Mortar: Evolving Mechanics for Automatic Game Design

Muhammad U. Nasir
University of the Witwatersrand
muhammad.nasir@wits.ac.za

Yuchen Li
New York University
yl6394@nyu.edu

Steven James
University of the Witwatersrand
steven.james@wits.ac.za

Julian Togelius
New York University
julian@togelius.com

Abstract

We present MORTAR, a system for autonomously evolving game mechanics for automatic game design. Game mechanics define the rules and interactions that govern gameplay, and designing them manually is a time-consuming and expert-driven process. MORTAR combines a quality-diversity algorithm with a large language model to explore a diverse set of mechanics, which are evaluated by synthesising complete games that incorporate both evolved mechanics and those drawn from an archive. The mechanics are evaluated by composing complete games through a tree search procedure, where the resulting games are evaluated by their ability to preserve a skill-based ordering over players—that is, whether stronger players consistently outperform weaker ones. We assess the mechanics based on their contribution towards the skill-based ordering score in the game. We demonstrate that MORTAR produces games that appear diverse and playable, and mechanics that contribute more towards the skill-based ordering score in the game. We perform ablation studies to assess the role of each system component and a user study to evaluate the games based on human feedback.

CCS Concepts

• **Computing methodologies** → **Natural language generation**; **Genetic algorithms**; Lifelong machine learning; *Simulation environments*; *Simulation evaluation*.

Keywords

Quality Diversity Optimization, Automated Game Generation, Large Language Models, Game Evaluation

ACM Reference Format:
Muhammad U. Nasir, Yuchen Li, Steven James, and Julian Togelius. 2026. Mortar: Evolving Mechanics for Automatic Game Design. In *Genetic and Evolutionary Computation Conference (GECCO '26), July 13–17, 2026, San Jose, Costa Rica.* ACM, New York, NY, USA, 9 pages. https://doi.org/10.1145/3795095.3805100

1 Introduction

Procedural content generation (PCG) is a well-studied approach in game design, concerned with the automatic creation of game content such as levels, maps, items and narratives [15, 27]. PCG serves multiple purposes: enabling runtime content generation in games such as roguelikes, providing ideation tools for designers, automating the production of repetitive content, and facilitating research into creativity and design processes. Traditionally, PCG research has focused on structural aspects of games—particularly level or layout generation—where the goal is to produce environments that are coherent, solvable, and varied [25].

By contrast, comparatively little attention has been paid to the procedural generation of *game mechanics*—the underlying rules for interactions that govern gameplay. Yet mechanics play a central role in shaping the player experience, determining not just how players act, but what kinds of strategies and emergent behaviours are possible. Designing mechanics is inherently challenging: unlike levels, which can be evaluated by solvability or novelty, the utility of a mechanic depends on the dynamics it induces within the context of a game. This makes generation and evaluation significantly harder.

A central premise of this work is that evaluating game mechanics is fundamentally more difficult than evaluating assets or level layouts. Unlike these forms of content, a mechanic *cannot be judged in isolation*—it only gains meaning through the gameplay it enables. A mechanic that appears novel or complex may still be uninteresting if it does not support skill-based interaction. This insight motivates our approach: effective automation of mechanic design requires not only a generative model, but also a principled way to assess a mechanic's utility in the context of play.

We address this challenge by introducing a mechanic-centric framework for automatic game design. The central idea is to evolve mechanics not in isolation, but through their contribution to the quality of full games. Specifically, we evaluate mechanics by constructing complete games around them, and measuring whether the resulting games induce a skill-based ordering over players of different capabilities. This allows us to define a concrete notion of importance for a mechanic: its contribution to the overall expressivity and skill gradient of the games in which it appears.

We introduce MORTAR[1], a system that evolves game mechanics using a quality-diversity algorithm guided by a large language model (LLM). As illustrated by Figure 1, MORTAR maintains a diverse archive of mechanics, represented as code snippets, which are mutated and recombined through LLM-driven evolutionary operators. Each evolved mechanic is evaluated by embedding it into full games constructed via Monte Carlo Tree Search [5], which incrementally builds games by composing mechanics from the archive. These games are evaluated on their ability to induce a consistent

GECCO '26, San Jose, Costa Rica

ACM ISBN 979-8-4007-2487-9/2026/07
https://doi.org/10.1145/3795095.3805100

[1] By definition MORTAR is a cement-based paste used in construction to bond masonry units like bricks, stones, and concrete blocks. We kept the name MORTAR as game mechanics is the paste that bonds the game together.

skill-based ranking over a fixed set of agents. We define a novel fitness measure, which quantifies the contribution of a mechanic to the final game's skill-based ordering, inspired by Shapley values [28].

We demonstrate that MORTAR can evolve a diverse set of game mechanics that contribute to the quality and playability of the generated games.[2] The resulting games exhibit coherent structure, varied interaction patterns, and meaningful skill gradients. Through ablations, we show that both the tree-search-based composition and the LLM-driven mutations are critical for generating high-quality mechanics. Our results highlight the potential for using LLMs not only as generators, but as evaluators and collaborators in the game design loop. We demonstrate that it is possible to do this in code space without a domain-specific language, and introduce a novel fitness measure.

The system described here is a research prototype for the purposes of understanding how to best generate complementary game mechanics. However, it could also serve as an ideation tool for game designers, suggesting new mechanics and mechanic combinations, perhaps in response to designer input. It is not meant to generate complete games, and aims to empower rather than replace game designers.

2 Method

MORTAR is an evolutionary algorithm for generating game mechanics, where an LLM is used to implement code-level variation operators. A core principle of the method is that a mechanic's value lies in the gameplay it enables; mechanics are evaluated not in isolation, but by the contribution they make to full games. We formalise this through a notion of *importance*, which guides the search process.

2.1 Evolution Setup

MORTAR employs a Quality-Diversity (QD) algorithm, using a fixed 2D archive (as in MAP-Elites [17]) to store and explore diverse game mechanics. We refer to this structure as the *Mechanics Archive*. Each mechanic is represented as a Python function belonging to a game class, and is placed in the archive based on two behavioural descriptors:

(1) **Mechanic Type**: A categorical descriptor indicating the gameplay behaviour the mechanic enables. We define 8 mechanic types (detailed in Section 3), each associated with 10 descriptive category words. To classify an evolved mechanic, we compute similarity scores between the mechanic's name and all category words, creating a normalised similarity vector. The mechanic type is determined by identifying the highest similarity score's index and multiplying this index by the score to produce a positional similarity value that serves as the behavioural descriptor.

(2) **Code Complexity**: Computed using weighted Abstract Syntax Tree (AST) analysis. We parse the mechanic's code into an AST representation and calculate complexity as a weighted sum of function calls, assignments, and return statements. Function calls receive the highest weight, as mechanisms requiring more function calls exhibit greater complexity. Assignments are weighted to reflect that additional variables may enable more interesting behaviours. Return statements contribute to complexity scoring because multiple exit paths can produce diverse behavioural outcomes.

Mechanics are selected from the archive and modified using several LLM-implemented evolutionary operators: *Mutation* adds new functionality to a single mechanic; *diversity mutation* samples three mechanics and prompts the LLM to generate a behaviorally distinct variant; *crossover* merges two mechanics (selected based on AST similarity) into a functional combination that integrates elements from both; and *compatibility mutation* generates mechanics that complement existing ones in a game, primarily used during game evaluation (see subsequent sections).

2.2 Evaluating Game Mechanics

Each evolved mechanic is represented as a function within a Python class. To prepare it for evaluation, we prompt the LLM to construct the rest of the class around it in a step-by-step fashion, starting with the `__init__()` method to define any required variables and scaffolding, `step` method to add actions, `reset` and `render` method as needed.

The mechanic is then tested for syntax and runtime errors. If no errors occur, we simulate it in a static test environment with simple objects and characters for the mechanic to interact with them, if necessary, by creating it as an OpenAI Gym interface [1]. We provide a basic character-based 2D level, which is linked to the Gym class.

A Monte Carlo Tree Search (MCTS) [5] agent is used to interact with the environment, verifying that the mechanic is functional. Only mechanics that pass both tests proceed to the *importance* evaluation stage. Failed mechanics are discarded to reduce unnecessary LLM calls.

2.3 Automated Game Construction

To evaluate a mechanic's importance, we embed it within a full game. Games are constructed through an `EvaluationMCTS`, where the root node is the evolved mechanic, and each expansion adds a new mechanic that is either sampled from the archive or generated via compatibility mutation. Each path through the tree represents a particular combination of mechanics; that is, a complete game. Algorithm 1 demonstrates the pseudo-code for the `EvaluationMCTS`.

Games are also implemented as Python classes, following a common template with core methods, such as `step`, `reset`, `render`, move mechanics and preset variables, similar to the mechanics evaluation step. Move mechanic is provided because we assume that it could be needed by many other mechanics. The LLM is prompted to modify or add functionality to these methods as needed, in an iterative manner. It is also asked to define any helper methods or variables required by the mechanics. At the end of this process, the LLM is prompted to define a win condition and generates a corresponding termination function. It also selects appropriate tiles from a predefined set, maps them to characters, and generates a 2D string-based level layout using these mappings. A final function defines which tiles are walkable, interactive, or character-specific.

The complete game script includes the game class, the tile and map generation functions, and a preset function that instantiates the full game. Simple postprocessing ensures the map is rectangular,

[2] Play the generated games at: https://mortar-x3p7.onrender.com/

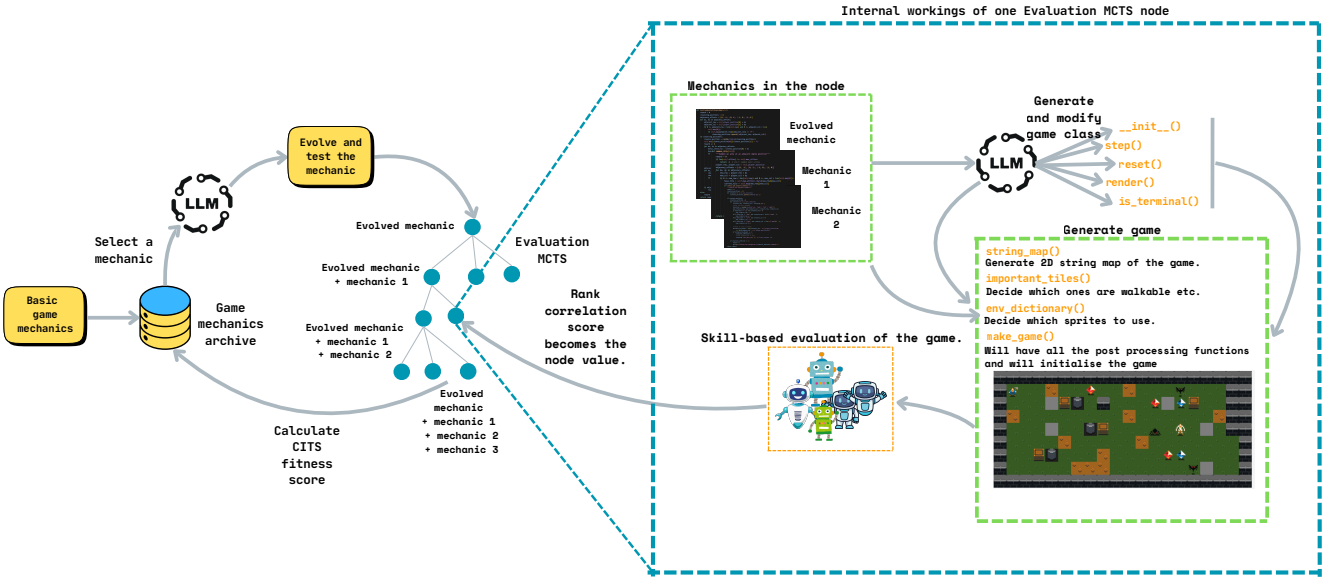


Figure 1: A flow diagram of MORTAR. Starting from a set of hand-designed basic mechanics, MORTAR selects and evolves a mechanic. The mechanic is tested and evaluated via the `EvaluationMCTS`, which generates games incorporating it alongside other mechanics. A *CITS* score quantifies the mechanic's contribution to these games and serves as its fitness for inclusion in the archive.

contains exactly one player, and is free of formatting issues (e.g. whitespace padding). See Section A of the supplementary material for all the prompts and helper functions.

2.4 Evaluation of the Game

A central idea in our work is that a game's quality is revealed through the emergence of a consistent skill gradient or game depth: a well-designed game should allow players of differing abilities to be meaningfully distinguished. This view is well-established in game design theory, where the length of a game's *skill chain* (the number of distinct skill levels that form a transitive ranking among players) is widely regarded as a core measure of depth [13, 32], and has been applied to automated game design by measuring performance gaps between AI agents of varying strength [2, 16].

We implement this by evaluating how well the game ranks a fixed set of players by skill. This approach allows us to evaluate not just whether a game is playable, but whether it rewards skill—a more robust signal of design quality.

To assess whether a game rewards skill, we fix a pool of five agents with varying ability levels, inspired by Nielsen et al. [23]. These include three MCTS agents with increasing numbers of rollouts, a random agent, and an agent that takes no actions.[3] This defines a clear expected skill ordering: the strongest agent should be the MCTS variant with the most rollouts, followed by the medium and low rollout agents, then the random agent, and finally the no-op agent. The outcome rank is induced by playing the game and recording empirical win rates. To quantify alignment between the expected and outcome rankings, we compute Kendall's Tau (τ), a standard measure of rank correlation: $\tau = \frac{C-D}{p(p-1)/2}$. Here, C and D are the number of concordant and discordant pairs, respectively, and p is the number of players (five in this case). Concordance occurs when the

[3] Any agents with a clear capability ordering would suffice, such as heuristic agents with different search depths, or reinforcement learning agents with varying training budgets.

Algorithm 1 `EvaluationMCTS`

Input: candidate mechanic, archive, max tree depth $D_{\max}$, max children $C_{\max}$
Initialize: root node with candidate mechanic
prev_mechanics ← [candidate mechanic]
for depth = 1 to $D_{\max}$ **do**
  for c = 1 to $C_{\max}$ **do**
    Sample action ∈ {select from archive, create new mechanic}
    if select from archive **then**
      new_mechanic ← pick mechanic from archive
    else
      new_mechanic ← generate and evaluate new mechanic via LLM
      if evaluation fails **then**
        new_mechanic ← pick mechanic from archive
      end if
    end if
    current_mechanics ← prev_mechanics + [new_mechanic]
    Expand current node with new_mechanic
    game ← create game with LLM using current_mechanics
    Compute Kendall's τ for game with five baseline agents
  end for
end for
Compute CITS scores for candidate mechanic and all generated mechanics
Return generated mechanics

relative ranking between two players agrees between the expected and observed orders; discordance occurs when they disagree. A value of 1 indicates perfect alignment with the expected ranking, 0 indicates no correlation, and -1 reflects a completely reversed order. We consider a game unplayable if $\tau = -1$.

While τ provides a global measure of game quality, it reveals nothing about the source of that quality. To address this, we introduce *Constrained Importance Through Search* (CITS), a scoring method to measure each mechanic's marginal contribution to the emergence of a skill gradient. Inspired by Shapley values [28], CITS estimates how much each mechanic contributed to the final game's τ score. However, computing full Shapley values would require evaluating every subset of mechanics, which is exponential in the number of mechanics. Instead, CITS is defined over the exploration tree constructed during generation, making it computationally tractable and grounded in actual gameplay evaluations. Formally, the CITS score for mechanic i is:

$$\text{CITS}_i = \frac{1}{|N_i|} \sum_{n \in N_i} \phi_i^{(n)},$$

where $N_i = \{n \in T : i \in M_n, n \neq n_{root}\}$ is the set of non-root nodes in the tree T that contain mechanic i, and M_n is the mechanic set at node n. The contribution $\phi_i^{(n)}$ is computed using the standard Shapley formula over the restricted set of explored subsets:

$$\phi_i^{(n)} = \sum_{S \subseteq M_n \setminus \{i\}} \frac{|S|! \cdot (|M_n| - |S| - 1)!}{|M_n|!} \cdot \Delta_i^{(n)}(S),$$

where the marginal value term is defined as the difference in value when adding mechanic i to the subset S:

$$\Delta_i^{(n)}(S) = v_T(S \cup \{i\}) - v_T(S).$$

Finally, the value function $v_T(S)$ returns the τ_m score for the node m with exactly mechanics S, if such a node exists in the tree; otherwise, it is defined to be 0:

$$v_T(S) = \begin{cases} \tau_m & \text{if } \exists m \in T \text{ s.t. } M_m = S \\ 0 & \text{otherwise} \end{cases}$$

This search-constrained Shapley approach allows us to assess a mechanic's value in context, measuring its contribution within actual, discovered game designs rather than hypothetical combinations. As such, the CITS score provides a principled, interpretable, and efficient mechanism for attributing gameplay quality to individual mechanics.

3 Experiment Setup

MORTAR employs a 2D Quality-Diversity (QD) archive with dimensions for mechanic type (0–1.0) and code complexity (4–40), forming a 13×13 grid. The first dimension categorises mechanics into nine types: *movement, interaction, combat, progression, environment, puzzle, resource management, exploration, time manipulation*. To categorise a mechanic, we use DistilBERT [26] embeddings to compute the similarity between mechanic function names and associated category words (detailed in Section C of the supplementary material). The complexity dimension and archive ranges were determined through experimentation to maximise archive coverage.

The system operates with a batch size of 10, selecting individuals from the archive and applying evolutionary operators in parallel. Operator selection probabilities are 50% for diversity mutation, 30% for mutation, and 20% for crossover. Diversity mutation samples three mechanics, while crossover selects pairs based on AST similarity. The static environment used to evaluate the evolved mechanic in isolation can be found in the Section A of the supplementary material.

For game construction, we use `EvaluationMCTS` with 20 iterations, where each expansion adds one mechanic per node (maximum 3 children per node). For the selection phase, we use UCT [12], but unlike traditional MCTS, we do not simulate; instead, we evaluate the complete game formed by all mechanics on the path from root to the newly expanded node, then backpropagate before proceeding to the next expansion. Compatibility mutation generates new mechanics within nodes, with a 50% probability of creating novel mechanics versus selecting from the existing archive. All LLM operations use GPT-4o-mini for both evolution and game creation. Skill assessment employs five agents with a clear capability ordering: MCTS variants with 100,000, 10,000, and 1,000 iterations, plus random and no-action agents for Kendall's Tau rank correlation computation.

We evaluate MORTAR through a series of targeted ablation studies designed to isolate the contributions of its core components. All experiments are averaged over five runs due to computational constraints (approximately $30–50 per run with GPT-4o-mini).

- First, we ablate the `EvaluationMCTS` procedure by replacing it with three alternatives: random mechanic selection, LLM-prompted selection, and greedy fitness-based selection. Each method generates games with 1–4 mechanics to compute CITS scores.
- Second, we ablate the quality–diversity mechanism by replacing the QD archive with either a standard genetic algorithm (GA) population or a purely random population. In both cases, populations are generated by an LLM. The GA variant follows the same evolutionary steps as MORTAR, but without maintaining an archive, while the random variant resamples a new population at each iteration, keeping all other steps identical.
- Third, we assess the system's ability to build upon existing human-designed content by initialising MORTAR with a *Sokoban* [18] level. This experiment evaluates whether the method can generate meaningful extensions and variations of an existing game, reflecting its potential use as an ideation tool for game designers.
- Fourth, we validate the CITS scoring function by comparing it to Shapley values [28]. We randomly select 20 generated games with at most three mechanics (due to computational cost) and compute Shapley values by evaluating all possible mechanic subsets. Kendall's τ rank correlation becomes the objective function here. We then find correlation coefficients between the CITS score and the Shapley values.

We track system performance using multiple complementary metrics. The Quality–Diversity (QD) score measures overall progress by summing fitness values across the archive. Accumulated rank correlation aggregates Kendall's τ scores across tree nodes. We additionally report maximum and mean CITS fitness, the number of

Table 1: Comparison of MORTAR with alternative mechanic selection strategies: LLM-based selection, random selection, and greedy fitness-based selection across quality-diversity metrics. All results are expressed as mean and standard deviation over 5 runs.

Method	No. of elites ↑	QD score ↑	Max CITS ↑	Mean CITS ↑	Games success rate ↑
MORTAR (ours)	**155 ± 4.51**	**31.18 ± 8.10**	**0.59 ± 0.11**	**0.20 ± 0.05**	16.97 ± 4.64
MORTAR-GA	100 ± 0.0	–	0.31 ± 0.17	0.13 ± 0.06	13.10 ± 2.00
Random population	100 ± 0.0	–	0.09 ± 0.07	0.03 ± 0.02	6.10 ± 2.50
LLM Selection	141 ± 5.83	17.64 ± 4.91	0.27 ± 0.09	0.13 ± 0.06	11.69 ± 5.14
Random Selection	144 ± 9.10	9.86 ± 6.71	0.14 ± 0.08	0.06 ± 0.04	11.77 ± 4.19
Greedy Selection	139 ± 5.15	25.37 ± 2.81	0.51 ± 0.06	0.18 ± 0.13	**18.24 ± 1.19**
Sokoban Initialisation	110 ± 10.52	15.19 ± 3.12	0.45 ± 0.12	0.19 ± 0.07	15.11 ± 2.83

elites retained in the archive, and the game creation success rate, defined as the proportion of functional games among all generated games.

Finally, we conduct a user study to assess the human-perceived quality of the generated games. Participants are presented with pairs of similar generated games and asked to play both before indicating which they find more *interesting*, *novel*, *fun to play*, and *easy to understand*. A *Neither* option is provided for each criterion to capture cases where no meaningful preference exists, allowing us to distinguish between genuine engagement and forced comparisons.

4 Results

In this section, we analyse results from the complete MORTAR pipeline, targeted ablation studies, and a human user study.

4.1 Ablation Results

We begin by analysing the behaviour of the full MORTAR pipeline before examining targeted ablations of its core components.

Overall performance. The QD score, which sums the fitness of all individuals in the archive, demonstrates MORTAR's ability to evolve increasingly better mechanics over time (Figure 2a). Complementary trends are shown in Figure 2b: mean CITS fitness increases steadily across generations, while maximum fitness improves in a stepwise manner, indicating continued discovery of novel, high-performing mechanics rather than premature convergence. Figure 2c provides further evidence of effective search behaviour. The accumulated Kendall's τ rank correlation across `EvaluationMCTS` trees increases over generations, showing that MORTAR progressively identifies games that induce meaningful, skill-based player rankings.

`EvaluationMCTS` ablation. We next examine the impact of the EvaluationMCTS component by comparing MORTAR against alternative evaluation strategies. Table 1 summarises results for random mechanic selection, LLM-prompted selection, and greedy fitness-based selection. Across all metrics, MORTAR achieves the highest QD score, maximum CITS score, and mean CITS score, while also attaining the greatest archive coverage (See Section C of the supplementary material for archive evolution). This indicates superior evolvability and a stronger ability to discover diverse, high-quality mechanics. Greedy selection attains a marginally higher game creation success rate, likely because it preferentially selects highly fit mechanics that are more likely to produce playable games. However, its lower diversity and reduced archive coverage highlight the limitations of purely exploitative search. Overall, these results demonstrate the effectiveness of tree search-based evaluation for mechanic composition.

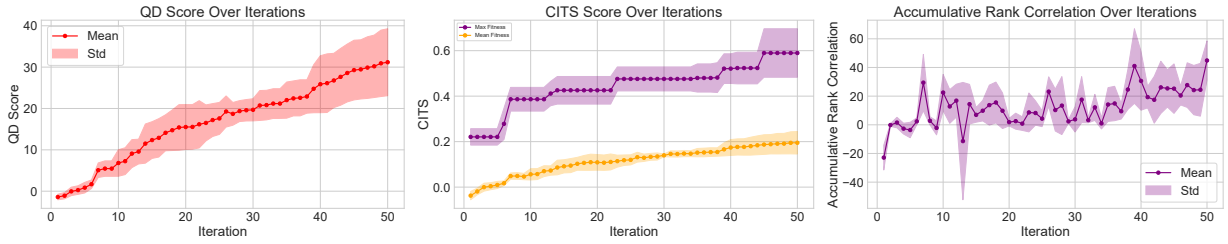
Archive and population ablation. We then ablate the QD mechanism by replacing the archive with either a genetic algorithm elitist-based archive or a random population. In both cases, performance degrades substantially relative to MORTAR, with fewer elites discovered and lower CITS scores overall. These results indicate that maintaining a QD archive plays a critical role in the overall system. Initialisation with a *Sokoban* level further reveals the sensitivity to starting mechanics and layouts. In this setting, we observe a markedly reduced number of elites, suggesting constrained evolvability when the initial design space is tightly structured. This highlights both the influence of initial conditions and the limits of mechanic discovery under strong priors. Figure 5 shows representative games produced under this initialisation.

CITS validation. Finally, we validate the CITS scoring function by comparing it against Shapley values [28]. Over 20 generated games, we observe a positive linear correlation with Shapley values (Pearson $r = 0.64$, $p = 0.002$), as well as a positive monotonic correlation (Spearman $\rho = 0.68$, $p = 0.001$) These results suggest that CITS provides a computationally efficient approximation of Shapley values, computing relative contributions without requiring evaluation over all possible subsets.

4.2 Qualitative Results

Figures 3 and 4 showcase two games generated by MORTAR, demonstrating diversity in level layouts, win conditions, and mechanics. *AllyCraft* (Figure 3) presents a challenging strategic experience where players control both their character and summoned allies, with escalating difficulty requiring versatile tactics. Effective strategies involve summoning allies strategically and eliminating enemies in optimal sequences. This game achieves a Kendall's τ of 0.8, maintaining clear agent rankings despite low overall rewards, with only minor rank switching between the do-nothing and random agents due to negative scoring.

By contrast, *TreasureHunt* (Figure 4) exhibits a Kendall's τ of 0.4, showing significant rank distortion except for the top-performing agent. This lower correlation suggests reduced strategic depth—once



(a) QD score **(b) CITS score** **(c) Accumulative Kendall's τ score.**

Figure 2: Performance metrics over evolutionary generations, with mean and standard deviation reported over five trials.

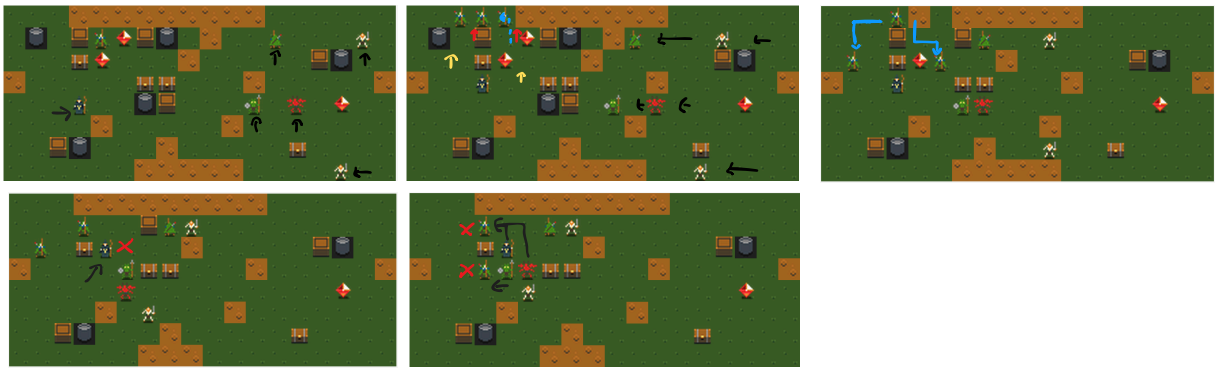


Figure 3: *AllyCraft* gameplay sequence. Top left: Initial state with enemies to defeat and items to collect. Top centre: The player spawns and deploys allied units. Top right: Allies collect items as enemies advance. Bottom left: An ally is defeated while eliminating an enemy unit. Bottom centre: A coordinated attack fails as the player is overwhelmed by enemies, resulting in a loss.

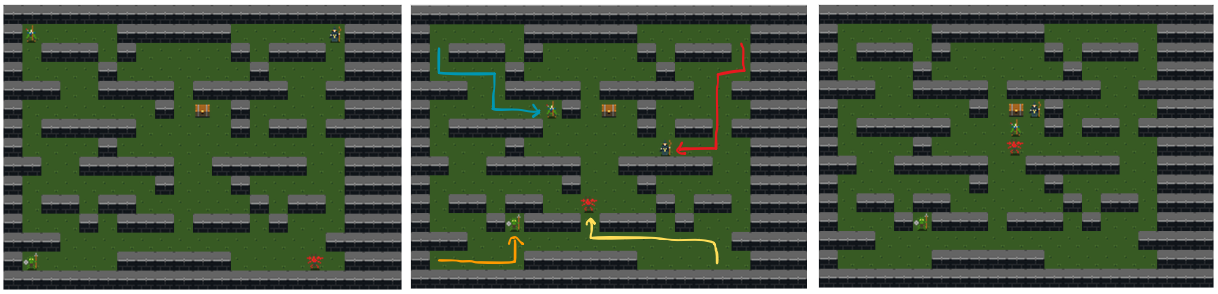


Figure 4: *TreasureHunt* gameplay sequence. Left: Initial state with a capture-the-flag–style treasure objective. Centre: Player spawns at the top-left (blue marker) as enemies begin pursuit. Right: Final state showing a close race between the player and a red-marked enemy, with the outcome determined by action processing order. Enemy movement is governed by an evolved A* pathfinding algorithm (see code in Section B of the supplementary material).

players discover the optimal path, the game loses replay value. *AllyCraft*'s higher τ score correlates with sustained engagement through multiple viable strategies, while *TreasureHunt*'s deterministic solution path limits long-term interest. Both games incorporate sophisticated mechanics, including ally summoning, multi-unit control, and pathfinding algorithms. The complete evolved code for these mechanics is provided in Section B of the supplementary material.

4.3 User study

To determine whether the quantitative metrics correlate with human preferences, we conducted a small comparative user study with 14 participants who evaluated 6 games generated by MORTAR across five dimensions: interestingness, novelty, frustration level, fun factor, and ease of understanding. Table 2 presents the results alongside each game's Kendall's τ score for comparison with MORTAR's automated evaluation.

The study compared three pairs of games: *TreasureHunt* versus *HuntBreakout* (capture-the-flag variants where *HuntBreakout* adds

Table 2: User study results comparing games. Values indicate the number of participants (out of 14) selecting each option. Total score represents the sum of positive metrics minus "Frustrating". Neither is calculated by subtracting all of the metrics by "Frustrating".

Games	Interesting ↑	Novel ↑	Frustrating ↓	Fun to play ↑	Easy to understand ↑	Total ↑	τ ↑
TreasureHunt	0	1	5	1	6	3	0.4
HuntBreakout	12	12	5	11	4	34	0.5
Both	1	0	0	1	4	6	—
Neither	1	1	4	1	0	3	—
AllyCraft	7	6	6	7	4	18	0.8
CrystalCavernsCommander	4	4	3	4	3	12	0.3
Both	1	2	5	0	1	-1	—
Neither	2	2	0	3	6	-13	—
MagneticProwess	4	4	5	4	4	11	0.6
HeroHunt	8	7	4	6	5	20	0.3
Both	0	0	2	1	3	2	—
Neither	2	3	3	3	2	-7	—

wall-breaking mechanics), *AllyCraft* versus *CrystalCavernsCommander* (RPG-style games differing in ally control mechanisms), and *MagneticProwess* versus *HeroHunt* (Sokoban-based games with magnetic pulling and enemy combat mechanics, respectively). See Section D of the supplementary material for games in the user study.

We observe a general correlation between the total human preference score and MORTAR'S calculated Kendall's τ values. In the first comparison, the τ difference has a smaller magnitude than the total score difference, yet both favour the same game. The second comparison shows alignment in both magnitude and direction between total score and τ. However, the third comparison reveals opposing trends where the total score contradicts τ, though this discrepancy may reflect the inherent difficulty of aligning automated skill-based metrics with subjective human preferences across diverse game genres.

Treating the total score as a meaningfulness metric—comprising interestingness, novelty, fun factor, ease of understanding, minus frustration—the "Neither" votes provide additional insight. These scores indicate that games in the second comparison are perceived as less meaningful, likely due to excessive complexity. This finding aligns with intuitive game design principles: mini-games benefit from appropriate rather than maximal complexity. While complexity can enhance engagement in full games through progressive difficulty scaling, these mini-game experiences demonstrate reduced effectiveness when sophisticated mechanics overwhelm fundamental gameplay elements. Qualitative participant feedback consistently highlighted visual limitations, particularly the lack of animations and restricted sprite sets—a known limitation of the current system.

4.4 Preliminary results with GPT5.1

Table 3 reports preliminary results for MORTAR using GPT-5.1. These results should be interpreted with caution, as they are based on a single run; however, the observed trends are consistent with the expected scaling behaviour. Relative to GPT-4o-mini, GPT-5.1 yields a substantially higher game creation success rate. The difference in maximum CITS score is not significant, though the absolute values remain high, with scores around 0.66 corresponding to games dominated by a single mechanic.

We additionally performed a short exploratory run of five generations using *Sokoban* initialisation. Figure 5 illustrates an example from this run, in which the player must push a box towards a red crystal target. A gravity mechanic is active, restricting upward movement to ladder tiles only. Notably, GPT-5.1 selected an invalid tile type for the ladder; however, the fallback mechanism produced a visually and functionally plausible substitute resembling a ladder segment.

Table 3: Preliminary investigation with GPT-5.1.

LLM	CITS (max)	Games Success Rate (%)
GPT-5.1	0.66	84
GPT-4o-mini	0.59 ± 0.11	16.97 ± 4.64

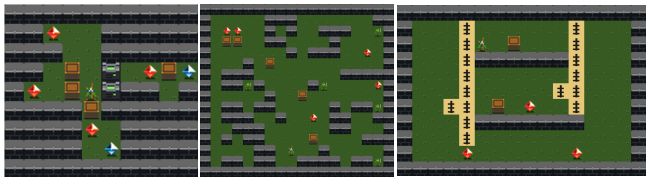


Figure 5: Games generated via *Sokoban* initialisation. Left: A *Sokoban*-style objective in which boxes must be placed on red crystal tiles; additional mechanics include teleportation from blue crystal tiles and a magnet that pulls boxes towards the player when standing on green machine tiles. Middle: A similar objective augmented with adversarial enemies; the episode terminates upon contact between an enemy and the player. Right: A gravity-based *Sokoban* variant generated by GPT-5.1 in which upward movement is restricted to ladder tiles.

5 Related Work

The core focus of MORTAR is evolving game mechanics to serve as an ideation and prototyping tool for game designers and generate novel games for testing learning algorithms. This research falls under Automatic Game Design (AGD), pioneered by Nelson and Mateas [22], who formalized game mechanics through WordNet to generate micro games. Browne [3] and Togelius and Schmidhuber [35] independently proposed evolutionary approaches to AGD across different domains. The latter introduced learnability as a quality criterion, inspiring various approximations of skill differentiation over the years [11, 23] that influence our current approach. Sumner et al. [30] introduced Mechanic Maker, a mechanics generation pipeline that used a Domain-Specific Language (DSL) to integrate the mechanic in the game and then created an automated frontend generation as well for the mechanic. Our work differs from Sumner et al. [30] as we generate mechanics Python code while using LLMs. Later on, Gonzalez et al. [7] used reinforcement learning to evaluate the generated mechanics in Mechanic Maker 2.0. A very closely related work is Pixie, introduced by Cook [4], which used genetic programming-style code evolution to generate mechanics. This work is closely related to MORTAR as we also evolve code for the game mechanics. MORTAR'S Constrained Importance Through Search (CITS) is related to Green et al. [8] work, which uses playtrace information, through comparing different MCTS setups, to find important game mechanics.

Non-evolutionary AGD approaches include constraint solvers for mechanics generation [37] and autoencoders for learning and generating mechanics [24]. Recent work incorporates LLMs into AGD pipelines: ScriptDoctor generates PuzzleScript games [6], while Gavel evolves Ludii games using LLMs and Quality-Diversity algorithms [34]. Gavel differs in the aspect that it evolves board games with a specific DSL, therefore it is limited to the expressiveness of the DSL. Similar approaches have generated 2-player games using XML-based languages [10]. MORTAR distinguishes itself by leveraging the full expressiveness of Python code generation, creating a search space that scales with advancing LLM capabilities.

MORTAR also relates to LLM-driven procedural content generation [15, 27, 36]. Early work included Sokoban level generation using GPT-2/3 [33], MarioGPT for Super Mario Bros levels with novelty search [29], and human-in-the-loop GPT-3 fine-tuning [21]. Word2World and Word2Minecraft generate 2D and 3D games with fixed mechanics [9, 20]. MORTAR extends this paradigm by generating multiple game aspects, including mechanics and levels.

Finally, MORTAR contributes to research on LLMs as evolutionary operators in Quality-Diversity (QD) algorithms like MAP-Elites [17]. This approach has been applied to robot morphology evolution [14], and neural architecture search [19]. Notably, GAVEL [34] uses a QD algorithm to evolve board games, which comes closest to our work. MORTAR differs in many ways: MORTAR is the first of its kind to evolve game mechanics in Python Programming language, and then evaluate using a novel `EvaluationMCTS` tree. Furthermore, MORTAR also proposes a different measure of importance for the game mechanic through CITS score, which is a cheaper approximation to Shapley values [28].

6 Limitations

While MORTAR successfully generates novel game mechanics and coherent games with semantically meaningful CITS scores, several limitations warrant future investigation. The system currently modifies game rendering functions without incorporating animations, limiting visual richness. Our experiments used a relatively modest LLM (GPT-4o-mini), and a preliminary result with GPT5.1; stronger models could potentially yield more sophisticated mechanics and improved code quality. The current 2D top-down perspective constrains the search space—extending to 3D environments would significantly expand creative possibilities.

Archive initialisation presents another challenge, as improved seeding strategies could enhance convergence and final quality. Similarly, increasing MCTS iterations during evaluation might produce higher-quality games at the cost of computational resources. Perhaps most significantly, MORTAR's autonomous evolution lacks designer control mechanisms. A controllable variant that accepts design constraints or preferences could better serve as an ideation tool, allowing game developers to guide the search toward specific gameplay goals while maintaining the system's creative discovery capabilities.

7 Conclusion And Future Directions

We present MORTAR, a novel system for generating games through mechanic evolution. MORTAR combines MAP-Elites, a Quality-Diversity algorithm, with LLM-driven code-level mechanic evolution. The system evaluates mechanics through MCTS, which constructs complete games in each tree node and assesses them using skill-based ranking. We introduce the Constrained Importance Through Search (CITS) score, derived from Shapley values, which quantifies a mechanic's contribution within the actually searched combination space rather than hypothetical alternatives.

Our quantitative results demonstrate MORTAR's high evolvability and progressive improvement across generations through ablation studies. Qualitative analysis reveals that games with higher scores exhibit greater strategic depth and complexity, while MORTAR consistently produces diverse gaming experiences with sophisticated mechanic interactions.

MORTAR offers several promising research directions. As an ideation tool, it could support game designers by suggesting novel mechanic combinations responsive to design constraints. The system's scalability suggests that initialisation with extensive mechanic libraries and extended evolution periods could explore previously undiscovered regions of game design space. Finally, the generated games could serve as rich environments for testing generalisation in reinforcement learning agents [31], offering diverse challenges with measurable skill gradients.

References

[1] Greg Brockman, Vicki Cheung, Ludwig Pettersson, Jonas Schneider, John Schulman, Jie Tang, and Wojciech Zaremba. 2016. Openai gym. *arXiv preprint arXiv:1606.01540* (2016).

[2] Cameron Browne. 2022. Quickly detecting skill trace in games. In *2022 IEEE Conference on Games (CoG)*. IEEE, 604–607.

[3] Cameron Bolitho Browne. 2008. *Automatic generation and evaluation of recombination games*. Ph. D. Dissertation. Queensland University of Technology.

[4] Michael Cook. 2025. Pixie: Code-Level Mechanic Generation for Game Designers. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 21. 206–215.

[5] Rémi Coulom. 2006. Efficient selectivity and backup operators in Monte-Carlo tree search. In *International conference on computers and games*. Springer, 72–83.

[6] Sam Earle, Ahmed Khalifa, Muhammad Umair Nasir, Zehua Jiang, Graham Todd, Andrzej Banburski-Fahey, and Julian Togelius. 2025. ScriptDoctor: Automatic Generation of PuzzleScript Games via Large Language Models and Tree Search. *arXiv preprint arXiv:2506.06524* (2025).

[7] Johor Jara Gonzalez, Seth Cooper, and Matthew Guzdial. 2023. Mechanic Maker 2.0: reinforcement learning for evaluating generated rules. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 19. 266–275.

[8] Michael Cerny Green, Ahmed Khalifa, Gabriella AB Barros, Tiago Machado, and Julian Togelius. 2019. Automatic Critical Mechanic Discovery in Video Games. *arXiv preprint arXiv:1909.03094* (2019).

[9] Shuo Huang. 2025. *Word2Minecraft: Generating 3D Game Levels through Large Language Models*. Master's thesis. New York University Tandon School of Engineering.

[10] Ruiz-Quiñones Jorge and Fernández-Leiva Antonio J. 2023. Automated videogame mechanics generation with XVGDL. *ICGA Journal* 44, 4 (2023), 124–152.

[11] Ahmed Khalifa, Michael Cerny Green, Diego Perez-Liebana, and Julian Togelius. 2017. General video game rule generation. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 170–177.

[12] Levente Kocsis and Csaba Szepesvári. 2006. Bandit based monte-carlo planning. In *European conference on machine learning*. Springer, 282–293.

[13] Frank Lantz, Aaron Isaksen, Alexander Jaffe, Andy Nealen, and Julian Togelius. 2017. Depth in Strategic Games.. In *AAAI Workshops*.

[14] Joel Lehman, Jonathan Gordon, Shawn Jain, Kamal Ndousse, Cathy Yeh, and Kenneth O Stanley. 2023. Evolution through large models. In *Handbook of evolutionary machine learning*. Springer, 331–366.

[15] Jialin Liu, Sam Snodgrass, Ahmed Khalifa, Sebastian Risi, Georgios N Yannakakis, and Julian Togelius. 2021. Deep learning for procedural content generation. *Neural Computing and Applications* 33, 1 (2021), 19–37.

[16] Jialin Liu, Julian Togelius, Diego Pérez-Liébana, and Simon M Lucas. 2017. Evolving game skill-depth using general video game ai agents. In *2017 IEEE Congress on Evolutionary Computation (CEC)*. IEEE, 2299–2307.

[17] Jean-Baptiste Mouret and Jeff Clune. 2015. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909* (2015).

[18] Yoshio Murase, Hitoshi Matsubara, and Yuzuru Hiraga. 1996. Automatic making of sokoban problems. In *Pacific Rim International Conference on Artificial Intelligence*. Springer, 592–600.

[19] Muhammad Umair Nasir, Sam Earle, Julian Togelius, Steven James, and Christopher Cleghorn. 2024. Llmatic: neural architecture search via large language models and quality diversity optimization. In *proceedings of the Genetic and Evolutionary Computation Conference*. 1110–1118.

[20] Muhammad U Nasir, Steven James, and Julian Togelius. 2024. Word2world: Generating stories and worlds through large language models. *arXiv preprint arXiv:2405.06686* (2024).

[21] Muhammad U Nasir and Julian Togelius. 2023. Practical PCG through large language models. In *2023 IEEE Conference on Games (CoG)*. IEEE, 1–4.

[22] Mark J Nelson and Michael Mateas. 2007. Towards automated game design. In *Congress of the Italian Association for Artificial Intelligence*. Springer, 626–637.

[23] Thorbjørn S Nielsen, Gabriella AB Barros, Julian Togelius, and Mark J Nelson. 2015. General video game evaluation using relative algorithm performance profiles. In *European Conference on the Applications of Evolutionary Computation*. Springer, 369–380.

[24] Bernhard Rieder. 2018. Using procedural content generation via machine learning as a game mechanic. *Austrian Marshall Plan Foundation* (2018).

[25] Sebastian Risi and Julian Togelius. 2020. Increasing generality in machine learning through procedural content generation. *Nature Machine Intelligence* 2, 8 (2020), 428–436.

[26] Victor Sanh, Lysandre Debut, Julien Chaumond, and Thomas Wolf. 2019. DistilBERT, a distilled version of BERT: smaller, faster, cheaper and lighter. *arXiv preprint arXiv:1910.01108* (2019).

[27] Noor Shaker, Julian Togelius, and Mark Nelson. 2016. *Procedural Content Generation in Games*. Springer.

[28] Lloyd Shapley. 2016. 17. A Value for n-Person Games. In *Contributions to the Theory of Games, Volume II*. Princeton University Press, 307–318.

[29] Shyam Sudhakaran, Miguel González-Duque, Matthias Freiberger, Claire Glanois, Elias Najarro, and Sebastian Risi. 2023. Mariogpt: Open-ended text2level generation through large language models. *Advances in Neural Information Processing Systems* 36 (2023), 54213–54227.

[30] Megan Sumner, Vardan Saini, and Matthew Guzdial. 2024. Mechanic maker: accessible game development via symbolic learning program synthesis. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 20. 235–244.

[31] Richard S Sutton, Andrew G Barto, et al. 1999. Reinforcement learning. *Journal of Cognitive Neuroscience* 11, 1 (1999), 126–134.

[32] J Mark Thompson. 2015. Defining the abstract. *Game & Puzzle Design* 1, 1 (2015), 83–86.

[33] Graham Todd, Sam Earle, Muhammad Umair Nasir, Michael Cerny Green, and Julian Togelius. 2023. Level generation through large language models. In *Proceedings of the 18th International Conference on the Foundations of Digital Games*. 1–8.

[34] Graham Todd, Alexander G Padula, Matthew Stephenson, Éric Piette, Dennis J Soemers, and Julian Togelius. 2024. Gavel: Generating games via evolution and language models. *Advances in Neural Information Processing Systems* 37 (2024), 110723–110745.

[35] Julian Togelius and Jurgen Schmidhuber. 2008. An experiment in automatic game design. In *2008 IEEE Symposium On Computational Intelligence and Games*. IEEE, 111–118.

[36] Julian Togelius, Georgios N Yannakakis, Kenneth O Stanley, and Cameron Browne. 2011. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games* 3, 3 (2011), 172–186.

[37] Alexander Zook and Mark Riedl. 2014. Automatic game design via mechanic generation. In *Proceedings of the AAAI Conference on Artificial Intelligence*, Vol. 28.

Received 20 February 2007; revised 12 March 2009; accepted 5 June 2009