2. Related Work / Literature Survey

This section reviews existing research related to game scripting, domain-specific languages, narrative control, engine support, and simplified game development approaches. Rather than treating prior studies independently, the discussion traces the evolution of ideas across the literature, highlighting how successive works attempt to address the limitations of earlier approaches and how these developments motivate the proposed GameScript (GS) system.

2.1 Rule-Based and Declarative Game Scripting

Early research in game scripting sought to reduce procedural complexity by representing game logic through rules and declarative constructs. Initial studies demonstrated that game mechanics and levels could be generated automatically from structured rule sets, showing that expressive gameplay does not require detailed procedural coding (Paper 1). This work established the feasibility of rule-based representations as a foundation for game logic.

As rule-based systems gained attention, researchers examined their expressive power in practical settings. Analysis of declarative rule-based languages revealed that compact rule formulations can effectively capture complex interactions and emergent behaviors within games (Paper 16). However, as these systems grew in size, concerns shifted toward script organization, efficiency, and maintainability. Subsequent studies emphasized that poorly structured scripting languages can negatively impact both development speed and runtime performance (Paper 5).

To improve clarity and long-term maintenance, later approaches introduced structured declarative models that explicitly separate game state from behavioral rules (Paper 18). While this separation enhanced readability and organization, existing solutions often lacked a unified, lightweight scripting approach that balances simplicity with expressive control. These observations motivate GS, which adopts a concise, rule-driven scripting model designed to remain readable and maintainable as game logic scales.

2.2 Domain-Specific Languages for Game Development

Building on the limitations of general-purpose scripting, researchers explored domain-specific languages (DSLs) as a means to raise the level of abstraction in game development. Early DSL approaches embedded game-specific constructs within host languages, enabling developers to describe behavior at multiple abstraction levels while relying on existing execution environments (Paper 10). This marked a shift toward separating game logic from low-level implementation concerns.

Subsequent work extended this idea by introducing extensible game description languages, allowing game rules and behaviors to be modified or expanded without changes to engine code (Paper 13). These languages improved flexibility but also highlighted the need for clear domain concepts to avoid overcomplication. As DSL research matured, attention turned to educational and serious games, where clarity and accessibility are especially important. DSLs designed for modeling adventure and educational games demonstrated that domain-focused constructs can effectively structure gameplay progression and interaction logic (Paper 14).

Further evolution toward language-driven development showed that defining game logic before engine construction improves modularity and reuse (Paper 15). However, many DSLs still impose steep learning curves or rigid syntax. GS addresses this gap by offering a minimal, rule-oriented DSL that preserves abstraction benefits while remaining easy to understand and modify.

2.3 Game Script Engines and Engine Architecture

As high-level scripting and DSLs became more prevalent, research focus expanded to how such scripts are executed efficiently within game engines. Early work addressed performance and memory limitations by proposing lightweight scripting engines tailored for constrained environments such as mobile platforms (Paper 2). These engines demonstrated that simplified execution models can significantly reduce resource overhead.

Parallel research examined engine architecture more broadly, emphasizing modular design and performance optimization to support increasingly complex game logic (Paper 9). While such architectures improved scalability, they also increased system complexity. This raised concerns about long-term maintenance, which were later confirmed by studies identifying code quality and sustainability issues in popular open-source game engines (Paper 11).

To mitigate these issues, object-oriented design patterns were applied to improve reusability and flexibility (Paper 28). However, this solution often shifts complexity from scripting to architecture, making systems harder to understand for non-expert designers. These developments suggest that simplifying the scripting layer itself, rather than relying on complex engine designs, is a more sustainable approach—one that GS explicitly targets.

2.4 Narrative, Dialogue, and Interactive Story Scripting

The evolution of game scripting also encompasses narrative-driven and interactive storytelling systems. Early research showed that structured scripts can manage branching narratives and dialogue flows, enabling content creators to design interactive stories without extensive programming expertise (Paper 4). These ideas were validated through practical implementations where scripts controlled dialogue sequences, puzzles, and story progression in adventure-style games (Paper 7).

As interest in narrative automation grew, researchers explored AI-based solutions. Recent studies applied large language models to generate interactive story sequences dynamically, reducing manual scripting effort (Paper 21). However, these approaches introduced challenges related to predictability, authorial control, and runtime performance.

Complementary research investigated the structural properties of game dialogues (Paper 22) and applied behavioral modeling techniques to manage non-linear narratives (Paper 25). Although these methods advance narrative flexibility, they often sacrifice transparency and simplicity. In contrast, GS emphasizes explicit, rule-based narrative scripting, offering designers greater control over story logic while avoiding the complexity of heavy AI-driven systems.

2.5 Educational Games and Gamification Systems

The need for simple and accessible scripting mechanisms is especially pronounced in educational and gamified applications. Early studies demonstrated that script-based knowledge representations can effectively organize educational game logic and learning activities (Paper 3). To support novice users, simplified scripting engines with limited instruction sets were later introduced, enabling students to experiment with game behavior without deep programming knowledge (Paper 6).

Building on this foundation, authoring systems were developed to allow educators to design gamified learning activities through high-level scripting interfaces (Paper 8). Subsequent research focused on collaborative learning, teamwork assessment, and inquiry-based games (Papers 24, 26, 27, and 30). While these studies prioritize pedagogical outcomes

over technical design, they consistently highlight the importance of clear and understandable representations of game logic, reinforcing the relevance of GS.

2.6 Model-Driven and Visual Game Design Approaches

In parallel with scripting-based solutions, researchers explored model-driven and visual abstractions to further reduce development effort. Domain-specific modeling languages and model-driven frameworks demonstrated that high-level models can be transformed into executable games, minimizing the need for manual coding (Papers 17 and 19). Graphical DSLs extended this idea by allowing designers to define game entities and interactions visually (Paper 20).

While these approaches effectively abstract low-level details, they often limit fine-grained control over runtime behavior and adaptability. This evolution highlights a trade-off between abstraction and flexibility, positioning GS as a middle-ground solution that maintains scripting expressiveness while remaining simple and readable.

2.7 Game Design and Conceptual Planning

Some research focuses on conceptual and early-stage design support rather than executable logic. Techniques such as game sketching enable rapid exploration of design ideas (Paper 23), while structured game design documents support consistency and thematic coherence (Paper 29). These contributions are valuable for planning but do not address the specification or execution of game behavior.

Finally, work on federated learning incentive mechanisms using blockchain and game theory (Paper 12) lies outside the scope of this study, as it does not contribute to game scripting, logic abstraction, or game development methodologies.