EduGameGen: A Framework for Educational Game Generation with Large Language Models

Markus Schatten
University of Zagreb Faculty of Organization and Informatics
Artificial Intelligence Laboratory
Pavlinska 2, 42000 Varadin, Croatia
markus.schatten@foi.unizg.hr

Abstract. *This work-in-progress paper presents EduGameGen, a design science artifact for prompt-driven creation and evaluation of educational game prototypes in PyGame using large language models (ChatGPT). It extends the Role–Context–Command–Format prompt structure by decomposing the context into priority-ordered educational, technical/usability, and game-design aspects, and includes a teacher-friendly rubric aligned with these aspects for non-technical evaluators. An artificial, researcher-led evaluation comprised 24 iterative prompt-refinement attempts across five educational game types. Final prototypes achieved a mean score of 32.60/36 with no weak rubric scores, suggesting structured context engineering and rubric-based iteration support rapid prototyping, while classroom evaluation remains necessary for assessing learning effectiveness.*

Keywords. educational games, prompt engineering, ChatGPT, PyGame, PygBag, vibe coding, design science research, RCCF prompt format, educational game evaluation

1 Introduction

Educational games can enrich teaching through interaction, feedback, narrative, challenge, and repeated practice. However, their classroom value depends on more than playfulness: a useful educational game must be aligned with learning outcomes, appropriate for learners' age and prior knowledge, technically usable, and feasible within the time constraints of a lesson. This is a demanding combination for teachers, who often know their subject, students, language, and classroom context well, but may not have the time, programming skills, or artistic resources needed to create custom games. As a result, the practical adoption of game-based learning and gamification remains limited by the availability of games that match concrete teaching needs (Clark et al., 2016; Dicheva et al., 2015; Hamari et al., 2014).

The need for custom-tailored games is particularly clear in primary and secondary education. Teachers may want a short game for introducing cyberbullying, practicing Python syntax, discussing traffic safety, or exploring mechanical systems. Such games should reflect the curriculum, learner characteristics, cultural context, accessibility needs, and available teaching materials. Off-the-shelf educational games rarely satisfy all of these requirements simultaneously. Generative artificial intelligence and Large Language Models (LLMs) therefore create a new opportunity: through prompt-based or "vibe coding" workflows, teachers may be able to describe a learning situation in natural language and obtain a playable prototype. Yet educational game generation is not only a coding task. It requires coordinated control over educational content, technical constraints, game mechanics, visual assets, and deployment.

This paper proposes *EduGameGen*, a design science artifact for prompt-driven educational game prototyping with ChatGPT and PyGame. The framework builds on the Role–Context–Command–Format (RCCF) prompt structure and expands its context component into priority-ordered educational, technical/usability, and game-design aspects, following the broader idea that structured prompt engineering can improve the quality and controllability of generated outputs (Korzyski et al., 2023; Sahoo et al., 2024; White et al., 2023). In its extended form, EduGameGen supports a multiphase workflow: generation of a short game development document (GDD), PyGame code, and graphics specification; optional generation and quality control of visual assets; and final integration for local execution or browser deployment using PygBag (Pygame Community, 2026; pygame-web contributors, 2026).

The study continues the author's previous work on gamification, educational games, design science approaches to game-based systems, and cognitive agents in learning environments (Saari & Schatten, 2024; Schatten & Schatten, 2020; Schatten, 2024; Schatten, Peharda, et al., 2022; Schatten & Schatten, 2019; Schatten et al., 2021, 2023; Tomii & Schatten, 2020). Its goal is not to claim that LLMs can replace teach-

ers or game developers, but to investigate whether a structured prompt framework can help transform teachers' instructional intentions into small, playable, and evaluable prototypes. The paper contributes: (1) the EduGameGen prompt-construction framework; (2) its extension toward graphics specification, quality control, integration, and PygBag-compatible deployment; and (3) a teacher-friendly rubric for evaluating generated prototypes along educational, technical/usability, and game-design dimensions. The artifact is evaluated through an artificial, researcher-led design science research (DSR) evaluation based on 24 iterative prompt-refinement attempts across five educational game types.

2 Related Work

EduGameGen builds on research in educational games, prompt engineering, and AI-assisted game creation. Prior game-based learning and gamification studies emphasize that game elements are educationally useful only when they are aligned with learning activities, feedback, and instructional goals rather than added as decoration (Clark et al., 2016; Dicheva et al., 2015; Hamari et al., 2014). Recent work on LLMs in education extends this argument by suggesting that playful and game-based learning can help preserve practice, motivation, and human judgment while using generative AI, but also stresses the need for domain expertise and critical evaluation of generated outputs (Huber et al., 2024; Kasneci et al., 2023). EduGameGen follows this line by treating the teacher not as a passive recipient of generated code, but as the author of the educational context and evaluator of the resulting prototype.

A second stream concerns LLMs in and for games. Surveys describe rapidly growing work on LLM-supported game design, narrative generation, NPCs, player modelling, and game development tools (Gallotta et al., 2024; Sweetser, 2024). Related educational work has used LLMs as interactive agents inside meaningful games; for example, Prompt-Gaming explored an LLM-evaluating agent in an energy game and reported potential for easy prototyping of educational games (Isaza-Giraldo et al., 2024). In contrast, the present work does not embed an LLM inside the running game. Instead, it uses the LLM as a design and development partner before deployment, producing static PyGame/PygBag prototypes that can be evaluated and revised before classroom use.

The closest studies are recent papers on ChatGPT-assisted educational game creation. Borisova et al. propose a methodology for creating educational games with ChatGPT and discuss examples and risks in school computer-technology education (Borisova et al., 2024). Hadzhikoleva et al. similarly describe an AI-driven, iterative process in which teachers formulate requirements, use multiple prompt cycles, and test the resulting game code (Hadzhikoleva et al., 2024). Spasi et al. focus explicitly on teachers without coding experience and show how a carefully designed prompt can generate a browser-based memory game (Spasi et al., 2025). EduGameGen extends these works by formalizing the prompt as a priority-ordered educational, technical/usability, and game-design context; adding a multiphase workflow for GDD generation, graphics specification, quality control, integration, and web deployment; and linking prompt construction to a teacher-friendly evaluation rubric.

Methodologically, the paper is situated in DSR, where artifacts are built and evaluated with respect to a problem context (Hevner et al., 2004; Peffers et al., 2007). This continues the author's previous work on educational games, gamification, cognitive agents, and design-oriented game systems (Saari & Schatten, 2024; Schatten & Schatten, 2020; Schatten, 2024; Schatten, Peharda, et al., 2022; Schatten & Schatten, 2019; Schatten et al., 2021, 2023; Tomii & Schatten, 2020). The present contribution differs by focusing on prompt construction and evaluation as the central artifact, rather than on a single educational game or game-based system.

3 EduGameGen Framework

EduGameGen is a prompt-construction framework for generating small educational games with LLMs. It adapts the RCCF structure to educational game development: the *Role* defines the model as an educational game designer, instructional designer, and Python/PyGame developer; the *Command* requests a complete playable prototype; and the *Format* fixes the expected output, including the concept, code, controls, running instructions, and self-checklist. The main contribution is the *Context* component, which is decomposed into optional educational, technical/usability, and game-design aspects, as described in Table 1.

All context fields are optional. This is important because teachers may begin with only a topic, a learning outcome, or a short description of a classroom need. Empty fields are treated as unspecified and may be inferred by the model. However, the fields are ordered by priority: when a prompt contains many clues or conflicting requirements, earlier items should guide the generation more strongly than later ones. This makes the framework usable both as a minimal prompt scaffold and as a detailed specification instrument.

The educational context expresses what the game should teach and to whom. Its highest-priority elements are the learning outcome and content, since these constrain the correctness and relevance of the generated game. The technical/usability context limits the implementation space. In the present work, this is intentionally restricted to PyGame to keep the generated games inspectable, modifiable, and suitable for teachers with limited technical support. The game-design context turns the learning activity into gameplay by

Table 1. Priority-ordered EduGameGen context model.

Context group	Optional aspects, ordered from higher to lower priority
Educational	learning outcome; learning materials/content; topic; learner profile; prior knowledge; learner diversity and accessibility considerations; required feedback; assessment/progress logic; cognitive level; misconceptions; subject/domain.
Technical/usability	external dependencies; external assets; code structure; runtime environment; interaction method; screen and interface requirements; accessibility requirements; testing/running instructions.
Game design	core mechanic; link between mechanic and learning outcome; goal; rules; feedback loop; challenge; win/loss conditions; progression; reward/scoring; session length; genre; aesthetics/theme; narrative.

defining player actions, goals, rules, feedback, and progression. The explicit requirement to state the link between the core mechanic and the learning outcome is intended to prevent purely decorative gamification, such as adding points or a story without changing the learning activity.

The current version of EduGameGen further extends the basic RCCF prompt into a three-phase workflow. In the first phase, the model produces a short GDD, a complete PyGame prototype, and, when external graphics are desired, a separate graphics specification. During this phase, only placeholder graphics are used so that the logic of the game can be checked independently from the final visual assets. In the second phase, the specified graphics are generated and quality-controlled against the GDD and graphics specification. In the third phase, the final assets are integrated into the game, the prototype is rechecked, and the deliverable files are prepared.

This multiphase structure was introduced because educational games often require both executable logic and coherent visual presentation. For example, a visual-novel game about cyberbullying may require consistent characters, scenes, speech bubbles, and alternative endings. Separating the generation of code and visual assets reduces the risk that graphics generation distracts from educational alignment or that attractive images hide incomplete gameplay. The final integration step may embed generated assets directly into the Python file, for example by using base64 encoding, which supports the goal of producing self-contained prototypes.

Finally, EduGameGen treats browser deployment as a design requirement rather than a later technical afterthought. The generated games are expected to run locally and, where possible, in a web browser through PygBag, which packages Python/PyGame projects for execution in modern browsers (pygame-web contributors, 2026). This enables the creation of static, client-side educational games that do not require a custom backend. For schools, this is practically important: a game can be hosted as ordinary web content on a school website or learning platform, making it easier for teachers to share custom-tailored prototypes with pupils without installing Python or maintaining a server application.

Table 2. Generated educational game types.

Game	Description
Cyberbullying visual novel	A fifth-grade visual novel about cyberbullying. The story starts when pupils share humiliating meme images of a classmate. The player, also a pupil in the class, chooses how to react; the branching narrative leads to multiple endings.
Portals rogue-like	A teacher-facing dungeon crawler projected in class. When the teacher-controlled player touches a monster, the class answers a question about educational portal websites for second-grade Informatics. Correct answers remove monsters; finding the treasure after at least three correct answers gives the class a reward.
Traffic RPG	A second-grade role-playing game (RPG) about traffic rules and signs. The pupil-player solves quests, collects hidden traffic-related items, answers quizzes, gains experience, and levels up; each level changes the player's class and appearance.
Python 3D puzzle	A programming game focused on Python `for` loops. The player controls 3D objects by writing code with predefined `move` and `rotate` functions, aiming to pass objects through holes across ten progressively harder tasks with hints.
Cogwheels sandbox	A mechanical sandbox for experimenting with cogwheels of different sizes, direct connections, and straight or crossed belts. Any cogwheel can become a motor; the simulation shows rotation speeds and detects mechanical jams.

4 Evaluation Method

EduGameGen was evaluated through an artificial, formative DSR evaluation rather than immediate classroom deployment (Peffers et al., 2007). The goal was not to measure learning gains, but to examine whether the framework can support the generation of prototypes that are technically feasible, pedagogically plausible, and sufficiently game-like for later pilot use with teachers and learners.

The evaluation used 24 generated prototypes across five game types. For each game type, the prompt was iteratively refined until a usable prototype was obtained. A prototype was considered acceptable only if none of the twelve rubric criteria received a score below adequate. The five final games, shown in Fig. 1, were intentionally diverse in topic, genre, audience, and interaction style, as summarized in Table 2.

The rubric follows the three EduGameGen context dimensions. Educational quality includes clarity of the learning objective, correctness of content, alignment between learning and activity, and feedback. Technical/usability quality includes whether the prototype runs, ease of use, readability/accessibility, and stability. Game-design quality includes clarity of goals and rules, challenge/progression, engagement, and quality of the core mechanic. Each criterion was scored from 0 to 3, resulting in a maximum of 36 points. The evaluation was researcher-led and should therefore be interpreted as evidence of prototype feasibility and design

Table 3. Final prototype scores.

Prototype	E	T	G	Total	Class
Cyberbullying visual novel	12	11	12	35	Strong
Portals rogue-like	12	12	11	35	Strong
Traffic RPG	11	11	11	33	Strong
Python 3D puzzle	9	10	9	28	Usable
Cogwheels sandbox	10	10	12	32	Strong

Table 4. Prompt-refinement effort by game type.

Game type	Attempts	First	Final	Gain
Cyberbullying visual novel	8	18	35	+17
Portals rogue-like	2	30	35	+5
Traffic RPG	9	21	33	+12
Python 3D puzzle	4	18	28	+10
Cogwheels sandbox	1	32	32	0

plausibility rather than evidence of learning effectiveness.

5 Results

Across all 24 attempts, the mean score was 25.46 out of 36. Non-final attempts achieved a mean score of 23.58, whereas the final accepted prototypes achieved a mean score of 32.60. Four final prototypes were classified as strong, and one was classified as usable, as shown in Table 3. All final prototypes met the stricter acceptance criterion of having no individual rubric criterion below adequate.

The number of required attempts differed notably across game types, as presented in Table 4. The cogwheels sandbox reached an acceptable result in one attempt, while the portals rogue-like required two attempts. More complex or less straightforward game types required more refinement: the Python 3D puzzle required four attempts, the cyberbullying visual novel required eight attempts, and the traffic RPG required nine attempts. The largest improvement was observed for the cyberbullying visual novel, which increased from 18 to 35 points, followed by the traffic RPG, which increased from 21 to 33 points.

The most frequent weak points across all attempts were ease of use, game mechanic quality, readability/accessibility, clarity of goals and rules, and learning–activity alignment. In contrast, all attempts received strong scores for being runnable prototypes and for stability during play. This suggests that the main bottleneck was not generating executable PyGame code, but shaping the generated artifacts into coherent, usable, and educationally aligned games. The screenshots in Fig. 1 illustrate the diversity of genres and interfaces produced by the same framework, ranging from code-oriented puzzles to visual-novel and sandbox mechanics.

6 Discussion

The findings return to the central motivation of this paper: teachers often need short, curriculum-specific games, but do not usually have the time, programming experience, or graphical resources required to build them from scratch. EduGameGen addresses this gap by turning a teacher's instructional intention into a structured prompt specification. The five prototypes show that the same framework can support substantially different classroom purposes: a branching story for discussing cyberbullying, a teacher-led dungeon crawler for whole-class questioning, an RPG for traffic-safety quests, a code-based 3D puzzle for Python loops, and a mechanical sandbox for experimentation. This diversity is important because it suggests that the framework is not tied to a single genre or subject area, but to a reusable way of specifying educational, technical/usability, and game-design context.

Compared with prior work on ChatGPT-assisted educational game creation (Borisova et al., 2024; Hadzhikoleva et al., 2024; Spasi et al., 2025), EduGameGen makes three practical advances. First, it formalizes the prompt context as a priority-ordered and optional structure rather than as a single monolithic prompt. This allows a teacher to start with minimal information, such as a topic and learning outcome, while still supporting detailed specification when materials, learner characteristics, accessibility needs, game mechanics, visual style, or deployment constraints are known. Second, it explicitly connects the construction of the prompt with evaluation through the same three dimensions. The rubric therefore does not merely score finished games, but helps identify which part of the prompt should be refined. Third, the framework extends beyond code generation by including GDD creation, graphics specification, optional image generation and quality control, and final integration.

The evaluation also clarifies the role of LLMs in this process. The results do not support a simple one-shot view of educational game generation. Several prototypes required multiple iterations before reaching the acceptance threshold, and the most frequent weak points concerned ease of use, game mechanic quality, readability/accessibility, clarity of goals and rules, and learning–activity alignment. These weaknesses are precisely the aspects that teachers and instructional designers must be able to inspect. At the same time, all attempts were runnable and stable, indicating that the main challenge was less the generation of PyGame code and more the transformation of code into a coherent educational game. In this sense, EduGameGen advances the idea of prompt engineering as a digital competence (Korzyski et al., 2023; Sahoo et al., 2024; White et al., 2023): the teacher is not expected to program the game manually, but must still articulate educational intent, judge the output, and iteratively improve the specification.

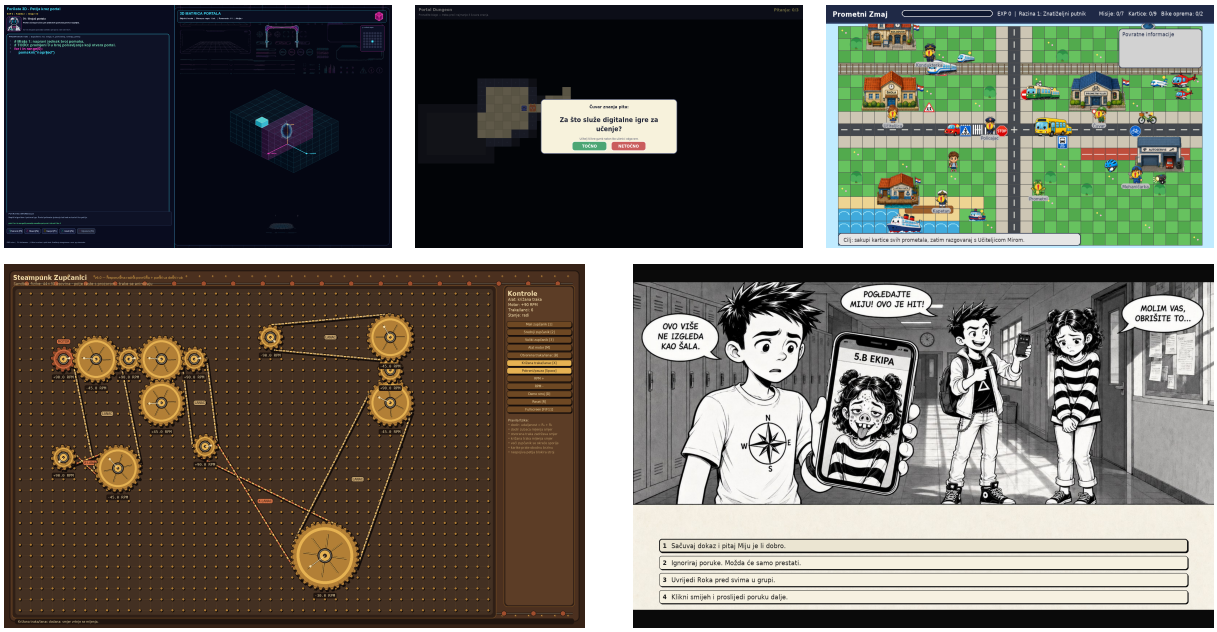


Figure 1. Screenshots of the final generated prototypes: Python 3D puzzle, portals rogue-like, traffic RPG, cogwheels sandbox, and cyberbullying visual novel.

The browser-deployment requirement further strengthens the practical relevance of the framework. By targeting PyGame prototypes that can be packaged with PygBag, EduGameGen supports games that run as static client-side applications without a custom backend. This is particularly suitable for schools, where installation rights, server administration, and platform heterogeneity can become barriers to classroom use. A generated game can therefore be distributed through a school website or learning platform, making it easier for teachers to share tailor-made prototypes with pupils and colleagues.

Finally, the results should be interpreted as evidence of feasibility and design plausibility, not as evidence of learning effectiveness. The evaluation was artificial and researcher-led, and the prototypes were not tested with teachers or learners. Nevertheless, this is consistent with an early DSR cycle (Hevner et al., 2004; Peffers et al., 2007): the study demonstrates an artifact, applies it across diverse cases, and identifies concrete refinement needs. The next step is naturalistic evaluation in classrooms, where the framework should be studied not only in terms of generated game quality, but also in terms of teacher workload, ease of prompt authoring, learner engagement, and learning outcomes.

7 Conclusion and Future Research

This paper presented EduGameGen, a design science artifact for prompt-driven educational game prototyping with ChatGPT and PyGame. The framework responds to the need for lightweight, teacher-accessible ways of creating custom educational games that are tailored to a specific subject, learner group, classroom situation, and learning outcome. Rather than treating prompt writing as an ad hoc activity, EduGameGen structures the process through an RCCF-based prompt format, an optional and priority-ordered context model, a three-phase generation workflow, and a rubric for formative evaluation.

The main contribution is the connection between prompt construction, artifact generation, and rubric-based refinement. The context model decomposes the prompt into educational, technical/usability, and game-design aspects, while allowing any field to remain unspecified when the teacher does not have the time or expertise to define it. The three-phase workflow extends the framework beyond code generation by including GDD creation, optional graphics generation and quality control, and final integration. By targeting PyGame prototypes that can be deployed through PygBag, the framework also supports browser-playable games that can be distributed as static web applications, which is especially relevant for school websites and learning platforms.

The artificial evaluation across 24 prompt-refinement attempts and five game types provided encouraging initial evidence. The final prototypes achieved a mean score of 32.60 out of 36, and all satisfied the acceptance criterion of having no weak rubric score. The results indicate that the framework can support diverse educational game genres, including visual novels, rogue-likes, RPGs, programming puzzles, and sandbox simulations. At the same time, the evaluation showed that useful prototypes usually emerge through iteration. The most frequent weaknesses concerned usability, game mechanics, readability/accessibility, clarity of goals and rules, and

learning–activity alignment, while technical execution and stability were less problematic.

The study has several limitations. The evaluation was researcher-led and artificial, the number of game types was limited, and the results do not measure actual learning outcomes. Furthermore, the framework was evaluated only with PyGame-based prototypes and within a small set of Croatian educational scenarios. The findings should therefore be interpreted as evidence of feasibility and design plausibility, not as evidence that the generated games improve learning.

Future research will focus on naturalistic classroom evaluation. Teachers should use EduGameGen to design games for their own lessons, after which both teacher and student feedback can be collected to refine the framework. Such studies should examine ease of prompt authoring, teacher workload, classroom usability, student engagement, accessibility, and perceived educational value. In later stages, the games should also be evaluated in terms of learning outcomes and compared with alternative teaching activities. Additional work should investigate inter-rater reliability of the rubric, refine the context fields and priority rules, and explore how the graphics-generation and PygBag deployment phases can be made more robust for everyday school use.

A EduGameGen Prompt Template

The following compact template, provided in Listing 1, gives the main EduGameGen structure for reuse. In normal use, only the **Context** block should be edited; all its fields are optional and may be left empty.

Listing 1. Copyable EduGameGen prompt template.

```
# MULTI-PHASE PROMPT FOR CREATING AN EDUCATIONAL PYGAME GAME

## MAIN GOAL
Create a small, complete, and playable educational game prototype in Python/PyGame
    ↪ .

The work is performed in up to three phases:
1. Phase 1 - Game concept, GDD, code, and graphics specification
2. Phase 2 - Graphics generation and quality control
3. Phase 3 - Integration, final checking, and delivery of files

Do not tell the user which phase you are in unless explicitly asked.

# MULTI-PHASE WORKFLOW RULES

## Workflow decision
If the user asks for generated graphics, then:
1. Perform Phase 1 only.
2. Do not generate images in Phase 1.
3. Use only programmatically created placeholder graphics in PyGame during Phase
    ↪ 1.
4. Create a file named generate_graphics.md containing the graphics specification
    ↪ and image-generation prompts.
5. After finishing the GDD, code, and graphics specification, inform the user that
    ↪  the program code is ready.
6. Ask whether the user wants to continue with graphics generation.
7. Do not continue to Phase 2 until the user explicitly confirms.
8. Do not offer the .py and .md files for download during Phase 1.

If the user does not ask for generated graphics, then:
1. Skip Phase 2.
2. After Phase 1, proceed directly to Phase 3.
3. In Phase 3, deliver the final files and summary.

Treat graphics generation as requested if the user mentions generated visual
    ↪ elements, sprites, scenes, illustrations, comic panels, backgrounds,
    ↪ character images, graphics, or other external visual resources.

# GENERAL RULES

* Do not generate images during Phase 1 or Phase 3.
* If visual elements are needed during game development, use PyGame placeholder
    ↪ graphics.
* If generated graphics are used, generate them only in Phase 2.
* Respect any visual, narrative, or genre style specified by the user.
* If educational, technical, and aesthetic requirements conflict, prioritize the
    ↪ learning outcomes.
* The game must be complete, understandable, and playable.
* Do not leave obvious TODO notes, unfinished systems, or unimplemented GDD
    ↪ elements.
* After generating the code, check whether everything specified in the GDD has
    ↪ been implemented. If not, fix the code before answering.
* The code must be in a single Python file unless the user explicitly requests
    ↪ otherwise.
* Avoid external dependencies other than PyGame.

# PHASE 1

## ROLE
Act as an expert educational game designer, instructional designer, and Python/
    ↪ PyGame developer. If the user specifies a particular visual, narrative
    ↪ , or genre style, also act as an expert in that style. Your goal is to
    ↪  create a small playable educational game prototype that aligns game
    ↪ mechanics with learning outcomes.

## CONTEXT
# Only edit this Context block. All fields are optional. Leave unknown or
    ↪ irrelevant fields empty.

### Educational context
* Subject/domain:
* Topic:
* Learning outcome(s):
* Cognitive level / learner age:
* Learning materials or content to include, if any:
* Learner profile:
* Prior knowledge:
* Learner diversity or accessibility considerations:
* Common misconceptions:
* Required educational feedback:
* Assessment or progress logic:

### Technical context
* Code structure: one file
* External dependencies: preferably none except PyGame
* External resources / generated assets:
* GUI or screen requirements:
* Interaction method:
* Runtime environment:
* Browser deployment: the game should be runnable locally and in a web browser
    ↪ through PygBag, without requiring a backend.
* Accessibility requirements:
* Testing or running instructions:

### Game-design context
* Genre:
* Core mechanic:
* Link between mechanic and learning outcome(s):
* Goal:
* Rules:
* Challenge:
* Progression:
* Feedback loop:
* Reward or scoring system:
* Win/loss condition:
* Aesthetics/theme:
* Session length:
* Narrative:

### Constraints and priorities
* Must-have requirements:
* Nice-to-have requirements:
* Avoid:
* If there is a conflict, prioritize:

## COMMAND
Before generating code, develop a short and realistic Game Design Document (GDD)
    ↪ and save it as gdd.md.

Create a complete, executable PyGame prototype of the educational game described
    ↪ in gdd.md. The game should be simple enough to understand quickly, but
    ↪  complete enough to be playable. Include educational content, player
    ↪ interaction, feedback, scoring or progress tracking, and a clear end
    ↪ condition.

The game should be runnable locally and in a web browser through PygBag. For local
    ↪  execution, use fullscreen mode unless otherwise specified. For
    ↪ browser execution, set an appropriate fixed resolution.

After generating the code, check whether all GDD elements have been implemented.
    ↪ If anything is missing, revise the code before the final response.
    ↪ Repeat this check until the implementation matches the specification.

If graphics generation was requested, do not offer generated .py or .md files for
    ↪ download in Phase 1. Offer downloads only in Phase 3 after graphics
    ↪ generation and integration.

# PHASE 2

If graphics generation was requested in Phase 1, open generate_graphics.md and
    ↪ generate all graphics according to the specified prompts. If the
    ↪ number of required images exceeds the system limit, generate the
    ↪ maximum possible number, then ask the user whether to continue. If
    ↪ generation continues in several rounds, use the first generated set as
    ↪  a reference to preserve consistency of characters, style, proportions
    ↪ , and important objects.

After generating the graphics, inspect each image and perform quality control
    ↪ against generate_graphics.md and gdd.md. Check whether each image can
    ↪ be integrated into the game. If any image does not match the prompt,
    ↪ has the wrong resolution, or is unsuitable for integration, adjust the
    ↪  prompt and regenerate the image.

After graphics generation is complete, inform the user and ask whether to start
    ↪ the final integration. Continue to Phase 3 only after the user
    ↪ confirms.

# PHASE 3

## FORMAT
Based on the previous phases, return the final answer in this structure:
```

```
1. Brief game concept
2. Short explanation of how the game mechanic supports the learning outcome(s)
3. Complete Python/PyGame code in a single file; provide a download link instead
       ↪ of printing the full code. If generated graphics were requested,
       ↪ integrate the graphics into the code by converting all images to
       ↪ base64 and embedding them directly in the .py file. If other necessary
       ↪  files exist, convert and embed them as well. Before integration,
       ↪ inspect all graphics and adjust the code to ensure appropriate size,
       ↪ resolution, cropping, and visual quality.
4. Controls and running instructions
5. Short checklist confirming whether the educational, technical, and game-design
       ↪ requirements were satisfied
```

Acknowledgement

This research has been financed by NextGenEU project AI-ARENA Framework for Ethical use of AI Agents in Multidisciplinary Research of the Digital Society (Okvir za etiku upotrebu AI agenata u multidisciplinarnim istraivanjima digitalnog drutva).

The authors acknowledge that ChatGPT 5.5 by OpenAI has been used to assist in editing and streamlining sections of this paper.

References

Borisova, M., Hadzhikoleva, S., & Hadzhikolev, E. (2024). ChatGPT as a tool for creating educational games for school education in computer technology. *Proceedings of the International Conference on Virtual Learning*, *19*, 323–333. https://doi.org/10.58503/icvl-v19y202427

Clark, D. B., Tanner-Smith, E. E., & Killingsworth, S. S. (2016). Digital games, design, and learning: A systematic review and meta-analysis. *Review of Educational Research*, *86*(1), 79–122. https://doi.org/10.3102/0034654315582065

Dicheva, D., Dichev, C., Agre, G., & Angelova, G. (2015). Gamification in education: A systematic mapping study. *Educational Technology & Society*, *18*(3), 75–88.

Gallotta, R., Todd, G., Zammit, M., Earle, S., Liapis, A., Togelius, J., & Yannakakis, G. N. (2024). Large language models and games: A survey and roadmap. *arXiv preprint arXiv:2402.18659*. https://doi.org/10.48550/arXiv.2402.18659

Hadzhikoleva, S., Gorgorova, M., Hadzhikolev, E., & Pashev, G. (2024). AI-driven approach to educational game creation. *Web Proceedings of the 16th International Conference ICT Innovations*, 112–123. https://proceedings.ictinnovations.org/attachment/paper/601/ai-driven-approach-to-educational-game-creation.pdf

Hamari, J., Koivisto, J., & Sarsa, H. (2014). Does gamification work? a literature review of empirical studies on gamification. *Proceedings of the 47th Hawaii International Conference on System Sciences*, 3025–3034. https://doi.org/10.1109/HICSS.2014.377

Hevner, A. R., March, S. T., Park, J., & Ram, S. (2004). Design science in information systems research. *MIS Quarterly*, *28*(1), 75–105. https://doi.org/10.2307/25148625

Huber, S. E., Kiili, K., Nebel, S., Ryan, R. M., Sailer, M., & Ninaus, M. (2024). Leveraging the potential of large language models in education through playful and game-based learning. *Educational Psychology Review*, *36*(1), 25. https://doi.org/10.1007/s10648-024-09868-z

Isaza-Giraldo, A., Bala, P., Campos, P. F., & Pereira, L. (2024). Prompt-gaming: A pilot study on LLM-evaluating agent in a meaningful energy game. *Extended Abstracts of the CHI Conference on Human Factors in Computing Systems*, 1–12. https://doi.org/10.1145/3613905.3650774

Kasneci, E., Sessler, K., Küchemann, S., Bannert, M., Dementieva, D., Fischer, F., Gasser, U., Groh, G., Günnemann, S., Hüllermeier, E., Krusche, S., Kutyniok, G., Michaeli, T., Nerdel, C., Pfeffer, J., Poquet, O., Sailer, M., Schmidt, A., Seidel, T., … Kasneci, G. (2023). ChatGPT for good? on opportunities and challenges of large language models for education. *Learning and Individual Differences*, *103*, 102274. https://doi.org/10.1016/j.lindif.2023.102274

Korzyski, P., Mazurek, G., Krzypkowska, P., & Kurasinski, A. (2023). Artificial intelligence prompt engineering as a new digital competence: Analysis of generative AI technologies such as ChatGPT. *Entrepreneurial Business and Economics Review*, *11*(3), 25–37. https://doi.org/10.15678/EBER.2023.110302

Peffers, K., Tuunanen, T., Rothenberger, M. A., & Chatterjee, S. (2007). A design science research methodology for information systems research. *Journal of Management Information Systems*, *24*(3), 45–77. https://doi.org/10.2753/MIS0742-1222240302

Pygame Community. (2026). *Pygame: Python game development*. Retrieved July 6, 2026, from https://www.pygame.org/

pygame-web contributors. (2026). *Pygbag: Pygame wasm*. Retrieved July 6, 2026, from https://pygame-web.github.io/wiki/pygbag/

Saari, S., & Schatten, M. (2024). Integrating cognitive agents into visual novels. *Proceedings of the 35th Central European Conference on Information and Intelligent Systems*, 27–32.

Sahoo, P., Singh, A. K., Saha, S., Jain, V., Mondal, S., & Chadha, A. (2024). A systematic survey of prompt engineering in large language models: Techniques and applications. *arXiv preprint arXiv:2402.07927*. https://doi.org/10.48550/arXiv.2402.07927

Schatten, M., & Schatten, M. (2020). A comparative study of gamification in programming education in a croatian high school. *2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO)*, 700–704. https://doi.org/10.23919/MIPRO48935.2020.9245237

Schatten, M. (2024, February). *O_hai 4 games - d4.4. case study 4 - hologame* (tech. rep. No. AIL202401) (Technical Report No. AIL202401 – First release, February 2024, edit April 2024. This work has been supported in full by the Croatian Science Foundation under the project number IP-2019-04-5824.). Laboratorij za umjetnu inteligenciju, Fakultet organizacije i informatike, Sveuilite u Zagrebu. Varadin, Croatia. https://urn.nsk.hr/urn:nbn:hr:211:989793

Schatten, M., Okrea uri, B., & Peharda, T. (2021). An agent-based game engine layer for interactive fiction. *Highlights in Practical Applications of Agents, Multi-Agent Systems, and Social Good. The PAAMS Collection*, 385–389. https://doi.org/10.1007/978-3-030-85739-4_38

Schatten, M., Peharda, T., et al. (2022). Towards a streamed holographic 3d game engine. *Central European Conference on Information and Intelligent Systems*, 17–22.

Schatten, M., & Schatten, M. (2019). Gamification of game programming education: A case study in a croatian high school. *Central European Conference on Information and Intelligent Systems*, 13–18.

Schatten, M., Schatten, M., & Inkret-Martinevi, L. (2023). Development of gamified cognitive agents for primary schools – a case study. *Proceedings of the 34th Central European Conference on Information and Intelligent Systems*, 3–8.

Spasi, A. J., Mladenovi, M., Nikoli, B., & Stoi, L. (2025). ChatGPT-assisted educational game development: A prompt engineering technique for teachers without coding experience. *Proceedings of the 1st International Scientific Conference Education and Artificial Intelligence (EDAI 2024)*, 153–164. https://doi.org/10.46793/EDAI24.153S

Sweetser, P. (2024). Large language models and video games: A preliminary scoping review. *arXiv preprint arXiv:2403.02613*. https://doi.org/10.48550/arXiv.2403.02613

Tomii, I., & Schatten, M. (2020). A conceptual network analysis of gamification practices in primary and secondary education. *2020 43rd International Convention on Information, Communication and Electronic Technology (MIPRO)*, 1091–1097. https://doi.org/10.23919/MIPRO48935.2020.9245225

White, J., Fu, Q., Hays, S., Sandborn, M., Olea, C., Gilbert, H., Elnashar, A., Spencer-Smith, J., & Schmidt, D. C. (2023). A prompt pattern catalog to enhance prompt engineering with ChatGPT. *arXiv preprint arXiv:2302.11382*. https://doi.org/10.48550/arXiv.2302.11382