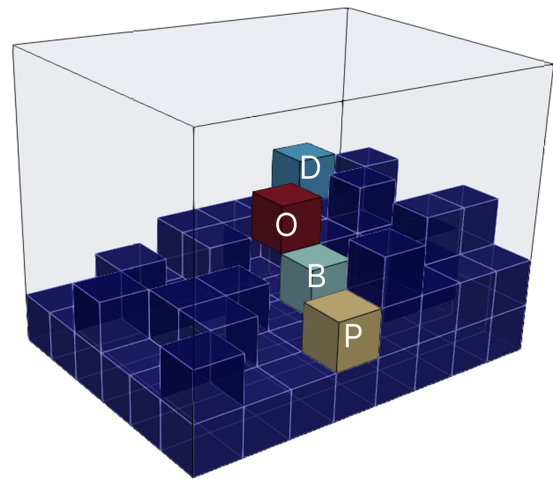


Figure 1: First frame of blending *Sokoban* $\times$ *Block Dude*. The output is generated by merging patterns extracted from spatially 2D training examples (3D spacetime blocks) by reorganizing them in orthogonal planes to each other. The visualization is displayed in Paraview. Labels show P: Player; B: Crate; O: Crate-Target; and D: Exit.

and *story* to refer to the three spatial dimensions. As an example, if one game’s mechanics apply in the row-column plane, and the other’s apply in the column-story plane, these could be blended to create a game that takes place in the row-column-story cube. This unlocks several new avenues to test, explore, and generate novel solvable levels for game blending ideas where a designer can unlock new perspectives on applying a game mechanic with added different layers of complexities to it.

We build on the approach of Vandara et al. [31], which learns mechanics for 2D turn-based and tile-based games from example playthroughs of levels. In this case, a *mechanic* is a way that a spatial arrangement of tiles can change in a single timestep. The approach explicitly includes the *time* dimension in a 3D spacetime block, using a constraint-based spacetime approach to ensure completability of the generated levels by the learned mechanics. Summarizing the approach, given a training playthrough to learn from, constraints are added on the generated level such that local patterns of tiles in the first and last timestep of a generated playthrough correspond to such patterns in the first and last timestep of the training



This work is licensed under a Creative Commons Attribution 4.0 International License. *FDG '26, Copenhagen, Denmark*

© 2026 Copyright held by the owner/author(s).

ACM ISBN 979-8-4007-2495-4/26/08

<https://doi.org/10.1145/3815598.3815702>

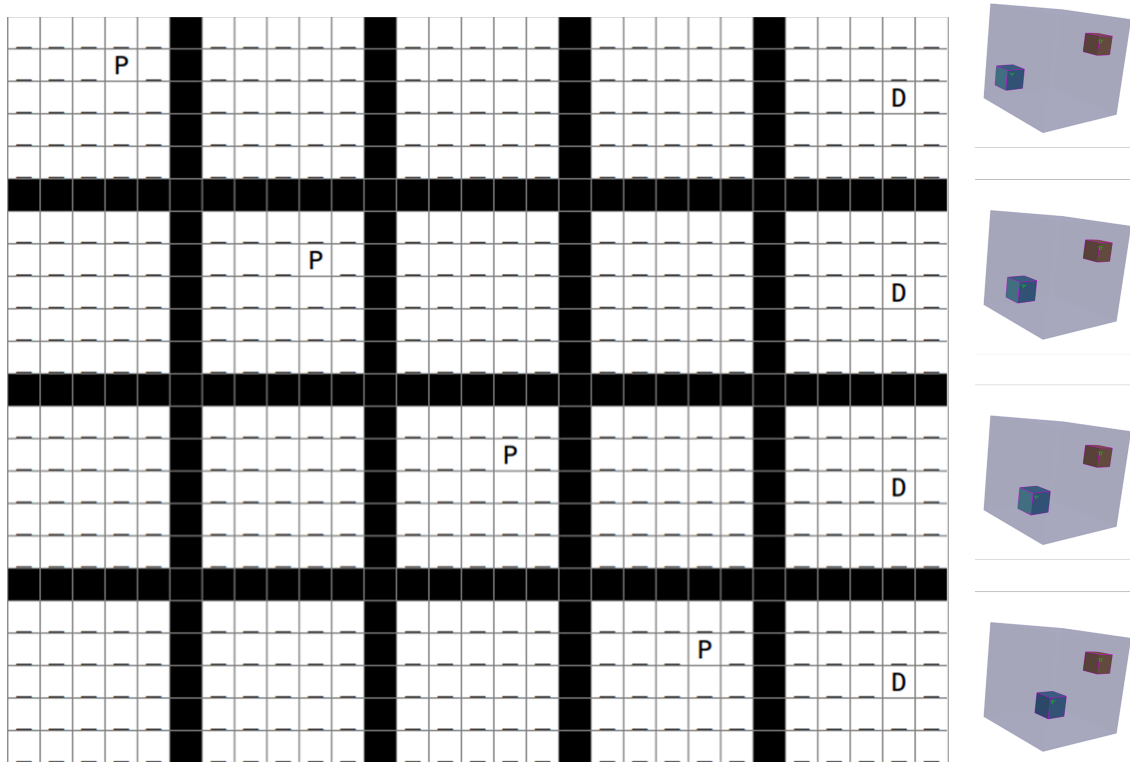


Figure 2: Illustration of the 2D *expanded filmstrip* representation used in this work to represent a 4D spacetime block, with P: Player and D: Exit. On the left, the text-based expanded filmstrip. Each rectangle of characters is a single row-column slice through a frame, with all such slices (the third spatial dimension, stories) for a frame unrolled along the column-axis. Each frame is thus a 3D row-column-story block. Frames themselves are unrolled vertically along the row-axis. On the right, the corresponding voxel representation of the frames (using Paraview [1]).

playthrough, and between timesteps, each tile updates in a way that locally corresponds to some update pattern seen in the training level. Thus the generated playthrough is an example solution to the level in its first timestep.

Whereas Vandara et al.’s work was learning to generate spatially 2D games in 3D spacetime blocks, here we generate spatially 3D games in 4D spacetime blocks, where 2D games can independently constrain mechanics within different planes, blending the games. Where that work used a *filmstrip* representation to encode the 3D spacetime block into a 2D grid by unrolling time along the row-axis, we use an *expanded filmstrip* representation to encode the 4D spacetime block into a 2D grid: unrolling time along the row-axis and stories (the 3rd spatial dimension) along the column-axis. This encoding allows us to work with a conceptually 4D spacetime block in an existing 2D system. An example is shown in Figure 2.

We found that this approach can be used to blend mechanics from puzzle games to produce new puzzles incorporating mechanics from multiple games along different spatial dimensions.

This work contributes an approach to blending games along different spatial dimensions to produce levels for new games of higher spatial dimension, in a constraint-based framework that ensures that the levels are solvable.

2 Related Work

2.1 Procedural Content Generation

Procedural content generation [22] has long been used to generate content for existing games. Although there are a variety of approaches, most closely related to this work are constraint-based approaches that generate levels. These approaches formulate desired properties of levels and then use constraint solvers to find satisfying levels.

Much work in this area has made use of Answer Set Programming [26], for example to generate mazes and dungeons [17, 23], graph-based levels [27], and educational games [24]. Some work has extended Answer Set Programming to rule out undesired solutions to levels [25]. Some Answer Set Programming approaches can also be seen to incorporate discrete timesteps explicitly, to model e.g. reachability through a maze, creating a solvable level, and also to constrain the solution length [26].

Constraint-based approaches have also been developed that can incorporate information learned from training examples. The WaveFunctionCollapse algorithm [10], which learns local patterns of tiles and generates images or levels containing those patterns, has become popular. It has been studied [12] and extended in various

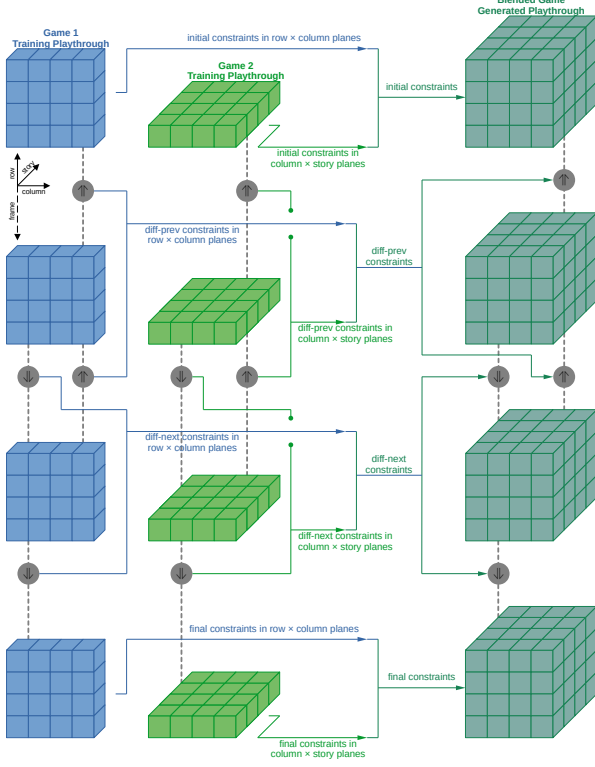


Figure 3: Conceptual overview of the learning and generation process. Double arrows ($\Downarrow$, $\Uparrow$) represent a diff with the next or previous frame.

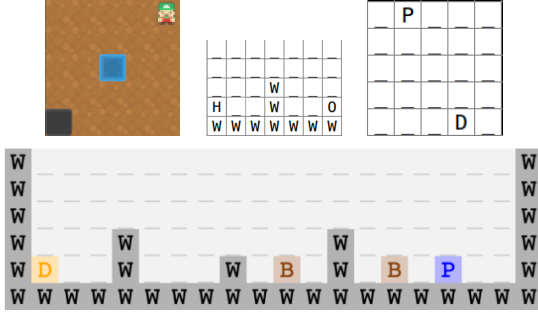


Figure 4: First frame from filmstrip of training examples. Top row (left to right): Sokoban, Snake, Field. Bottom: Block Dude.

ways, such as incorporating negative examples [13] and working hierarchically [2] or interactively [14].

2.2 Spacetime Constraints

In this work we explicitly represent time as a dimension of the constraint problem, in order to generate a solution along with a level. Early work in such spacetime constraints was applied to

Game	Time, sec.	Eff. Length
Field (2D) $5 \times 5 \times 15$	0.13 (0.00)	9.3 (3.8)
Sokoban (2D) $5 \times 5 \times 15$	0.48 (0.02)	7.7 (1.8)
Block Dude (2D) $6 \times 8 \times 15$	0.93 (0.05)	8.3 (2.4)
Snake (2D) $5 \times 7 \times 15$	0.15 (0.01)	7.8 (2.1)

Table 1: Summary of individual game generation results (10 runs each). Mean (standard deviation) given.

Game	Time, sec.	Eff. Length
Field $\times$ Field $5 \times 5 \times 5 \times 15$	0.44 (0.01)	6.9 (1.5)
Sokoban $\times$ Sokoban $5 \times 5 \times 5 \times 15$	3.14 (0.75)	14.2 (1.8)
Sokoban $\times$ Field (2D, same plane) $5 \times 5 \times 15$	1.04 (0.44)	14.8 (0.6)
Sokoban $\times$ Block Dude (2D, same plane) $6 \times 6 \times 10$	0.94 (0.30)	9.0 (1.5)
Sokoban $\times$ Block Dude $6 \times 8 \times 6 \times 20$	24.4 (15.52)	18.7 (3.2)
Block Dude $\times$ Snake $5 \times 7 \times 5 \times 15$	10.97 (0.61)	12.1 (3.3)

Table 2: Summary of blended game generation results (10 runs each). Mean (standard deviation) given.

animations, allowing authorial control while ensuring physical plausibility of the animations [16, 32].

Recently, work in the Sturgeon constraint-based level generation system [6] has incorporated spacetime representations to apply constraints regarding the solvability of levels, in both grid-based [31] and graph-based [7] applications.

2.3 Game Blending

Some recent work has also explored game blending, often using machine-learning techniques, trained on multiple games.

Some work has use neural-network techniques such as LSTMs [19] or different types of variational autoencoders [20, 21] to generate blended levels. Some have approached determining how a player would need to be able to move to get through such a blended level [28]. Others have used constraint-based techniques to combine level tile patterns and player movement models from multiple games [5] to generate new solvable, blended levels.

Recent work in large-scale world models, such as Genie [4, 18], may be considered to fall into this category, as they can learn to generate a wide variety of games as controllable videos. However, they are not directly designed to blend specific games, and may not necessarily generate games with goals that can be solved or be played for an extended duration.

3 Approach

This work uses the Sturgeon constraint-based level generation system [6] to learn spacetime patterns from input examples and

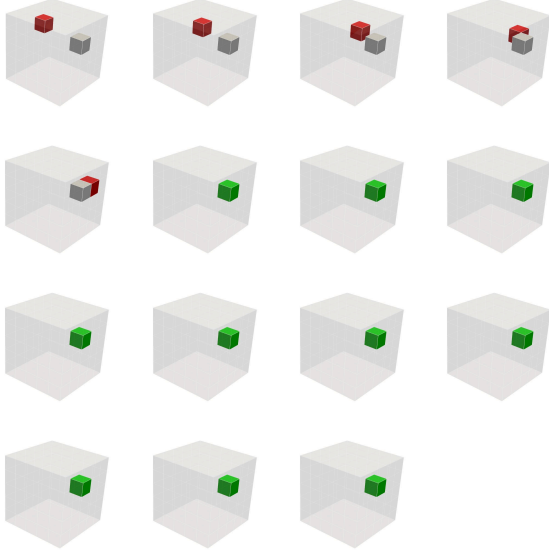


Figure 5: *Field × Field* ($5 \times 5 \times 5 \times 15$) — Trained across all three orthogonal planes (row $\times$ col, col $\times$ story, row $\times$ story).

■ Player, ■ Exit, ■ Terminal

apply them to generate spacetime playthroughs like in previous work [31]. The main difference between this and the previous work is that, where that work learned and applied patterns from one game at a time, in this work we learn and apply patterns from multiple games at once, along different planes, to blend spatially 2D games in to a 3D game.

Sturgeon is a system that uses constraint solvers to generate 2D tile-based levels. Sturgeon supports extracting *patterns* from training examples levels and applying them to higher-level design rules to generate levels by setting them up as a Boolean constraint satisfaction problem. It then solves the problem using a lower-level constraint solver to find the solution and generate the level. In this work, we are using PySAT’s [11] gluecard41 SAT solver, which builds on the Glucose [3], MiniCARD [15], and MiniSAT [9] solvers.

Sturgeon primarily works on 2D levels, so to encode a 4D playthrough, which incorporates three spatial dimension and one time dimension (row $\times$ col $\times$ story $\times$ frame) as 2D, we expanded our previous 3D (row $\times$ col $\times$ frame) spacetime block filmstrip representation laterally, which we call an *expanded filmstrip*.

We use the *diff method* from previous work [31] to dynamically extract patterns from these varied plane organizations into *groups* of patterns. The initial and final frames simply extract the tiles present in these frames into the corresponding *initial* and *final* pattern groups. These will constrain what tiles are allowed at the beginning and end of the playthrough, which will ensure that a level is solved by the final frame. The intermediate frames are used to learn the *diff-next* and *diff-prev* pattern groups. These patterns are extracted by looking, for each tile in each frame, at the corresponding tile in the next frame or the previous frame, and, if it

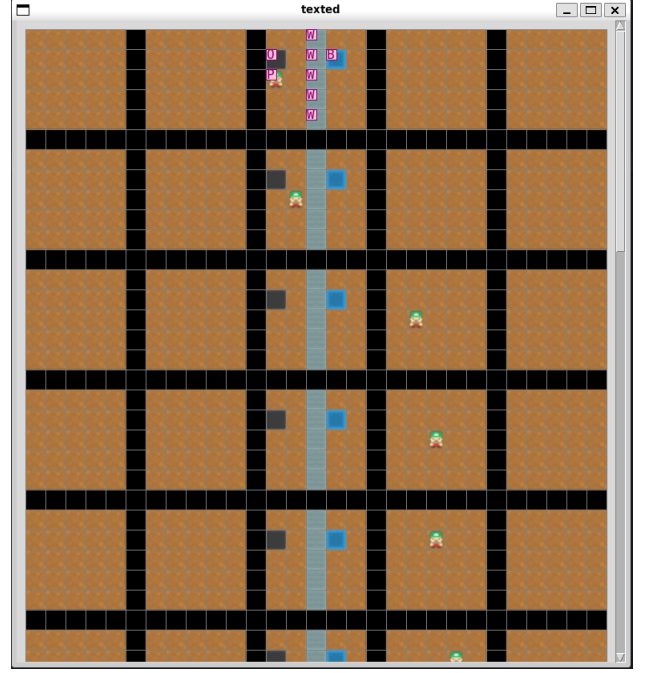


Figure 6: *Sokoban × Sokoban* ($5 \times 5 \times 5 \times 15$) — Using the interactive editor to add/remove constraints in *Sokoban × Sokoban*, to yield interesting or curated results (first few frames).

changed, what other tiles changed with it. The *diff-next* and *diff-prev* groups essentially capture how a tile can change into another tile forward and backward in time. These will constrain how the tiles can change between frames in a generated playthrough.

By default, game playthroughs are organized spatially in the row $\times$ col plane. In order to blend games along different axes, we first reorganize a playthrough along a different plane (such as col $\times$ story or row $\times$ story). After learning the pattern groups for each game, we then merge the groups together. That is, the *initial* group patterns from all the games are merged into one *initial* pattern group, and the same for the *final*, *diff-next* and *diff-prev* groups. These merged patterns are then used to constrain an expanded filmstrip of three spatial (rows $\times$ cols $\times$ stories) and one temporal (frames) dimension. The *initial* and *final* patterns are applied to the initial and final generated frames respectively, and the *diff-next* and *diff-prev* to the intermediate frames. We use the constraint that for *diff-next* and *diff-prev* patterns, exactly one change pattern from each group is used. A conceptual overview of the process is shown in Figure 3.

In the generated spacetime block, the first frame is the initial 3D game state of the new blended generated game, and the entire filmstrip provides an example solution to that level. The interactive editor tool [31] still supports this new representation and designers could add constraints to tiles at any points in spacetime, throughout the expanded filmstrip.

Because of the way diff patterns are learned, they depend on specific spacing of tiles in the training levels, and thus some dimensions of the training playthroughs cannot be changed in the

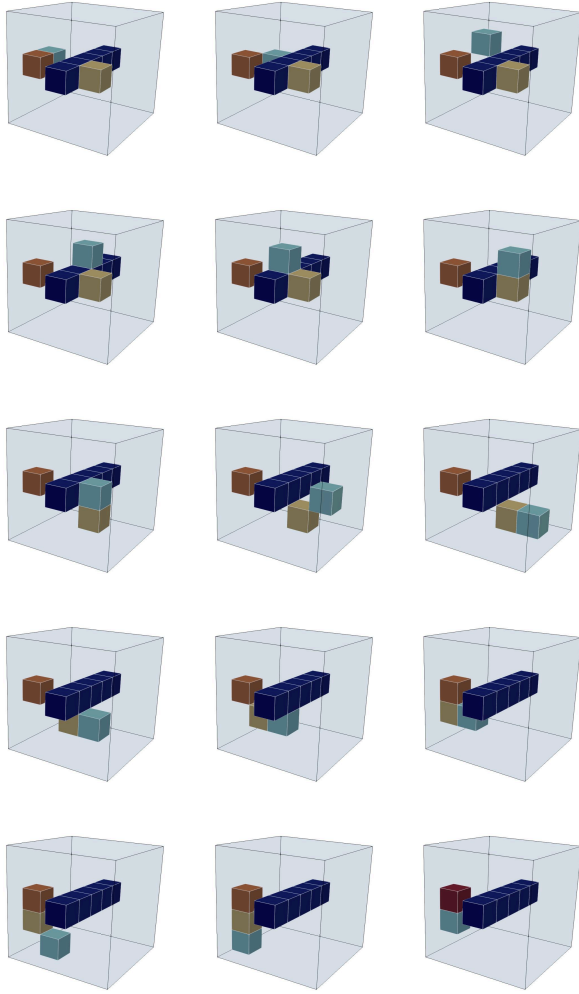


Figure 7: *Sokoban* $\times$ *Sokoban* ($5 \times 5 \times 5 \times 15$) — The Sokoban player, maneuvers around the wall-block by changing elevations, to push the crates into its target. (In this case, it was constrained to have exactly one player, one crate and one crate-target for the initial frame.)

■ Player, ■ Wall, ■ Crate
■ Crate-Target, ■ Terminal

generated playthrough directly. However, we expect that these dimensions could be changed using the spacing remapping approach from previous work [31].

Upon generation, we noticed that the expanded filmstrip representation could get complicated to follow along as more complex games were blended (like Sokoban $\times$ Block Dude). To resolve this, we added a way to convert generated levels into a *vtk* file. A *vtk* file is a standard file format used by The Visualization Toolkit (VTK) that acts as a consistent way to store structured geometric data (e.g. voxels) among its other capabilities. This data can be then imported into Paraview [1] to visualize it in an interpretable 3D voxel representation, for each of the respective frames. We add different

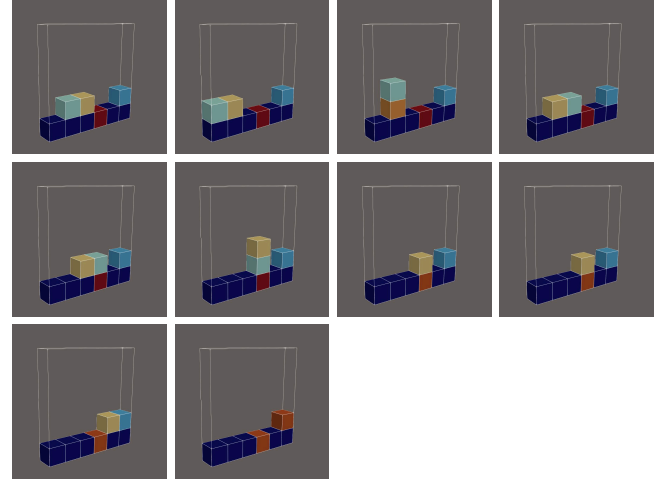


Figure 8: *Sokoban* $\times$ *Block Dude* ($6 \times 6 \times 10$) — Blended across same plane (row $\times$ col) for both of the games.

■ Player, ■ Crate, ■ Exit,
■ Crate-Target, ■ Walls,
■ Terminal, ■ Player-Pickup



Figure 9: *Sokoban* $\times$ *Field* ($5 \times 5 \times 15$) — Blended across same plane (row $\times$ col) for both of the games.

■ Player, ■ Crate,
■ Crate-Target, ■ Exit,
■ Terminal

threshold filters for the tile types to control their opacity, in order to fade out trivial tiles (void tiles, wall tiles) that could block the vision for what happens within the playthrough.

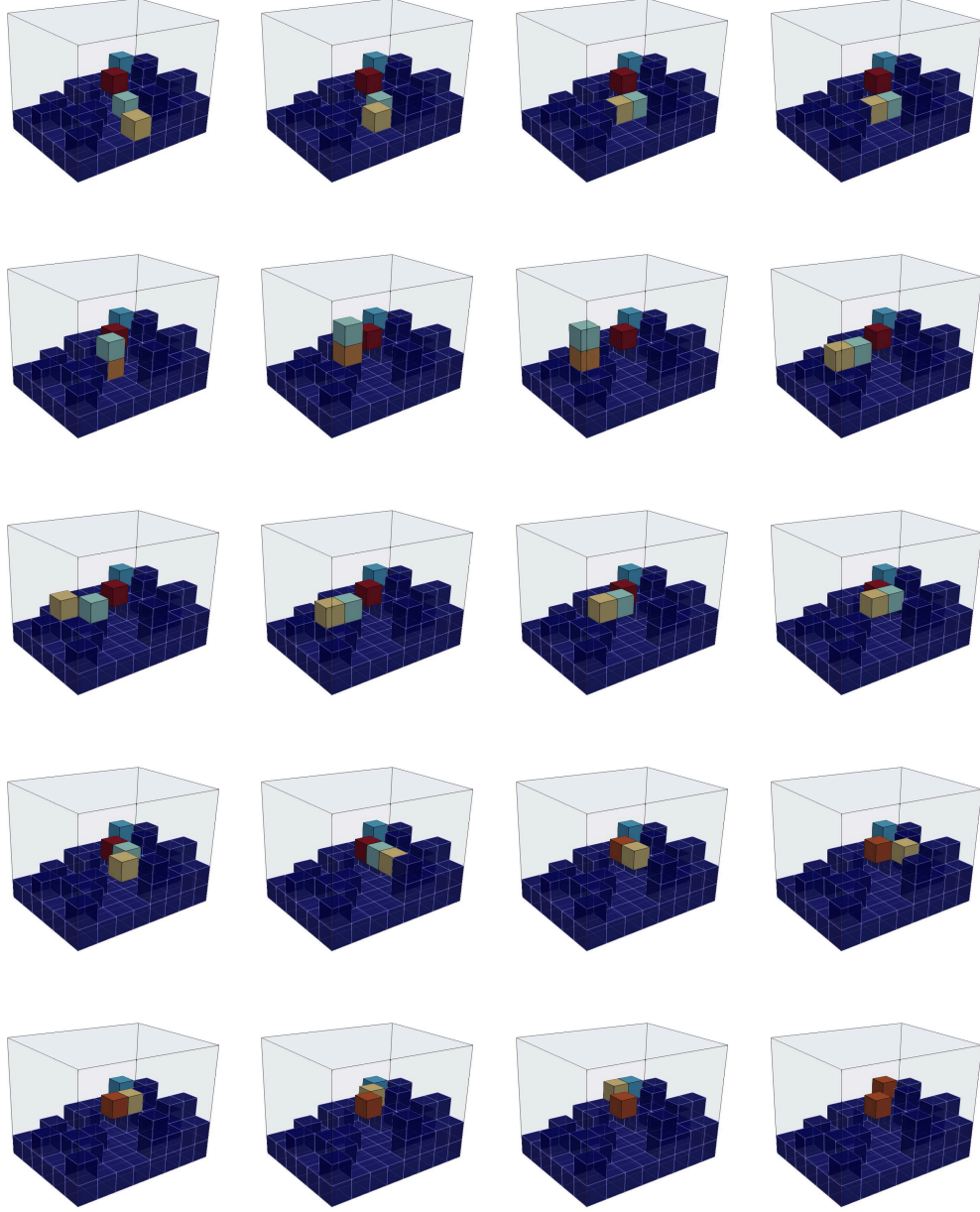


Figure 10: *Sokoban* $\times$ *Block Dude* ($6 \times 8 \times 6 \times 20$) - Block Dude (with below tile context for gravity) trained on row $\times$ col plane, and Block Dude trained on col $\times$ story plane. Initially, the crate and its target are at different elevation. Block Dude player is observed to pickup the crate and climb, and then drops the crate to push it into its target. (In this case, it was constrained to have exactly one player, one crate, one crate-target and one exit for the initial frame.)

■ Player, ■ Wall, ■ Crate, ■ Player-Pickup
■ Exit, ■ Crate-Target, ■ Terminal

4 Experiment and Analysis

To test blending across orthogonal planes, we used simple games like Field and Sokoban for proof of concept. Later on, we tested with relatively more complex games such as Block Dude and Snake,

to generate more interesting results. Here we describe the games briefly. We used our own constructed training levels for the games, using the diff method [31] leveraging the capabilities of Sturgeon [6]. We use simplified versions of some game mechanics. Each game

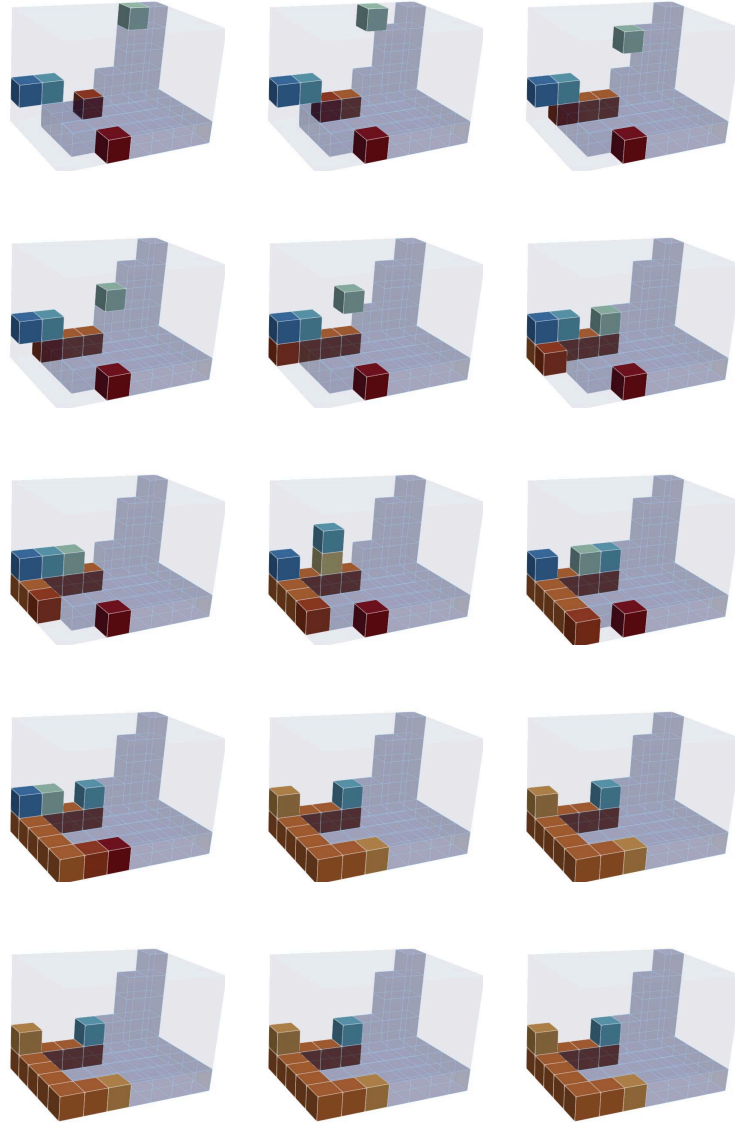


Figure 11: *Block Dude* $\times$ *Snake* ($5 \times 7 \times 5 \times 15$) - Block Dude (with below tile context for gravity) trained on row $\times$ col plane, and Snake trained on col $\times$ story plane. The player can be observed to walk on the snake and use it to reach the door. (In this case, it was constrained to have exactly one player, one snake-head, one block, one player-exit and one snake-exit for the initial frame. The player gets to pick the move, out of all possible moves to execute.)

■ Block-Player, ■ Snake-Head, ■ Snake-Body, ■ Snake-Exit
■ Player-Pickup, ■ Crate, ■ Player-Exit, ■ Terminal

has additional custom constraints on the number of elements in the level (such as having one player, a certain number of crates or targets, and similar), and we used the interactive editor (shown in Figure 6) to explore setting up interesting scenarios for the solver to complete.

Field — A simple game where the player navigates through an open field to reach the exit. The training examples had an effective size of $5 \text{ row} \times 5 \text{ col} \times 8 \text{ frames}$ and $8 \text{ row} \times 8 \text{ col} \times 13 \text{ frames}$.

Sokoban [30] — The player navigates through a space with walls to push a crate onto a target. The training examples had an

effective size of 5 row $\times$ 5 col $\times$ 10 frames and 6 row $\times$ 8 col $\times$ 25 frames.

Block Dude [8] — The player can pick up, move and climb up crates in order to reach the exit. We simplified by removing the player facing and not having crates falling due to gravity. When accounting for diff patterns, Block Dude includes a ‘context’ of the tile below the changed tiles to account for gravity. The training example had an effective size of 6 row $\times$ 20 col $\times$ 27 frames.

Snake — Controlling an infinitely growing snake around a maze, to reach an apple. The training example had an effective size of 5 row $\times$ 7 col $\times$ 13 frames.

We started with the simplest example level of Field. For training examples, we reorganized Field training level across three different planes with the following dimensions: 5 rows $\times$ 5 cols $\times$ 1 story $\times$ 8 frames (row $\times$ col plane), 5 rows $\times$ 1 col $\times$ 5 stories $\times$ 8 frames (row $\times$ story plane), and 1 row $\times$ 5 cols $\times$ 5 stories $\times$ 8 frames (col $\times$ story plane). The extracted patterns were then applied on a tag grid of dimensions 5 rows $\times$ 5 cols $\times$ 5 stories $\times$ 10 frames. The generated output was observed to have the same rules of player finding the exit expanded over to a 3D space.

The next step was to blend relatively complex games such as blending Sokoban with itself to get a 3D Sokoban. Sokoban-3D was trained in a similar way as Field-3D, by reorganizing the Sokoban training examples across three different planes. The generated output dimensions were 5 rows $\times$ 5 cols $\times$ 5 stories $\times$ 15 frames. Using the interactive editor, we were able to add custom constraints to achieve some interesting playthroughs. For example, adding an explicit wall barrier between the Sokoban player, and the crate and target. The solver was able to generate a solution having the player move to a different plane and come back in on the other side of the wall to push the crate into its target.

We also tried blending two separate games such as Sokoban merged with Block Dude. To approach this, the Sokoban patterns were extracted across the col $\times$ story plane and the Block Dude patterns in the row $\times$ col plane. This yielded interesting 3D results where the player was able to push the crate into its target (Sokoban rules), and could also pick up and put down the same crate, and reach the exit (Block Dude rules).

Doing this, also made us realize that the system could also blend different games into the same plane, like row $\times$ col. Some interesting examples of this are Sokoban $\times$ Field and Sokoban $\times$ Block Dude merged into the same plane. To extract the patterns, the training examples from all these games were organized in the same row $\times$ col orientation. These pattern were then merged and applied across a similar row $\times$ col $\times$ frame grid. In case of Sokoban $\times$ Field, the generated output showed adding a rule to the game of Sokoban, where the player will push the crate to its target and then find the exit to complete the level like in Field. In case of Sokoban $\times$ Block Dude, generating the output in the same plane showed some interesting mechanics, such as the player could disobey the gravity to push the crate(s) into the target(s), and then later on come down on the floor to try and find it’s way to reach the exit, which we thought had a potential to be an interesting mechanic.

Another interesting interplay between different games’ mechanics could be observed when we merged Block Dude with Snake. The Block Dude patterns were extracted across the row $\times$ col plane

and the Snake patterns were extracted over col $\times$ story plane. However, we desired that the Block Dude player be able to walk on the Snake’s snake tiles. To achieve this sort of equivalence, we trained Block Dude by switching wall tiles with the tiles from Snake tiles (head (H) and body (B)), so it could find patterns for the Block Dude player to walk on Snake tiles to reach the exit.

Figure 4 shows the first frame of some training examples. Various other Figures show generated outputs. Information about timing and solution length from various games and blends is given in Tables 1 and 2.

5 Discussion

We find that the process of blending games across orthogonal planes to generate new 3D games is, for relatively small levels and playthroughs, effective and performant enough to use, though scalability to larger sizes may remain an issue. We tested individual and blended game generation using the diff method on the games we tested for blending. This showcases the comparison between both. Looking at, for example, Field, Sokoban, and the blends of the two, blended levels unsurprisingly take more time to generate output for the expanded filmstrip. This could be due to the addition of extra tiles laterally, as the expanded filmstrip takes higher time to fill out those spaces with patterns. Different numbers of timesteps were used in some of the blended games, which also likely impacts the time to generate. This also suggests why 2D game generation, even with blending, are faster in comparison to the 3D games.

Looking at Block Dude for individual generation, the time taken to generate an output is very close to the time taken to generate a Sokoban $\times$ Block Dude blend in the same plane (close to 1 second), although the dimensions used are somewhat different. However, Block Dude blending with any other game in 3D is observed to have relatively higher generation time among all other blends. This could be due to the context in Block Dude training, to consider gravity into account, and the extra tiles due to relatively higher spatial and temporal dimensional values (Block $\times$ Snake takes 10.97 ± 0.61 and Soko $\times$ Block takes 24.4 ± 15.52 seconds to generate a result).

Effective length is the number of frames it takes to complete the level. The solver can tend to generate levels with simple solutions (e.g. placing the player next to the door, or having crates directly next to the target) if specific constraints (e.g. keeping the player and door apart, or keeping the crates away from the target) are not present. Thus not all of the temporal dimension may get used. This could also be avoided by using the interactive editor to add/remove constraints, until the designers find something they are happy with. However, the blended games (other than Field $\times$ Field) appear to use much of their frames in the solution, although this may also be impacted by the specific constraints applied to each setup.

6 Conclusion

In this work we presented a concept of blending multiple spatially 2D games along different planes to create a new spatially 3D game where the different games’ mechanics apply along different axes, and generate levels for these games. We learn mechanics from example playthroughs as local patterns of changes in tiles, and ensure solvability of generated levels using a constraint solving approach that embeds the levels in a 4D spacetime block.

There are many potential avenues for future work. One is playtesting of the generated games and levels to understand what players think of them. We would also like to scale the approach up to more complex games and generating larger spacetime blocks. We would also like to blend multiple spatially 3D games in a similar way, to make spatially 4D games (such as existing games like Miegakure [29]), which, including a time dimension, would make 5D spacetime blocks.

In this work the blended games still align along the time dimension. We speculate that it may be possible to blend games in such a way that their times do not align, resulting in possibly different time dimensions for different blended games, time dimensions that align but flow in different directions, or allowing movement forward and backward in time.

References

- [1] James Ahrens, Berk Geveci, and Charles Law. 2005. Paraview: An End-User Tool for Large Data Visualization. In *The Visualization Handbook*. Elsevier.
- [2] Shaad Alaka and Rafael Bidarra. 2023. Hierarchical Semantic Wave Function Collapse. In *Proceedings of the 18th International Conference on the Foundations of Digital Games (FDG '23)*. Association for Computing Machinery, New York, NY, USA, 1–10. doi:10.1145/3582437.3587209
- [3] Gilles Audemard and Laurent Simon. 2009. Predicting Learnt Clauses Quality in Modern SAT Solvers. In *Proceedings of the 21st International Joint Conference on Artificial Intelligence (IJCAI'09)*. Morgan Kaufmann Publishers Inc., San Francisco, CA, USA, 399–404.
- [4] Jake Bruce, Michael D. Dennis, Ashley Edwards, Jack Parker-Holder, Yuge Shi, Edward Hughes, Matthew Lai, Aditi Mavalankar, Richie Steigerwald, Chris Apps, Yusuf Aytar, Sarah Maria Elisabeth Bechtle, Feryal Behbahani, Stephanie C. Y. Chan, Nicolas Heess, Lucy Gonzalez, Simon Osindero, Sherjil Ozair, Scott Reed, Jingwei Zhang, Konrad Zolna, Jeff Clune, Nando de Freitas, Satinder Singh, and Tim Rocktäschel. 2024. Genie: Generative Interactive Environments. In *Forty-First International Conference on Machine Learning*.
- [5] Seth Cooper. 2022. Constraint-Based 2D Tile Game Blending in the Sturgeon System. In *Proceedings of the Experimental AI in Games Workshop*.
- [6] Seth Cooper. 2022. Sturgeon: tile-based procedural level generation via learned and designed constraints. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 18, 1 (2022), 26–36.
- [7] Seth Cooper and Mahsa Bazzaz. 2025. A Constraint-Based Graph Grammar Approach Unifying Level and Playthrough Generation. In *Proceedings of the Experimental AI in Games Workshop*.
- [8] Detached Solutions. 2001. *Block Dude*. <https://www.detachedsolutions.com/puzzpack/blockdude.php>
- [9] Niklas Eén and Niklas Sörensson. 2003. An Extensible SAT-solver. In *Theory and Applications of Satisfiability Testing*, Enrico Giunchiglia and Armando Tacchella (Eds.). Number 2919 in Lecture Notes in Computer Science. Springer Berlin Heidelberg, 502–518. doi:10.1007/978-3-540-24605-3_37
- [10] Maxim Gumin. 2016. WaveFunctionCollapse. <https://github.com/mxgmn/WaveFunctionCollapse>.
- [11] Alexey Ignatiev, Antonio Morgado, and Joao Marques-Silva. 2018. PySAT: a Python toolkit for prototyping with SAT oracles. In *Theory and Applications of Satisfiability Testing – SAT 2018*. 428–437.
- [12] Isaac Karth and Adam M Smith. 2017. WaveFunctionCollapse is constraint solving in the wild. In *Proceedings of the 12th International Conference on the Foundations of Digital Games*. 1–10.
- [13] Isaac Karth and Adam M. Smith. 2019. Addressing the fundamental tension of PCGML with discriminative learning. In *Proceedings of the 14th International Conference on the Foundations of Digital Games (FDG '19)*. Association for Computing Machinery, New York, NY, USA, 1–9. doi:10.1145/3337722.3341845
- [14] Thijmen Stefanus Leendert Langendam and Rafael Bidarra. 2022. miWFC-Designer empowerment through mixed-initiative Wave Function Collapse. In *Proceedings of the 17th International Conference on the Foundations of Digital Games*. 1–8.
- [15] Mark H. Liffiton and Jordyn C. Maglalang. 2012. A cardinality solver: more expressive constraints for free. In *Theory and Applications of Satisfiability Testing – SAT 2012*. 485–486.
- [16] C. Karen Liu, Aaron Hertzmann, and Zoran Popović. 2005. Learning Physics-Based Motion Style with Nonlinear Inverse Optimization. *ACM Trans. Graph.* 24, 3 (July 2005), 1071–1081. doi:10.1145/1073204.1073314
- [17] Mark J. Nelson and Adam M. Smith. 2016. ASP with applications to mazes and levels. In *Procedural Content Generation in Games*, Noor Shaker, Julian Togelius, and Mark J. Nelson (Eds.). Springer International Publishing, 143–157.
- [18] Jack Parker-Holder and Shlomi Fruchter. 2025. Genie 3: A New Frontier for World Models. <https://deepmind.google/blog/genie-3-a-new-frontier-for-world-models/>.
- [19] Anurag Sarkar and Seth Cooper. 2018. Blending Levels from Different Games Using LSTMs. In *Proceedings of the Experimental AI in Games Workshop*.
- [20] Anurag Sarkar and Seth Cooper. 2024. Latent Combinational Game Design. *IEEE Transactions on Games* 16, 3 (Sept. 2024), 659–669. doi:10.1109/TG.2023.3346331
- [21] Anurag Sarkar, Adam Summerville, Sam Snodgrass, Gerard Bentley, and Joseph Osborn. 2020. Exploring Level Blending across Platformers via Paths and Affordances. In *Proceedings of the Sixteenth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE'20)*. AAAI Press, 280–286.
- [22] Noor Shaker, Julian Togelius, and Mark J. Nelson. 2016. *Procedural Content Generation in Games*. Springer International Publishing.
- [23] A.J. Smith and J.J. Bryson. 2014. A Logical Approach to Building Dungeons: Answer Set Programming for Hierarchical Procedural Content Generation in Roguelike Games. *AISB 2014 - 50th Annual Convention of the AISB* (Jan. 2014).
- [24] Adam M. Smith, Erik Andersen, Michael Mateas, and Zoran Popović. 2012. A Case Study of Expressively Constraining Level Design Automation Tools for a Puzzle Game. In *Proceedings of the International Conference on the Foundations of Digital Games*. 156–163.
- [25] Adam M. Smith, Eric Butler, and Zoran Popovic. 2013. Quantifying over Play: Constraining Undesirable Solutions in Puzzle Design. In *Proceedings of the 8th International Conference on Foundations of Digital Games*. 221–228.
- [26] Adam M. Smith and Michael Mateas. 2011. Answer Set Programming for Procedural Content Generation: A Design Space Approach. *IEEE Transactions on Computational Intelligence and AI in Games* 3, 3 (2011), 187–200.
- [27] Thomas Smith, Julian Padgett, and Andrew Vidler. 2018. Graph-Based Generation of Action-Adventure Dungeon Levels Using Answer Set Programming. In *Proceedings of the 13th International Conference on the Foundations of Digital Games*. ACM, Malmö Sweden, 1–10. doi:10.1145/3235765.3235817
- [28] Adam Summerville, Anurag Sarkar, Sam Snodgrass, and Joseph C. Osborn. 2020. Extracting Physics from Blended Platformer Game Levels. In *Proceedings of the Experimental AI in Games Workshop*.
- [29] Marc ten Bosch. 2009. Miegakure.
- [30] Thinking Rabbit. 1982. Sokoban. Game.
- [31] Akshar Vandara, Kaylah Facey, and Seth Cooper. 2025. Spacetime Level Generation and Editing with Constraints from Examples. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* 21, 1 (Nov. 2025), 142–152. doi:10.1609/aiide.v21i1.36818
- [32] Andrew Witkin and Michael Kass. 1988. Spacetime Constraints. *ACM SIGGRAPH Computer Graphics* 22, 4 (June 1988), 159–168. doi:10.1145/378456.378507