TOWARDS A GRAPH-THEORETICAL FRAMEWORK FOR MODELING AND ANALYZING ESCAPE GAME MECHANICS

Gonzague Yernaux, Martin Verjans and Wim Vanhoof
Faculty of Computer Science
University of Namur
Rue Grandgagnage 21, 5000 Namur, Belgium
e-mail: gonzague.yernaux@unamur.be

KEYWORDS
Escape Game Modeling, Game Mechanics Formalization, Graph Representations, Game Analysis.

ABSTRACT

In this work, we propose a comprehensive graph-theoretical framework to formally model the structure and dynamics of escape games. Our approach introduces the `Static Graph`, a directed graph encoding the topological and logical organization of puzzles, clues, rooms, and player roles. To capture player interaction and game progression over time, we also define the `Dynamic Graph` that reflects the current game session state, along with a so-called game session forest that encapsulates all possible execution traces depending on player decisions. This layered representation enables automated analysis of game solvability, balance, and puzzle dependencies. We illustrate the framework with a proof-of-concept visualization and simulation tool, `GraphEG`. The tool supports both manual design and runtime interaction with escape game scenarios.

INTRODUCTION

Escape games, also known as escape rooms, have evolved from niche entertainment in Japan in the early 2000s to a global phenomenon used in team building, cognitive training and education (Romero and al. 2021). Their appeal lies in the combination of immersive storytelling, collaborative puzzle-solving, and time-bound challenges, which in pedagogical contexts encourages engagement and knowledge retention (Taraldsen et al. 2022). Existing frameworks such as *EscapED* (Clarke and Peel 2017) and the *Star Model* (Botturi and Babazadeh 2020) provide guidelines for integrating educational objectives into game mechanics, to create instances of what is often called *serious games*.

Apart from educational science, research on escape games includes endeavors in human–computer interaction and, more broadly, in game design theory. Foundational taxonomies (Nicholson 2015) and puzzle classifications (Wiemker, Elumir, and Clare 2015) exist that address narrative framing, collaboration, and engagement. Yet, as highlighted by Taraldsen et al.'s recent work, many of the internal mechanics specific to (serious) escape games remain largely informal (Taraldsen et al. 2022). This lack of formalization limits the possibilities to study game reproducibility and puzzle dependencies, among other properties that can be approximated by static analysis.

Existing formal approaches, including Petri nets (Araújo and Roque 2009) and finite state machines (Kebritchi, Hirumi, and Bai 2010), partly address player behavior and game dynamics, but these strategies often bypass the layered dependencies, spatial configurations and concurrent puzzle structures specific to escape games. As noted in (Araújo and Roque 2009), traditional modeling languages tend to struggle when verifying and validating complex interactive systems.

To address this gap, we introduce a graph-theoretical framework modeling escape games as dynamic, interactive systems. By representing them as directed graphs enriched with semantic constraints, we capture both the static configuration (rooms, puzzles, clues) and the evolving game state (player knowledge, progression). This enables automated reasoning tasks such as solvability checking, dependency validation, and balance assessment. We also present `GraphEG`, an open-source tool that implements our formalism for the creation, visualization, and simulation of escape games.

GRAPHS AND TREES FOR MODELING KEY ESCAPE GAME MECHANICS

An Escape Game (EG) can be seen as a finite, rule-driven environment where a group of players collaborate to achieve a goal (often, "escape") by navigating through interconnected rooms, discovering clues, and solving interdependent puzzles. Formally, we define an EG as a triple $(\mathcal{G}, \mathcal{A}, \delta)$, where $\mathcal{G}$ refers to a `Static Graph`, $\mathcal{A}$ a set of potential player actions, and δ a so-called transition function that specifies how each action impacts the game state. These concepts will be introduced in greater detail in the remainder of this section.

Blueprinting with the `Static Graph`

We start by defining a `Static Graph` (`SG`) as a directed graph $\mathcal{G} = (V, E)$ where:

- $V = R \cup Z \cup L \cup C \cup S \cup M$ is a finite set of *typed vertices*, in which R are *room vertices*, representing spatial areas in the game; Z are *puzzle vertices*, encoding the challenges to be solved; C contains *clue vertices*, being bits of information helping to solve puzzles; L holds the *role vertices*, modeling roles in puzzle solving or starting positions; S are *skill vertices*, representing cognitive and physical abilities needed to solve puzzles (e.g. "trigonometry basics"); and M represents the so-called *meta-information vertices*, which govern conditional branches, timers, game states, and win/loss conditions. We suppose the existence of at least one *victory* meta-information vertex in the set M.
- $E \subseteq V \times V$ is the set of directed edges representing logical dependencies and game mechanics between elements. The semantics of an edge is derived from the types of its origin and destination vertices, and some directed relations are prohibited, e.g. for r_1 and r_2 rooms and s_1 a skill, the edge $r_1 \to r_2$ represents a one-way path between the rooms, while $r_1 \to s_1$ is not allowed.

Each vertex type plays a specific role in the game logic. A puzzle $z \in Z$ *requires* at least one player to perform an action. This player is represented by a role $l \in L$ which may *require* a set of clues $\{c_1, \ldots, c_n\} \subseteq C$ and skills $\{s_1, \ldots, s_k\} \subseteq S$, and is *located* in a room $r \in R$. Solving a puzzle may *unlock* new room(s), *offer* new clue(s) or *unveil* new puzzle(s).

Now, the exact conditions on V and E that ensure that the `SG` is (semantically speaking) representing a valid EG are for now solely implemented in the companion proof-of-concept software (called `GraphEG`), and their formal study is left for future work. We refer the reader to `GraphEG`'s documentation in regard with what is currently permitted (or not) in its underlying `SG` formalization, as well as more details regarding the rules that govern the direction of the edges.
Also note that the model can be enriched with new vertex types if this is required to comply with the specifics of the considered EG. This aspect is left for future work; `GraphEG` should be seen as a first instantiation of the framework that aims to incorporate the most *usual* mechanisms of escape games.

An example `SG` is given in graphical form in Figure 1. Each different vertex type has received a different coloring and shape to ease the reader's comfort. For any given i, a **role** vertex of the form Li represents a starting position while Ri stands for a **room**. In the figure's graph, $L0$ is the standard starting position (to room $R1$) and $L1$ (to room $R2$) corresponds to a specific starting position for one of the players (who are thus separated at the start of the game). The player beginning his investigation in $R2$ is supposed to solve the **puzzle** $Z2$ (by taking role $L2.1$) to obtain access to the **clue** $C1.1$. This, in turn, allows him to take on the role $L1.1$, required to solve the puzzle $Z1$, finally allowing this player to open the door and join the rest of the team in $R1$ and $R3$, two rooms with free circulation (represented by double arrow edges in the graph). Regarding the puzzle $Z3$, it is implied in the graph that two roles ($L3.1$ and $L3.2$) should be endorsed simultaneously in order to solve it. This might correspond to a situation where two players must communicate from inside the room $R3$ to, e.g., activate switches simultaneously. As for Si nodes, they correspond to **skills**; in the example, $S2$ is required to assume the role $L1.1$ (and therefore the starting position $L1$). Finally, the meta-informative nodes $I0$ (*exit*) and $I1$ (*victory*) respectively remove players from the game when they enter $R5$ and declare that the game is successfully finished when no player is left.

As we can see, the `SG` component of an escape game essentially constitutes its blueprint, conceptualizing a plan of how the players can solve it in a step-by-step and chronological manner. Thanks to this representation, automated static analyses can be performed. Examples include precomputing the minimal number of different players required to solve the game (which amounts to 2 in the example above) as well as the detection of unreachable artifacts, dependency loops, and design inconsistencies; these checks are partially implemented in the application that will be introduced later on in the paper.

Temporal Modeling with the `Dynamic Graph`

We now define the set of *game states* $\mathcal{S}$ as being composed of tuples in the form (r, y, k, f), where r is a mapping from each player p to their current room, y and k are respectively the so-called *inventory* and *knowledge* mappings, associating clues and/or skills to each player, and f is a vector containing the current states of puzzles (e.g., solved/unsolved), clues (e.g. available/received/used) and rooms (e.g. closed/open/entered). Similarly, permissible *actions*, forming the set $\mathcal{A}$ that also parametrizes an EG, include moving between (accessible and open) rooms, interacting with a clue, a puzzle or a room, acquiring a clue, attempting to solve a puzzle and sharing some clue with another player.

We can finally define the third component of our definition of an EG, being the transition function $\delta : \mathcal{A} \times V \times \mathcal{S} \mapsto \mathcal{S}$ such that $\delta(a, v, s)$ represents the state into which one arrives when performing the

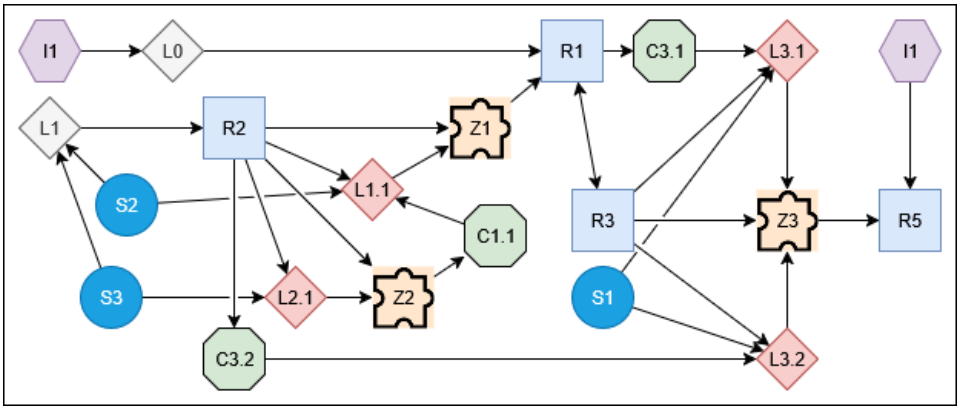

Figure 1: An example `Static Graph` (`SG`).

action a on artifact v when being in state s. The set of elements upon which an action can be carried out is the set V, the same set that represents the vertices of the Static Graph. We can then consider the existence, at any moment in a game, of the (current) game trace $D = \langle s_0, a_0, v_0, \cdots, s_{n-1}, a_{n-1}, v_{n-1}, s_n \rangle$, where $\forall i \in 1..n$: $s_i = \delta(a_{i-1}, s_{i-1}, v_{i-1})$.

For an EG taken as granted, its `Dynamic Graph (DG)` is then defined as the directed graph $\mathcal{G}_D = (V_D, E_D)$, where the set of vertices V_D is composed of those elements v from V upon which $\delta(s, a, v)$ can be applied for some $a \in \mathcal{A}$ and in the current state s (being retrieved as the last element of the trace D); $E_D \subseteq V_D \times V_D$ are directed and labeled edges, so that $(v_1, v_2)_a \in E_D$ represents the fact that a player (v_1) can perform an action $a \in \mathcal{A}$ on the artifact v_2, or that being in a room v_1 allows to perform the action a on v_2 (which is typically located in the room); D represents the current game trace, which operationally serves to update the graph by refreshing the available actions and artifacts according to the game's (past and present) states.

The `DG` can essentially be understood as a snapshot of the available actions in the current state (and taking the previous states into account), with vertices representing the parts of the game that can be interacted with by the involved players. In other words, the `DG` can be understood as its static counterpart put in motion (by making explicit how the roles endorsed by players, as well as their actions, affect the game state). This representation can for instance allow one to easily monitor a given game, detect difficulty issues in certain situations, perform replay analysis, and carry out a potential evaluation of decision paths and exploration diversity.

Covering all Paths with the `Game Session Forest`

While a `SG` allows one to statically approximate some interesting properties regarding a given escape game, it cannot cover some situations that occur in actual runs of the games, nor can it give definite answers to semantic (and, hence, undecidable) decision problems. On the other hand, `DG` covers some ground of situations that occur in real sessions, but it remains limited to representing one step of a single execution trace at the time. However, in order to induce some properties that hold at the level of the whole game, we are also interested in capturing *all* of the potential player progressions. To do this, we define the `Game Session Forest (GSF)` as a set of trees $\mathcal{F}$. Each tree $T \in \mathcal{F}$ corresponds to the tree representation of a game trace under a different decision policy. Without delving into the operational details of the `GSF` used internally in `GraphEG`, we will merely stretch out the fact that the `GSF`'s size can be kept reasonable in practice using adequate pruning techniques. For example, in Figure 1, the only game traces that should be considered are these traces that attribute both skills $S3$ and $S2$ to the player(s) entering $R2$ through $L1$, since all other traces cannot lead to the *victory* meta-informative vertex.

The `GSF` is designed to allow analyses to identify solvable paths (i.e., the existence of a trace ending in a goal state corresponding to reaching the *victory* vertex), but also to search for the most efficient such trace (according to some optimization criterion) and to perform redundancy and deadlock analysis. Additionally, it can be used to derive metrics regarding a game's average length or cognitive load, and to ensure e.g. that knowledge and item distribution remains balanced across players.

THE `GRAPHEG` VISUALIZATION TOOL

`GraphEG` is an open source interactive tool written in Python that allows game designers to build, visualize and simulate escape games based on the formalisms developed above[1]. At the core of `GraphEG` lies a declar-

[1]See the companion artifact at `https://anonymous.4open.science/r/GraphEG`.

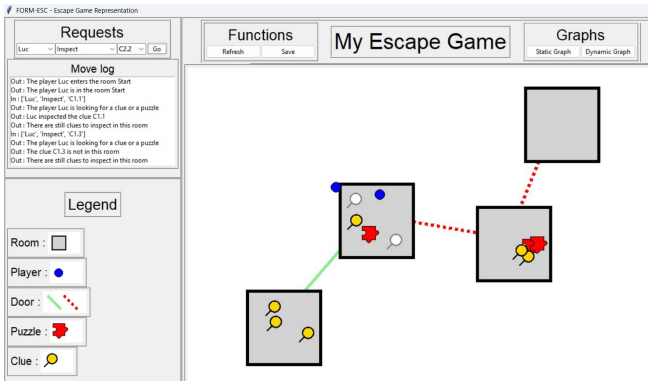

Figure 2: A `GraphEG` top-down layout with user controls.

ative JSON specification that describes a game's structure (called a *scenario*). The software parses this file and constructs graphs that are both sound in regard to the previous section's formalism, and that display the given scenario with as few edge intersections as possible. The `GraphEG` engine then performs a range of verifications and alerts the user of any of the tested properties that the model is or is not achieving. The software supports saving game states, still in a JSON file. To ease navigation, `GraphEG` offers two main visualization modes:

1. a *top-down layout*, being an interactive interface where users can observe and manipulate (by clicking) the spatial distribution of rooms, players, and interactive elements (a snapshot of an example top-down view is shown in Figure 2);

2. *graph visualizations* of the `SG` and the (current) `DG`, as well as an additional view, called a `Puzzle Dependency Graph (PDG)`, associated to each puzzle $p_j \in P$, and accessible by clicking on a puzzle's icon in the top-down view. This structure displays dependencies between puzzles, clues and skills in a more detailed (and user-friendly) way as in the `SG`. It can be leveraged by the program to perform puzzle solvability checking, dependency visualization, and assessment of difficulty layering, without the burden of the other information encoded in the `SG` and the `DG`. Figure 3 shows an example of each of these graphs (`SG`, `PDG` and `DG`) side by side.

Users can interact with the top-down layout through an action query interface composed of three selectors: *player*, *action*, and *target item*. Supported actions include *interact*, *inspect*, *take*, *resolve*, *share*, *move*, and *exit*. Each action is checked against the current game state, and validated inputs dynamically update both the graphical interface and the underlying JSON file, effectively implementing the function δ from the previous section. More specifically, if the user wishes to visualize the `Dynamic Graph`, the graph in question will reflect the effect of the last action taken by the team of players, as such depicting the current game state as well as the set of actions that can be performed in the state in question.

The interface also features a pop-up history displaying (timestamped) past queries and system outputs. This helps the user understand the logical consequences of actions and detect invalid moves or missing information. It is intended to be used by game masters who monitor an actual escape game in real time: keeping the visual representations up to date with the actions of the players helps to keep a trace of different game sessions. This can be useful to compare the paths taken by different teams and to identify those parts of the game that may need simplification or, the other way round, an increase in difficulty. Future developments will focus on alleviating this manual task. More specifically, the `GraphEG` system could connect to actual, IoT-based escape rooms, so that (part of) the visualization and its inner representations automatically capture and represent the progression of the players, without manual intervention.

CONCLUSIONS AND PERSPECTIVES

As escape games expand across platforms and disciplines, we believe our framework provides a robust theoretical foundation and a practical toolkit to support their design and analysis. Indeed, by unifying design-time architecture and run-time evolution in a layered representation, `GraphEG` already enables automated reasoning over game structure and gameplay balance.

Beyond recreational contexts, the framework opens avenues for serious game modeling and Artificial Intelligence (AI) techniques such as plan recognition and advanced automated reasoning (Ghallab, Nau, and Traverso 2004). In human-computer interaction, our graphs could help develop adaptive interfaces for immersive applications (Romero and al. 2021; Veldkamp et al. 2022). From a practical perspective, complex scenarios may benefit from a higher-level scripting layer, potentially using Large Language Models to generate valid `GraphEG` scenarios from textual descriptions.

However, several limitations remain: the current model abstracts away player cognition (reasoning strategies, coordination, intuition), only partially represents temporal dynamics (lacking continuous constraints and real-time decay), and does not yet capture narrative evolution such as branching dialogues (Riedl and Young 2006). Large-scale validation of `GraphEG`'s performance and impact on gameplay is also pending; for this, we plan to rely on structured evaluation methodologies such as the one described in (Kabimbi Ngoy, Yernaux, and Vanhoof 2023). Additionally, to further tackle these limitations, we aim to develop a new, updated version of the `GraphEG` tool that takes some of the above-mentioned features into account. Further future work will then in-

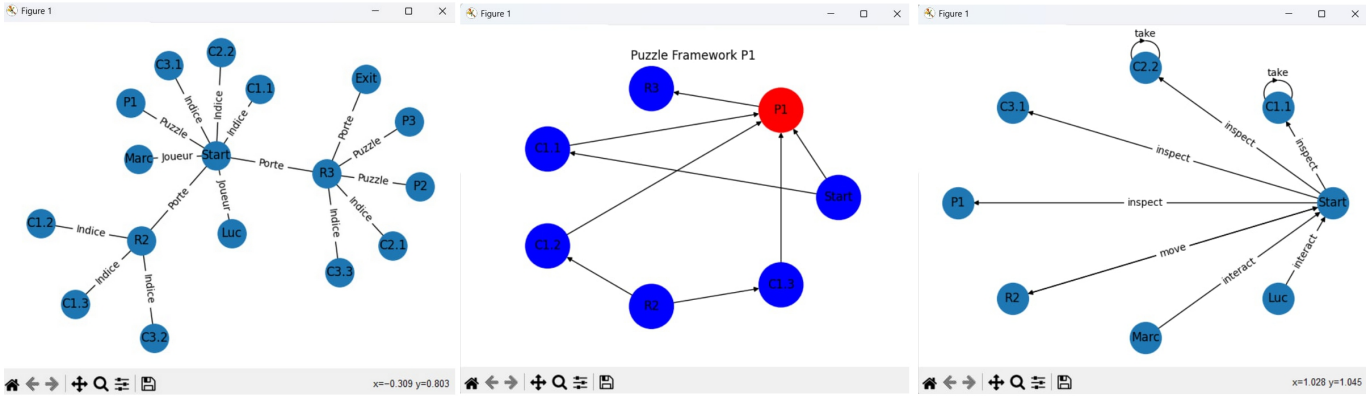

Figure 3: An example `SG` (left), `PDG` (middle) and `DG` (right), as displayed in `GraphEG`.

clude empirical evaluation of games designed with our software, integration with automated solvers and AI assistants, and the definition of metrics for graph complexity, educational efficacy, and player engagement.

References

Araújo, Manuel and Licínio Roque (2009). "Modeling Games with Petri Nets". In: *Digital Games Research Association Conference*. DOI: 10.26503/dl.v2009i1.493.

Botturi, Luca and Masiar Babazadeh (2020). "Designing educational escape rooms: validating the Star Model". In: *International Journal of Serious Games* 7, pp. 41–57. DOI: 10.17083/ijsg.v7i3.367.

Clarke, Samantha and David Peel (2017). "EscapED: A framework for creating educational escape room games". In: *European Conference on Games Based Learning*, pp. 111–118. DOI: 10.17083/ijsg.v4i3.180.

Ghallab, Malik, Dana Nau, and Paolo Traverso (2004). *Automated Planning: Theory and Practice*. Elsevier. DOI: 10.1016/B978-1-55860-856-6.X5000-5.

Kabimbi Ngoy, Rudy, Gonzague Yernaux, and Wim Vanhoof (2023). "EvscApp: Evaluating the Pedagogical Relevance of Educational Escape Games for Computer Science". In: *International Conference on Computer Supported Education*. SciTePress, pp. 241–251. DOI: 10.5220/0011715100003470.

Kebritchi, Mansureh, Atsusi Hirumi, and Haiyan Bai (2010). "What do we know about computer games and learning?" In: *British Journal of Educational Tech.* 41.1, pp. 18–38. DOI: 10.1111/j.1467-8535.2009.00997.x.

Nicholson, Scott (2015). *Peeking behind the locked door: A survey of escape room facilities*. White paper.

Riedl, M.O. and R.M. Young (2006). "From linear story generation to branching story graphs". In: *IEEE Computer Graphics and Applications* 26.3, pp. 23–31. DOI: 10.1109/MCG.2006.56.

Romero, Carla and et al. (2021). "Escape rooms as innovative pedagogical tools for active learning: A systematic review". In: *Educational Research Review* 34, p. 100404. DOI: 10.1016/j.edurev.2021.100404.

Taraldsen, Lene Hayden et al. (2022). "A review on use of escape rooms in education – touching the void". In: *Education Inquiry* 13.2, pp. 169–184. DOI: 10.1080/20004508.2020.1860284.

Veldkamp, Alice et al. (2022). "You escaped! How did you learn during gameplay?" In: *British Journal of Educational Tech.* 53. DOI: 10.1111/bjet.13194.

Wiemker, Markus, Errol Elumir, and Adam Clare (2015). "Escape Room Games: Can you transform an unpleasant situation into a pleasant one?" In: Haag, J., Weißenböck, J., Gruber, W. & Freisleben-Teuscher, C. F. (Eds.), IKON (Publisher): Game Based Learning, pp. 55–68. ISBN: 978-3-99023-411-2.

AUTHOR BIOGRAPHIES

GONZAGUE YERNAUX was born in Etterbeek, Belgium and went to the University of Namur, where he studied computer science, specializing in algorithms, declarative languages and program analysis. He obtained his PhD in 2025. Gonzague is also a fiction writer, and the designer of multiple home-made escape rooms and role-playing game scenarios.

MARTIN VERJANS is a Scada Integration Engineer at Luminus, a major player in the Belgian energy landscape. He completed a Master's degree in Computer Science at the University of Namur. His Master's thesis aimed at formalising escape games as graphs and laid the groundwork for the present paper.

WIM VANHOOF is an Ordinate Professor specialized in programming languages and algorithmics at the Faculty of Computer Science of the University of Namur. He has been the Dean of the Faculty for six consecutive years (2019-2025).