Formalizing Escape Game Mechanics: A Graph-Theoretical Framework for Modeling and Analyzing Puzzle-Based Environments

Gonzague Yernaux, Martin Verjans, and Wim Vanhoof

Faculty of Computer Science, University of Namur, Belgium

Abstract. Escape games have become widely popular across entertainment, educational, and training domains, yet their underlying mechanics remain largely informal and under-theorized. In this work, we introduce a novel framework that provides a formal representation of both the structural and dynamic aspects of escape games. Our approach relies on the Static Graph, a directed graph that encodes the topological and logical organization of puzzles, clues, rooms, and player roles. Game progression and player interaction are modeled through the Dynamic Graph, capturing the live state of a session, as well as the Game Session Forest, which represents the set of possible traces under alternative player choices. This graph-based design can easily be manipulated by verification algorithms, and as such paves the way for automated reasoning over essential aspects of escape games, ranging from solvability and balance to determining the initial constraints on players and rooms. The framework is operationalized and illustrated in `GraphEG`, an open-source visualization tool that supports both the design of escape game scenarios and their simulation.

1 Introduction and Related Work

Escape games, also known as escape rooms, have rapidly transitioned from a niche form of entertainment to a widespread cultural and educational phenomenon. Since their emergence in the early 2000s in Japan and their international rise through commercial adaptations like *Real Escape Game* and *Escape the Room*, they have been adopted for various purposes, including team-building exercises, cognitive training, and formal education [14].

Their popularity stems from a compelling mix of immersive storytelling, collaborative problem-solving, and time-bound challenge dynamics. In pedagogical contexts, escape games have been shown to increase engagement, foster critical thinking, and improve knowledge retention through experiential learning [4,15]. Frameworks such as *EscapED* [2] and the *Star Model* [3] have offered conceptual and narrative guidelines for game-based learning.

The escape game genre has in fact been investigated through various disciplinary lenses, ranging from educational science to human-computer interaction and game design theory. Nicholson's foundational works [11] provided an early taxonomy of escape room principles such as narrative framing, collaborative

puzzle-solving, and time-limited challenges. Wiemker et al. [17] also proposed classifications of puzzles and structural ideas for optimal player engagement.

Then, as the efficiency of escape games in active learning has been increasingly demonstrated in independent studies [14,5], several works proposed methodological guidelines for integrating educational objectives within game mechanics at the conception stage [3]. Meanwhile, in [13] and independently in [15], some authors insist on another crucial need, being the need to model the behavior of players during actual sessions of serious games. The existing approaches set on tackling this aspect relied on well-known mathematical objects such as Petri nets [1] and finite state machines [10].

However, such models tend to remain qualitative and lack expressivity when dealing with the layered dependencies, spatial configurations, and concurrent puzzle structures that characterize escape games. In that sense, Araújo and Roque caution that traditional modeling languages are limited when it comes to verifying and validating underlying game systems [1]; they then advocate for formalisms capable of handling more complex, concurrent interactions. In parallel, recent research in *serious games* (being games dedicated to serious outcomes such as learning) also tends to identify the need for formal models that support game design, since these can be used to enable simulation, complexity analysis, and Artificial Intelligence (AI)-driven design [13].

Attempts to formalize serious games in a more general way did emerge in the context of learning analytics and simulation [3]. But these apply poorly to escape games, which still suffer a lack of formal and computable representation. This limits their reproducibility, adaptability, and analytical potential. The internal mechanics of escape games thus remain to this day largely informal or heuristic.

In this context, this paper proposes a novel graph-theoretical framework for modeling escape games as dynamic, interactive systems; an endeavor that has, to the best of our knowledge, not been performed before, except in our own seminal work on the topic [18]. By abstracting escape games as directed graphs enriched with semantic constraints, we enable algorithmic reasoning over the structure and flow of gameplay. Our approach captures both the static configuration of an escape game (e.g., rooms, puzzles, clues) as well as the permanently evolving game state (e.g., player knowledge, progression or clues acquisition). As such, it allows for a range of formal validation checks that can shed light on the inner properties held by each of the modeled games.

To demonstrate the practical viability of our framework, we also introduce the latest version of a companion software tool called `GraphEG` that supports the creation, visualization, verification and simulation of escape games using the proposed formalism.

2 A Formal Framework for Modeling Escape Games

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, solving interdependent puzzles, and discovering

clues. More formally, we define an EG as a triple $(\mathcal{G}, \mathcal{A}, \delta)$. Specifically, $\mathcal{G}$ refers to a Static Graph conforming with what we will further define in Section 2.1, $\mathcal{A}$ is a set of potential player actions, and δ an associated transition function that details which actions are possible in which game situations. The latter two concepts, $\mathcal{A}$ and δ, will be introduced in greater details in Section 2.2.

2.1 Blueprinting with the Static Graph (SG)

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*:
  - R: *room vertices*, representing spatial areas in the game;
  - Z: *puzzle vertices*, encoding the challenges to be solved;
  - C: *clue vertices*, items and bits of information helping to solve puzzles;
  - L: *role vertices*, modeling roles in puzzle solving or starting positions;
  - S: *skill vertices*, representing cognitive and physical abilities needed to solve puzzles (e.g. "trigonometry basics");
  - M: *meta-information vertices*, governing 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, depending on the exact setting of each escape game, new vertex types could be incorporated in the model. This is left for future work; GraphEG should be seen as a first instantiation of the framework, and as such incorporates a collection of *usual* inherent mechanisms of escape games.

An example SG is given in graphical form in Fig. 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**. $L0$ is the standard starting position (to room $R1$) and $L1$ (to room $R2$) corresponds to a specific starting position for one of the players (so

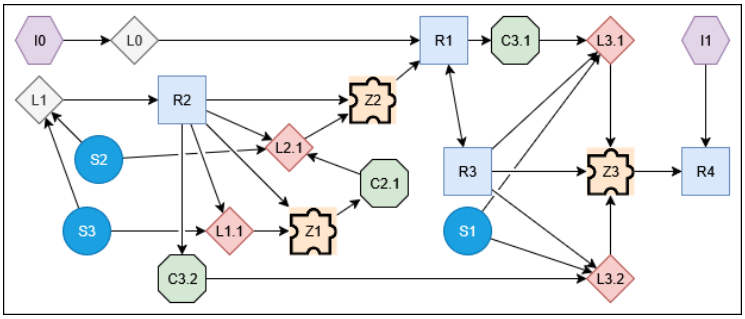

Fig. 1. An example Static Graph (`SG`).

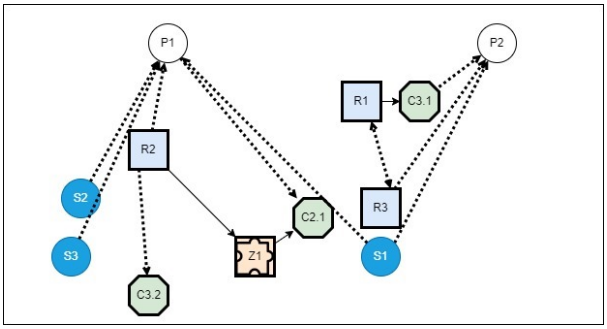

Fig. 2. A Dynamic Graph (`DG`) based on the Static Graph (`SG`) from Fig. 1

they are separated at game start). The player starting in $R2$ is supposed to solve the **puzzle** $Z1$ (by taking role $L1.1$) to obtain access to the **clue** $C2.1$. This, in turn, allows him to take on the role $L2.1$, required to solve the puzzle $Z2$, finally allowing this player to open the door and join the rest of the team in $R1$ and $R3$, two rooms with free circulation. Let us now consider 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. Si nodes correspond to **skills**; in the example, $S2$ is required to assume the role $L2.1$ (and therefore the starting position $L1$). Finally, the meta-informative nodes $I0$ (*victory*) and $I1$ (*exit*) respectively declare victory when no player is left and remove players from the game when they enter $R4$.

As we can see, the `SG` component of an escape game essentially constitutes its blueprint, conceptualizing a plan of how players can win in a step-by-step and chronological way. Thanks to this representation, automated static analyses can be performed. Examples include pre-computing 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.

2.2 In-game Modeling with the Dynamic Graph (DG)

To play the game, we add a set of players P. We can now define the Dynamic Graph (DG), given a Static Graph (SG) $\mathcal{G} = (V, E)$, as a directed graph $\mathcal{G}_d = (V_d, E_d)$, in which V_d is the vertex set V from SG to which we add *player vertices*; each vertex is then decorated with a *discovered* (boolean) attribute. $E_d \subseteq V_d \times V_d$ is the set of directed edges representing the players' (potential) progress.

Fig. 2 illustrates an in-game snapshot based on the Static Graph from before. The nodes that have been *discovered* have a bold border. Player $P1$ is currently *located* in room $R2$ and *possesses* skills $S1$, $S2$ and $S3$. He *carries* clue $C2.1$. Clue $C3.2$ has been *discovered* but not *picked up*[1]. Puzzle $Z1$ has been *solved* (since it is *discovered* and no edge from $\mathcal{G}_d$ is pointing towards it anymore). Player $P2$ is *located* in room $R3$. He *carries* clue $C3.1$ and *possesses* skill $S1$. $Z2$ and $Z3$ have not been discovered yet; hence their absence from the DG.

From a given DG $\mathcal{G}_d$, it is possible to compute an action set $\mathcal{A}(\mathcal{G}_d)$ representing the possibilities that each player has in the situation described by $\mathcal{G}_d$); this particular computation is included in the GraphEG tool introduced in the next section; it allows to systematically generate the DG and to assess the validity of potential player actions. Resolving an action from this set would then update the graph according to the possibly newly discovered elements. For example, DG from Fig. 2 would generate the action set $\{Explore(P1)$, $Pickup(P1, C3.2)$, $Move(P2, R1)$, $Explore(P2)\}$. The permissible actions also include actions of the form $Memorize(Px, Cy)$, $Enter(Px)$, $Drop(Px, Cy)$ and $Attempt(Px, Zy)$.

Now, operationally, one would typically need to *traverse* the (static and/or dynamic) graphs in order to perform formal property checks and verification. In this case, traversing the graph implies following dependencies whose semantics depend on the types of the connected vertices. More formally, a traversal step along an edge $(u, v) \in E \cup E_d$ is permitted if and only if some semantic conditions are satisfied with respect to the node types. For example, if $u \in Z$, then u must be solved for it to lead to some new element v in the DG; if $v \in R$ and the player is currently in room u, then there must exist an edge $(u \rightarrow v) \in E$. In the DG, traversing to a node v additionally triggers its activation (e.g., revealing all visible elements inside a room). Hence, a traversal of the graph corresponds to (the verification of) a sequence of valid state transitions, as defined by the third component of our definition of an EG.

The component in question is a transition function $\delta : \mathcal{A} \times \mathcal{G}_d \mapsto \mathcal{G}_d$, defining a sequence $\{\mathcal{G}_d^i\}_i \in 1..n$ such that $\mathcal{G}_d^{i+1} = \delta(a, \mathcal{G}_d^i)$ represents the DG obtained upon resolving the effect of a on $\mathcal{G}_d^i$. We can then consider the existence, at any moment in a game, of a succession of (past and present) DGs and actions, called the Game Session, and defined as $\langle G_d^0, a^0, \cdots, G_d^{n-1}, a^{n-1}, G_d^n \rangle$, where $\forall i \in 1..n$: $G_d^i = \delta(G_d^{i-1}, a^{i-1}) \wedge a^{i-1} \in \mathcal{A}(\mathcal{G}_d^{i-1})$.

[1] We refer the reader to GraphEG's documentation for more information on this regard.

```
C:\Escape Game\Example>git log --oneline
08f63e6 (HEAD -> Example_2_GS, tag: Victory_InitialSession-2) Player P2 moved from Room R2 to Room R3
34c2490 Player P1 moved from Room R2 to Room R3
2cabbbe Players [P1, P2] solved Puzzle Z3 and found Room R3 as reward.
49ccf06 Player P1 moved from Room R0 to Room R2
30d4f66 Player P2 found a puzzle Z3.
76305ef Player P1 moved from Room R1 to Room R0
37cb9fe Player P1 memorized Clue C.3.2
7c1dcb1 Player P1 found a clue C.3.2.
f69a635 Players [P1] solved Puzzle Z2 and found Room R0 as reward.
9ba7ac0 Player P1 found a puzzle Z2.
732f255 Player P2 found a door to room R0.
1bead6a Player P2 moved from Room R0 to Room R2
ce313d1 Player P1 memorized Clue C.2.1
1ddeb94 Players [P1] solved Puzzle Z1 and found Clue C.2.1 as reward.
5885761 Player P1 found a puzzle Z1.
502c909 Player P2 found nothing.
03e914d Player P2 memorized Clue C.3.1
a7e99d0 Player P2 found a clue C.3.1.
ee0a344 Player P2 found a door to room R2.
a51e97d Player P2 started the game in room R0
162d3c4 Player P1 started the game in room R1
3587f2c P1:S1-S2-S3_P2:S1
bec2b0f (InitialSession) Game initialized
```

Fig. 3. A full Game Session visualization using Git native commands

2.3 Temporal Modeling with the Game Session Forest (GSF)

While a SG allows one to statically approximate some interesting properties regarding a given escape game, it cannot cover what occurs in practical runs of the games. A DG represents an in-game situation, but is only a snapshot of a practical game execution at a given point in time. Then, a Game Session is essentially the linkage of subsequent DG that unlocks the monitoring of the actions executed by the players from the start of the game until its resolution.

Now, to induce some properties that will or will not hold with respect to an EG, we are 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 session (essentially representing alternative decision policies). In practice, the GSF's size can be kept reasonable using ad hoc pruning techniques. As an illustration, in Fig. 1, one should only consider those game traces that attribute both skills $S2$ and $S3$ to the player entering $R2$ through $L1$, since all other traces would make the game impossible to win.

The GSF is mostly designed to allow subsequent analyses to identify solvable paths (i.e., the existence of a trace ending in a goal state corresponding to reaching the *victory* condition), 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 remain balanced across players.

3 An Overview of the GraphEG Visualization Tool

GraphEG is an open source interactive tool written in C#[2] destined to be used by game designers for building, visualizing and simulating escape games based

[2] See the companion artifact at `https://github.com/martin-verjans/GraphEG`.

on the formalisms developed above. A first prototype version of the tool was introduced in [18]; however, it lacked several key features. In this section, we describe an updated version able to handle more situations and visualizations.

The `GraphEG` software relies on the `QuikGraph` and `GraphViz` libraries to render `SG`s and `DG`s in the `DOT` graph description language [6]. To represent a Game Session Forest, the software leverages Git's native operations. Game sessions are represented by a branch in the repository; each commit then represents a `DG`, so that tracing a session is equivalent to requesting the log for the given session or branch. Then, the `GSF` can simply be retrieved as the full Git repository, including all of its branches. Note that this allows us to take advantage of Git's inner representation of repositories as being, essentially, trees, as well as its capabilities for comparing commits and pruning repository branches automatically. Fig. 3 shows a command-line visualization of the whole `GSF` based on this. Next to the `GSF` integration, `GraphEG` offers two main user interfaces:

1. **The Designer** is the name given to the workspace that allows users to create and edit a `SG` by adding and removing nodes and edges. Fig. 4 shows a screenshot of the Designer in which we recreated the example `SG` from Fig. 1. The Designer also provides basic formalism verification by preventing the creation of forbidden edges (e.g., connecting a room to a skill) and validating vertex consistency (e.g., a puzzle must be connected to at least one role). It is additionally possible to save the graph or to load one from a file.
2. **The Gameplay** is a second interface that first requests the user to pick an existing `GSF` (by selecting a Git repository) or to create a new one from a given `SG` file. Then, it will display the existing Game Sessions from the selected `GSF` or propose the creation of a new one. Finally, the user is allowed to simulate the game. `GraphEG` then displays the current game state (embodied by a `DG`) and details its corresponding action set. Upon selecting one of the available actions, the user triggers an update in the `DG`; in that case, a new Git commit is also created on the fly. A screenshot of the Gameplay view is given in Fig. 5; it corresponds to the example `DG` given earlier in Fig. 2.

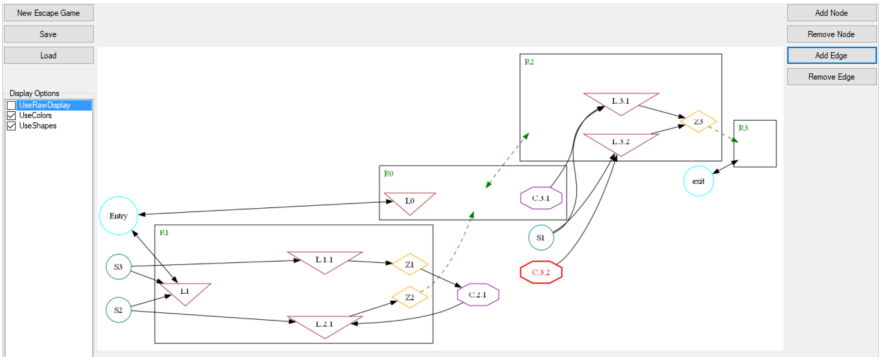

Fig. 4. A snapshot of the Designer interface corresponding to the `SG` from Fig. 1

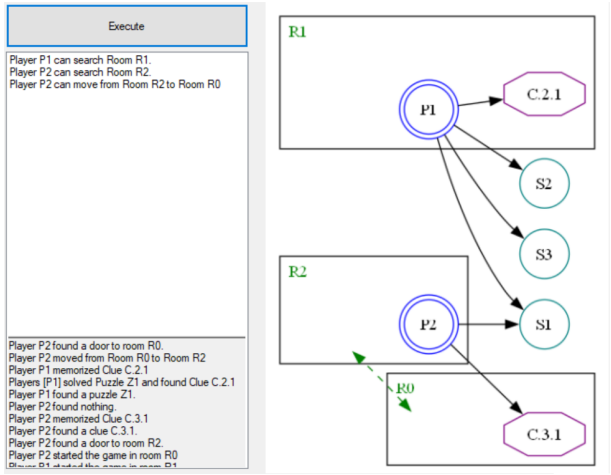

Fig. 5. A snapshot of the Gameplay interface corresponding to the `DG` from Fig. 2

In addition to these interfaces, `GraphEG` offers visualization features that facilitate the understanding of complex dependencies. For instance, different vertex types are distinguished by color and shape, and consistency checks are reported directly in the interface. Moreover, the possibility to display Static and Dynamic Graphs side by side helps designers follow the evolution of a session while preserving a clear overview of the underlying blueprint. Such visualization support is crucial to make the traversal semantics and player progress more transparent, especially for non-technical users such as educators and practitioners.

4 Discussion

This paper introduced a formal graph-theoretical framework for modeling, simulating, and analyzing escape games, treating them as structured, state-driven systems governed by interdependent puzzles, spatial progression, and player agency. Our layered representations offer a unified vocabulary for capturing both the design-time architecture and run-time unfolding of such experiences.

The model also enables automatic reasoning tasks such as solvability checking, dependency validation and balance assessment. This formalism was further operationalized through the development of a dedicated open-source application that allows designers to visualize, test, and iterate on game blueprints.

Despite its expressiveness and versatility, the proposed framework presents several limitations that merit discussion. First, while offering a robust formalization of structural and interactive components, it abstracts away the cognitive dimension of players. Aspects such as reasoning strategies, group coordination, intuition, and trial-and-error learning are not explicitly modeled, though they are often central to real-world escape room experiences. In the same vein, our model does not yet incorporate mechanisms for modeling evolving stories,

character-driven events and branching dialogue systems. This could be tackled by incorporating e.g. logic-based storytelling structures or narrative graphs [12].

Secondly, the framework captures only discrete event sequencing, thus lacking the richness of continuous temporal modeling such as duration-based constraints, real-time pressure, and decay functions over time to represent more complex scenarios. From a practical standpoint, the creation of such scenarios involving dozens of interconnected (and timed) puzzles can become unwieldy without the aid of a higher-level domain-specific scripting interface. Incorporating such a feature in the `GraphEG` user interface, e.g. by leveraging Large Language Models (LLMs) able to generate adequate JSON files from a textual description of the game is an interesting avenue for further work.

Finally, while the theoretical underpinnings of the model are sound, its empirical validation is still pending. We plan to tackle this by including experimental validation and benchmarking, regarding both `GraphEG`'s runtime performances and effectiveness in real-life scenarios. To achieve such measures, one can, for example, draw upon existing structured evaluation frameworks tailored for (educational) escape games such as the methodologies developed in [9] and [16].

5 Future Work

The limitations above in fact pave the way for interesting avenues of further work. As a first example, following the trend of escape games used to train critical thinking, collaboration, and active learning in educational and corporate environments [7,5], `GraphEG`'s core mechanics have been chosen so that serious games could easily benefit from our models.

Of direct use in that regard is the dynamic tracking of player paths via our various graphs, which can straightforwardly be leveraged to extract learning analytics. This can help instructors and designers to, e.g., identify conceptual bottlenecks in their storytelling, or to adapt the content of their courses or games. Such efforts to evaluate the relevance of pedagogical escape games have been studied before (see e.g. [15]) and should benefit from the structured data collected during a `GraphEG`-monitored session. Even more, the `GraphEG` application naturally supports the integration of pedagogical objectives directly into the structural blueprint of a game. This is handled by the software's exhaustive listing of prerequisites of a given action, clue or puzzle, and by the possibility of encoding differentiated pathways for learners with varying skill sets.

Apart from these education-driven applications, `GraphEG` opens promising intersections with other research fields. For example, thanks to its manipulation of game traces, our framework may support the modeling of agents operating in multi-goal and/or constrained environments arising in plan recognition and automated reasoning, two important fields in the AI landscape [8]. Human-computer interaction may similarly benefit from our user-based modeling, which paves the way for more adaptive interfaces, especially in immersive contexts such as those explored in [16]. This is especially relevant in extended reality (XR) applications, where spatial reasoning and clarity are critical [14].

Additional future work includes the integration with automated solvers and AI assistants, as well as the development of metrics to evaluate graph complexity, educational efficacy, and player engagement for `GraphEG`-modeled games.

References

1. Araújo, M., Roque, L.: Modeling games with Petri nets. In: Digital Games Research Association Conference (2009)
2. Borrego, C., Fernández, C., Blanes, S., Robles, S.: Designing escape rooms for the classroom. In: ACM conference on innovation and technology in computer science education. pp. 331–331 (2017)
3. Botturi, L., Babazadeh, M.: Designing educational escape rooms: validating the star model. International Journal of Serious Games **7**, 41–57 (2020)
4. Clarke, S., Peel, D.: EscapED: A framework for creating educational escape room games. In: European Conference on Games Based Learning. pp. 111–118 (2017)
5. Delmas, G., Champagnat, R., George, S.: Structuring learning activities in escape games: A preliminary typology. In: International Conference on Human-Computer Interaction. pp. 121–139. Springer (2020)
6. Ellson, J., Gansner, E.R., Koutsofios, E., North, S.C., Woodhull, G.: Graphviz - open source graph drawing tools. Lecture Notes in Computer Science **2265**, 483–484 (2002)
7. Fotaris, P., Mastoras, T.: Escape rooms for learning: A systematic review. In: European Conference on Games Based Learning (2019)
8. Ghallab, M., Nau, D., Traverso, P.: Automated Planning: Theory and Practice. Elsevier (2004)
9. Kabimbi Ngoy, R., Yernaux, G., Vanhoof, W.: EvscApp: Evaluating the pedagogical relevance of educational escape games for computer science. In: International Conference on Computer Supported Education. pp. 241–251. SciTePress (2023)
10. Kebritchi, M., Hirumi, A., Bai, H.: What do we know about computer games and learning? British Journal of Educational Technology **41**(1), 18–38 (2010)
11. Nicholson, S.: Peeking behind the locked door: A survey of escape room facilities (2015), white paper
12. Riedl, M., Young, R.: From linear story generation to branching story graphs. IEEE Computer Graphics and Applications **26**(3), 23–31 (2006)
13. Robison, B.: Games as models: Simulation, abstraction, and design. Simulation & Gaming **52**(5), 558–575 (2021)
14. Romero, C., et al.: Escape rooms as innovative pedagogical tools for active learning: A systematic review. Educational Research Review **34**, 100404 (2021)
15. Taraldsen, L.H., Haara, F.O., Lysne, M.S., Jensen, P.R., Jenssen, E.S.: A review on use of escape rooms in education – touching the void. Education Inquiry **13**(2), 169–184 (2022)
16. Veldkamp, A., Niese, J., Heuvelmans, M., Knippels, M.C., van Joolingen, W.: You escaped! How did you learn during gameplay? British Journal of Educational Technology **53** (2022)
17. Wiemker, M., Elumir, E., Clare, A.: Escape room games: Can you transform an unpleasant situation into a pleasant one? (2015)
18. Yernaux, G., Verjans, M., Vanhoof, W.: Towards a graph-theoretical framework for modeling and analyzing escape game mechanics. In: Geril, P. (ed.) 26th International Conference on Intelligent Games and Simulation. EUROSIS (2025)