A Systematic Review of Model-Driven Game Development Studies

Amirreza Payandeh, Mohammadreza Sharbaf, Shekoufeh Kolahdouz Rahimi

***Abstract*—Model-driven game development (MDGD) leverages the concept of model-driven engineering and game development. The focus of MDGD is to automate the game development process by emphasizing a higher level of abstraction, which will make game development faster and easier. In recent years, researchers in the MDGD community have developed several approaches in this domain. The goal of this paper is to survey and classify existing works in MDGD, identify the challenges in this domain, and provide promising future research directions. To achieve this, we conducted a systematic review by selecting 43 articles from a set of 849. The results show that MDE techniques are used to develop games in various genres. 42% of the investigated studies proposed a graphical concrete syntax for game specification and 56% of them used different target environment tools such as Unity Engine. Moreover, our suggestions include taking advantage of tooling environments and focusing on game components rather than a complete game.**

***Index Terms*—Model-driven software engineering, Game development, Domain-specific language, Game feature model**

I. Introduction

MODEL-Driven Engineering (MDE) is a software development paradigm that aims to increase the level of abstraction by using models to simplify the process of creating software systems [1]. Model-driven game development (MDGD) has emerged as a new field that applies MDE concepts to game development domain, with the goal of creating computer games from high-level models [2]. By focusing on game design rather than implementation, development time can be reduced. Therefore, model-driven engineers will provide a domain-specific language (DSL) to game designers as a concrete syntax to apply during the modeling process.

In recent years, MDGD has been an interesting field that received a lot of attention from researchers. Different studies were conducted in this field, and various approaches have been proposed. Only three research reviews have been carried out in the MDGD domain. Two of them ([3] and [4]) only focused on the educational games domain and covered only few of the MDGD approaches. The third review ([2]) covers more approaches, but it did not investigate the MDGD approaches from a game design perspective.

Manuscript received Month 00, 0000; revised Month 00, 0000.

A. Payandeh and M. Sharbaf are with the Department of Software Engineering, University of Isfahan, Isfahan, Iran (e-mail: a.payandeh@eng.ui.ac.ir; m.sharbaf@eng.ui.ac.ir)

S.K. Rahimi was with the Department of Software Engineering, University of Isfahan, Isfahan, Iran. She is now with School of Arts, University of Roehampton, London, U.K. (e-mail: sh.rahimi@eng.ui.ac.ir, shekoufeh.rahimi@roehampton.ac.uk)

To address the deficiencies of existing MDGD studies, we conducted a systematic review of MDGD approaches considering Brereton et al. [5] and Petersen et al. [6] guidelines. We investigated 43 studies from an initial set of 849. The articles were chosen from well-known online databases such as IEEE Xplore, ACM Digital Library, Springer Link, and ScienceDirect. To decide if a found publication should be in the research or not, we filtered the search results based on inclusion and exclusion criteria explained in section IV-A. In our systematic review, we formulated four research questions to be revealed in section IV. We identify current challenges in the community and suggest potential solutions in section VI.

The remainder of this paper is organized as follows. In section II we provide background information related to MDGD domain. Section III outlines three related works in this domain. In section IV, we describe the research method used for conducting the mapping study. Section V presents the results of the study. In section VI, we discuss issues and potential future directions for the MDGD domain. Section VII mentions threats to validity of the systematic review. Finally, section VIII, we conclude the study and state some future work.

II. Background

In this section, we describe the concepts related to this research, including MDE, Narrative, Entertainment, Simulation, and Interaction (NESI) feature model, and MDGD. First, we introduce what MDE is in section II-A. Then, the NESI feature model is introduced in section II-B. Finally, in section II-C, we discuss about MDGD.

A. Model-Driven Engineering

MDE is an approach for creating software engineering artifacts. The model is the primary artifact in this process that represents a high-level abstraction of a system. Using models and focusing on a higher degree of abstraction than code allows for better understanding during the design and implementation stages. This paradigm reduces the number of errors, development time, and expenses [1].

MDE employs rapid prototyping through the use of models as primary artifacts, allowing quick exploration and validation of design choices. Automated code generation is another benefit, as models can be automatically transformed into executable code, reducing manual coding efforts. MDSE also minimizes human errors by leveraging models for automated analysis and validation, detecting inconsistencies and violations early on.

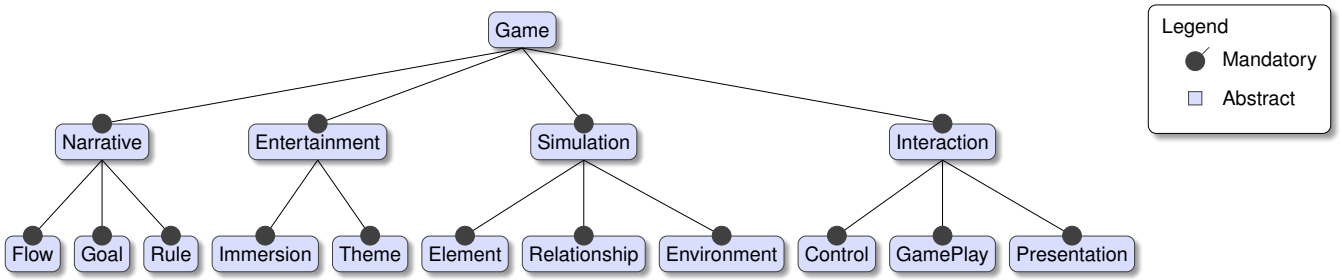

Fig. 1. NESI model for computer games representation [7].

Furthermore, by addressing errors within the models, MDSE streamlines the debugging and error-fixing process, saving time [1], [8], [9].

In MDE, a model must conform to its meta-model, i.e., it must follow the rules and constraints defined by the meta-model, which represents the abstract syntax of the language. Transformation is another important concept in MDE. After creating the models, they are automatically transformed using transformation tools into executable code [1].

B. NESI Feature Model

Variability is becoming an increasingly significant aspect of modern software development. According to Svahnberg [10], the ability to control software change in a certain context is referred to as software variability. Computer games are an example of this type of software that can change in a certain context. NESI is a feature model proposed by Sarinho and Apolinário [7] to design the variability aspects of computer games (Figure 1).

The NESI model is based on defining a game as a mix of four main standard features: Narrative, Entertainment, Simulation, and Interaction. In a nutshell, Narrative defines game Flow as a set of goals that are achieved by following defined Rules. Entertainment provides immersion when the player is playing the game based on a theme. Simulation is defined as elements (e.g., game resources, obstacles, etc.) and their relationships that exist in an environment. Finally, interaction includes game inputs and game outputs (defined as control and presentation features) which are controlled by GamePlay feature, such as game configurations [7].

C. Model-Driven Game Development

MDGD became a study focus through the application of MDE in game development. In MDGD, models are considered the foundation for game design, development, and maintenance. In game development, different experts focus on various aspects of the game. Some specialize in the game's conceptual level, while others are responsible for implementing game logic. The separation of design and implementation results in significant time and cost savings during game creation and maintenance [11].

In recent years, MDGD has been used in various aspects of computer games. For instance, Minović et al. [12], [13] used this approach to develop game UI elements, while Do Prado and Lucrédio [14] applied it in a specific game domain. Moreover, other approaches like Derakhshandi et al. [15] and Sánchez et al. [16] atteempted to use MDGD to produce a complete game by defining games in two structural and behavioral parts. However, the use of MDE in game development has some gaps, which are comprehensively explained in section VI.

III. Related Work

A survey of MDGD for educational games is provided by Tang and Hanneghan [3]. As their focus is only on the educational games domain, not much research is included in the article. Based on the review, they considered the NESI feature model [7] as a metamodel for this domain. Since the NESI model is defined as a general concept for game design, it cannot be considered as a meta-model for a specific domain like educational games. We use the NESI model to integrate MDE into the game development domain.

Another review was conducted by Zahari et al. [4], which further focused on the educational game domain, resulting in only three research papers in the educational adventure game domain. They focused on modeling as a general concept for game development and did not consider research that focuses on the integration of MDE into the game development domain.

A comprehensive review of the use of MDE in the game development domain is presented by Zhu and Wang [2], including 26 works. The MDE domain is the focus of this paper, which looks at them from a more technical viewpoint. In this research review, it was mentioned that some of the works do not support the domain they claimed to support; however, it did not provide suggestions and future directions for using MDE in game development. Additionally, they did not investigate them from a game design perspective.

Although the research in the MDGD domain in the reviews mentioned above is well-investigated, recommendations for tackling issues and clear research paths are not provided. As a result, the primary goal of this research is to examine research from a game design perspective, identify current problems in the community, classify applicable guidelines, and offer useful advice for solving those problems.

IV. Research Method

In this section, we outline the research methodology employed for this study, following the systematic mapping studies approach recommended by Brereton et al. and Petersen et al. Brereton et al. [5] advocate for a systematic literature review

(SLR) methodology, emphasizing the essential step of defining research questions during the planning phase. These research questions guide the entire review process and help focus the search and analysis on specific topics or areas of interest. The SLR methodology emphasizes the importance of formulating clear research questions that align with the objectives of the review. Similarly, Petersen et al. [6] propose a systematic mapping study (SMS) methodology, where defining research questions is a crucial step in planning the study. The research questions in an SMS play a vital role in determining the scope and purpose of the mapping study and provide guidance for the identification and classification of relevant research papers. Both methodologies recognize the significance of well-defined research questions in driving the systematic review or mapping study and ensuring that the review process remains focused and aligned with the intended objectives. This study primarily focuses on the following research questions:

- RQ1: What technologies and tooling environments are used for making MDGD tools?
- RQ2: What game features are supported by MDGD research?
- RQ3: What target domains, in terms of game genre and game dimensions, are supported?
- RQ4: What are the gaps and deficiencies in the MDGD domain?

This study includes all publications in this domain from 2004 to the end of 2022. The following sections provide descriptions of the phases of the study protocol: planning and conducting. The results of the study are covered in section V.

A. Planning Phase

The planning part of the used research protocol comprises defining the search strategy and the review process. The following explains the search strategy, while the review process is explained as part of the review conduction in section IV-B.

We identified two groups of words: the first category associated with the field of model-driven development, and the second group includes the terms related to game development. The relationship between the words within each group is OR. The following terms were used to filter out studies from relevant research. Since each group contains multiple keywords, a paper must contain at least one word from each group.

- A = Model-Driven, Model-Driven Software Engineering, MDSE, Model-Driven Engineering, MDE, Model-Driven Software Development, MDSD, Model-Driven Development, MDD, Modeling Language, Meta-Model, Domain-Specific Modeling Languages, DSML, Domain-Specific Languages, DSL, Code Generation
- B = Game Development, Game Design, Game Engine, Games, Computer Games, Video Games

These words can be combined to form the overall search string as follows:

Search String: A and B

We started by searching some of the terms on Google Scholar, which is considered a broad academic search engine [17]. Through this search, we discovered that the following online libraries hold the most relevant research in this domain.

- IEEE Xplore[1]
- ACM Digital Library[2]
- Springer Link[3]
- ScienceDirect[4]

In addition to performing an advanced search in the mentioned online databases and search engines using the search string, we ensured that the most relevant content was found using this method. Duplicate papers were removed after identifying all the papers from online databases.

The first filtering was then performed based on the title, abstract, keywords, introduction, and conclusion. Additionally, the following inclusion and exclusion criteria were set to determine which of the chosen publications should be considered as primary studies and which ones should be excluded.

Inclusion Criteria:

- Publications from 2000 to 2023.
- Publications that consider models as the central artifacts in the game development.
- Publications in English.
- Publications from journals, conferences, and workshops.

Exclusion Criteria:

- Summary, survey, or review publications.
- Any publication other than journals, conferences, or workshops.
- Books, websites, technical reports, dissertations, tutorials.
- Publications containing models that cannot be executed or converted into executable game code.
- Publications that lack game domain features based on the NESI feature model [7].

We also used an iterative snowballing step to find new studies that might be relevant to our study. The remaining papers were subjected to a full-text analysis at the end to filter out irrelevant papers. Sometimes a method is offered in various articles, each of which focuses on a different aspect of the method. In these situations, all sources were taken into account and integrated into the review process as a whole.

B. Conducting Phase

In the second phase of our systematic review, we conduct the selection process using the search protocol we defined in the planning phase. The process of finding and choosing primary studies for our review is described in more detail below. We then present the data extraction process and report the systematic map extracted from the primary studies.

1) Selection Process: Figure 2 shows the process of selecting primary studies. A database search with duplication removal and filtering is followed by full-text analysis and snowballing of remaining papers to select primary studies. We then finalize our primary studies and add sources that are spread out in different articles.

[1]https://ieeexplore.ieee.org
[2]https://dl.acm.org
[3]https://link.springer.com
[4]https://sciencedirect.com

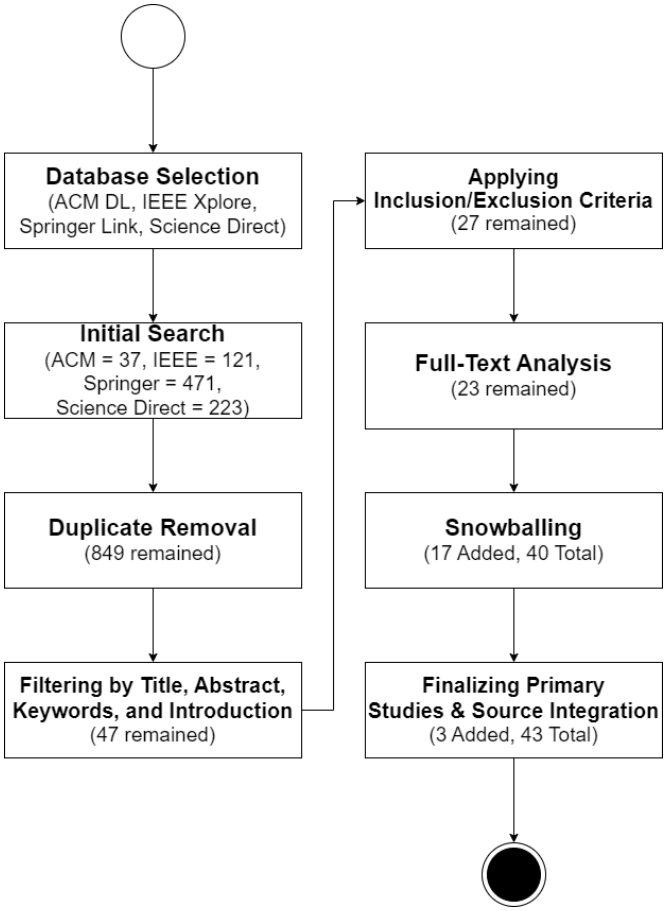

Fig. 2. Primary studies selection process

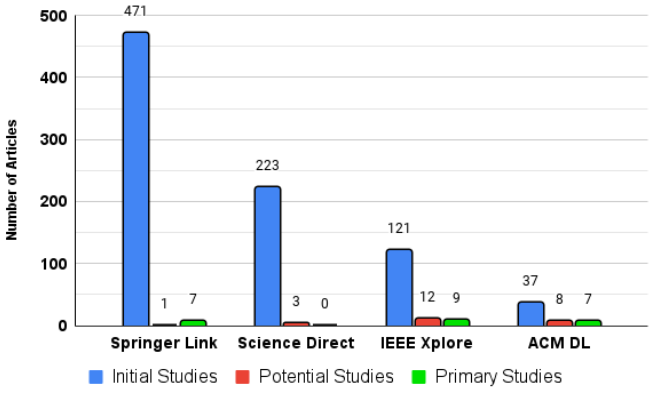

Fig. 3. Studies obtained from online databases

To begin the selection process, we ran the specified search queries on each online database to locate a set of relevant articles. Each online database has filtering options that were used to find only relevant papers such as journal, conference, or workshop papers. Figure 3 shows the number of studies retrieved from each online database. The results demonstrate that most of the articles in this field are published in IEEE.

After removing duplicate articles, there were still 849 articles that were parsed into the filtering stage. The filtering stage was done in multiple iterations. First, any article in which none of the defined terms appeared in the title, keywords, or abstract was removed. Then, for the remaining articles, any article that was not relevant to our study was excluded based on its abstract and introduction. As a result, 802 articles were rejected and 47 articles remained.

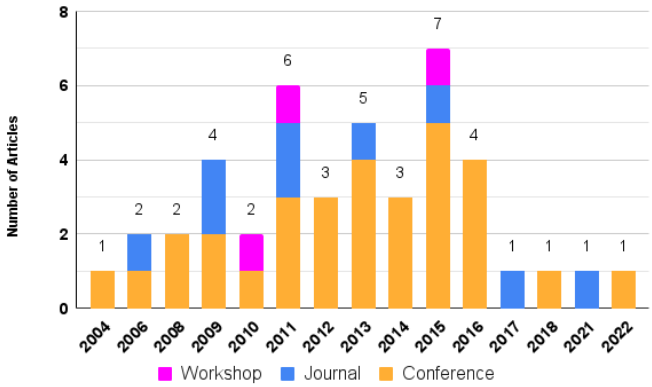

Fig. 4. Primary studies publication trends over years

In the next step, we applied inclusion and exclusion criteria, that were determined in section IV-A. The result was a set of 27 articles. A full-text analysis was performed on the remaining articles. Therefore, four of them ([18], [19], [20], and [21]) were contrary to our objectives and 23 articles remained. In the approach proposed by Thu and Nwe [18] model-driven development of mobile applications is the main focus. Although games can be a kind of mobile applications, games were not their area of interest, and no further information was provided. Ferreira et al. [19] presented a model to describe and represent location-based games, which does not involve model-driven approach in their work. Iftikhar et al. [20] proposed a model-based testing approach for automated black box functional testing of platform games. Finally, Bucchiarone et al. [21] presented the Gamification Design Framework (GDF), a tool for designing gamified applications through model-driven engineering mechanisms, which is unrelated to game development.

To reduce the possibility of missing any relevant research, we ran a snowballing step, reviewing the references of the remaining 23 articles. This way, we identified 17 more articles that met our criteria, raising our primary studies to 40. The selection process that we used demonstrates that it can be a good process for selecting the most relevant studies to the MDGD approach because, in addition to the studies previously reviewed by Zhu and Wang [2], we discovered studies that were not included in their studies but were related to their work, such as Altunbay et al. [22] and Karamanos et al. [23].

Sometimes a method is presented in multiple articles. As a result, we included any new source that may satisfy our objectives but is related to an article in the final step. This resulted in the addition of three new articles, bringing the total of studies to 43. Figure 4 shows publication trends of primary studies over years. Based on Figure 4, the use of the model-driven method in game development is increasing. Nevertheless, since 2016, less attention has been dedicated to

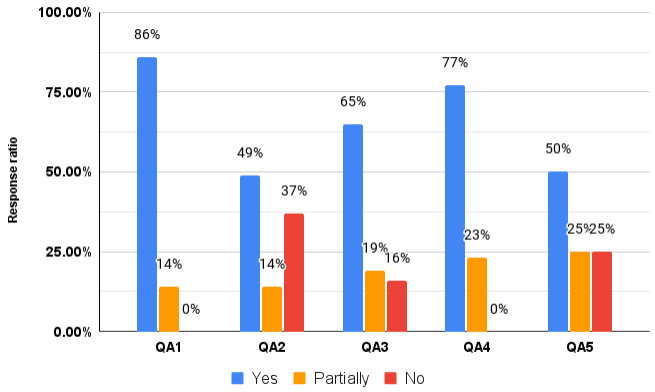

Fig. 5. Results of quality assessment of primary studies

TABLE I
QUALITY ASSESSMENT CHECKLIST

ID	Topic	Question
QA1	Objective	Did the study clearly define the research objectives?
QA2	Related Work	Did the study provide a review of previous work?
QA3	Evaluation	Did the study evaluate their approach?
QA4	Approach	Did the study explain domain concepts clearly?
QA5	Future Work	Did the study point out potential further research?

this area of study. This might be due to the wrong direction that these approaches have taken over the years. The possible causes of this incident will be covered in section VI.

To ensure that the chosen primary studies are of high quality, we additionally assessed the quality of the studies using the approach recommended by Brereton et al. [5] and Petersen et al. [6]. For this purpose, we designed a checklist with five questions for quality assessment, as shown in Table I. The score values were 'Yes' = 1, 'Partially' = 0.5, and 'No' = 0.

Figure 5 shows the percentages of primary studies that answered 'Yes', 'Partially', and 'No' to the five quality assessment questions. It demonstrates that for each of the five questions, the majority of the selected studies (63-100%) scored 'Yes' or 'Partially', confirming that the main studies chosen are of good quality.

2) Data Extraction and Synthesis: To answer each research question, we manually performed the data extraction of each primary study and gathered all requirements in a form that is shown in Table II. We reviewed all of the studies and gathered data from three perspectives:

- *General:* As a general perspective of primary studies, we gathered their information such as publication year and article type (which was mentioned previously in Figure 4). Based on this information, we will discuss the importance of this domain over the years and potential research that can be done in model-driven game development domain in section VI.

TABLE II
DATA EXTRACTION FORM

Category	Study Data	Relevant RQ
General	Title	Study Overview
	Year	
	Author	
	Article Type	
Domain Framework	Technologies	RQ1
	Target Environment	
	Future Work	RQ4
Target Game Domain	Game Features	RQ2, RQ3

- *Domain Framework:* From a technical perspective of primary studies in the model-driven development domain, we identified the technologies (such as ATL [24] and Acceleo[5]) that were used in the studies to provide a concrete syntax for the target domain. Some of the studies used a target environment (such as Unity Engine[6] and jMonkeyEngine[7]) to finalize and generate their games; therefore, these details are also retrieved. We also extracted information about potential research directions.
- *Target Game Domain:* From a game perspective of primary studies, we used the NESI feature model [7] to answer research questions 2 and 3. We chose this feature model to examine studies from a game design perspective. The NESI feature model is a model provided to design the variability aspects of video games to simplify the process of game design [7].

We believe that the right studies are chosen because we used the Brereton et al. [5] and Petersen et al. [6] guidelines for determining the primary studies, which are well-known methods and widely used approaches. The snowballing process also ensures that more relevant articles are included among the chosen articles. The quality assessment additionally shows that the majority of the selected articles have sufficient quality (63-100%).

V. RESULTS

In this section, we report the results of our systematic review. To properly answer each research question, we will first present a classification of approaches. Then, by answering each research question in a separate section, we will evaluate the studies.

A. Classification of Approaches

We chose to classify studies depending on their target environment based on the data retrieved in section IV-B2. Some of the research used tools (such as Unity Engine) as their target environment to finalize and generate executable games after generating code. Doing this would also help us assess any gaps in each strategy. Table III shows the result of classification of studies for RQ1 and RQ4.

[5]https://wiki.eclipse.org/Acceleo/Specification
[6]https://unity.com
[7]https://jmonkeyengine.org

TABLE III
CLASSIFICATION OF STUDIES FOR RQ1 AND RQ4

Category	MDGD Approach	Target Environment Tool
Without Tool (44%)	[22], [25], [26], [27], [28], [29], [30]–[32], [23], [33], [12], [13], [34], [35], [36], [16], [37], [38]	–
With Tool (56%)	[15]	Android Studio
	[39], [40], [41], [42]–[44]	Unity Engine
	[14]	jMonkeyEngine
	[45], [46]	FlatRedBall
	[47], [48]	Microsoft XNA
	[49]	Solar 2D
	[50], [51]	Aurora Toolkit
	[11], [52], [53]	Torque2D
	[54]	Adobe Flash Player
	[55], [56]	Haaf Engine
	[57]–[59]	DAE Sandbox

TABLE IV
CLASSIFICATION OF STUDIES FOR RQ2 AND RQ3

Category	MDGD Approach	Description
Genre-Based (89%)	[15], [22], [25], [26], [27], [28], [29], [30]–[32], [23], [33], [34], [35], [36], [16], [38], [39], [45], [46], [47], [40], [49], [50], [51], [11], [54], [55], [56], [42]–[44], [48], [52], [53], [57]–[59]	Different game genres such as Board, Tower defense, Physics-based, RPG, Adventure, Shooting and Platform games
Game Domain (2%)	[14]	Camera domain, Character domain, Scenario domain
Game System (2%)	[41]	Achievement System
User Interface (7%)	[12], [13], [37]	Educational game, UI components of virtual worlds

Another classification was to separate studies into different groups based on the game features that are supported by the approach. This can be used to answer RQ2 and RQ3. Table IV shows the result of classification for RQ2 and RQ3.

Next, we evaluate studies and answer research questions based on the data collected throughout the conducting phase.

B. RQ1: What target environments and tooling environments are used for making MDGD tools?

This question is separated into two parts. First, Table III shows what target environment tool is considered in each study workflow. Figure 6 also demonstrates that the most common tooling environment is Unity Engine, with four research studies using it as their target environment.

The second part of the question is about environments that were used to create a domain-specific language for games.

TABLE V
TOOLING ENVIRONMENT USED IN EACH MDGD APPROACH

Category	MDGD Approach
Eclipse Modeling Frameword (56%)	[15], [22], [25], [27], [28], [29], [30]–[32], [33], [16], [37], [38], [39], [40], [49], [41], [11], [54], [55], [56], [48], [52], [53]
Microsoft DSL Tools (9%)	[26], [45], [46], [47]
XSL Transformation Tools (4%)	[12], [13]
Aurora Toolset (4%)	[50], [51]
Magic Draw (2%)	[54]
Unity Engine (7%)	[42]–[44]
jMonkeyEngine (2%)	[14]
N/A (16%)	[23], [34], [35], [36], [57]–[59]

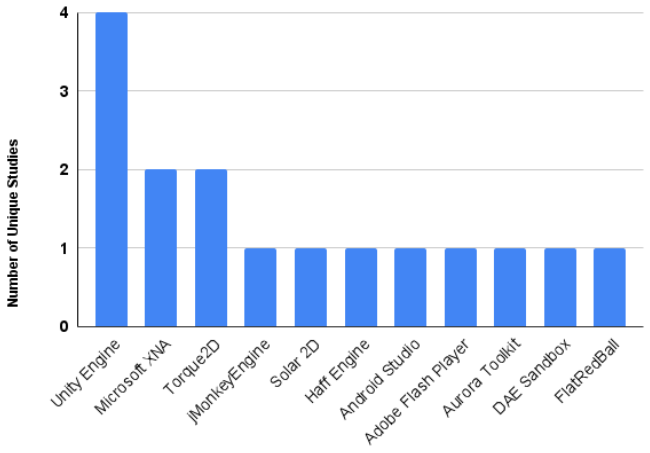

Fig. 6. Frequency of target environment tools used in different studies

Based on the retrieved data from section IV-B2, most of the studies (56%) used the Eclipse framework as their base tools for creating DSL. Other studies used tools such as Microsoft DSL Tools and Magic Draw. Table V shows what tooling environment and technologies are used in each study to create the DSL.

In each approach, as a result of using tooling environments and technologies, a platform is developed to ease the modeling process, which represents the concrete syntax of the DSL. Table VI illustrates the concrete syntax of each approach. This shows that most of the approaches (42%) provide a graphical modeling environment of any kind for users to ease the modeling process.

C. RQ2: What game features are supported by MDGD research?

Throughout the conducting phase, we investigated primary studies and extracted data based on the NESI feature model [7] down to its leaves. Based on the extracted data, not all features are supported in MDGD approaches. Table VII shows each approach with the supported features.

1) Narrative: Although most of the research supports definition of Goals and Rules in the proposed meta-model (and

TABLE VI
CONCRETE SYNTAX OF EACH MDGD APPROACH

Concrete Syntax	MDGD Approach
Graphical (42%)	[26], [29], [23], [33], [39], [45], [46], [47], [49], [54], [55], [56], [42]–[44], [57]–[59]
Textual (28%)	[25], [27], [28], [30]–[32], [12], [13], [16], [38], [41], [48]
Form-Based View (7%)	[34], [35], [14]
Tree-View (21%)	[15], [22], [37], [40], [50], [51], [11], [52], [53]
N/A (2%)	[36]

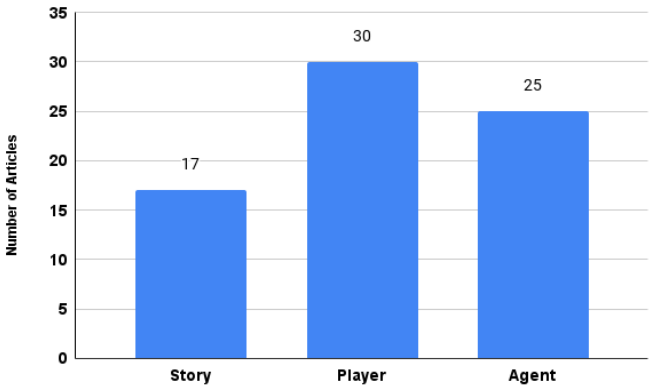

Fig. 7. Frequency of MDGD studies that support Story, Player, and Agent features

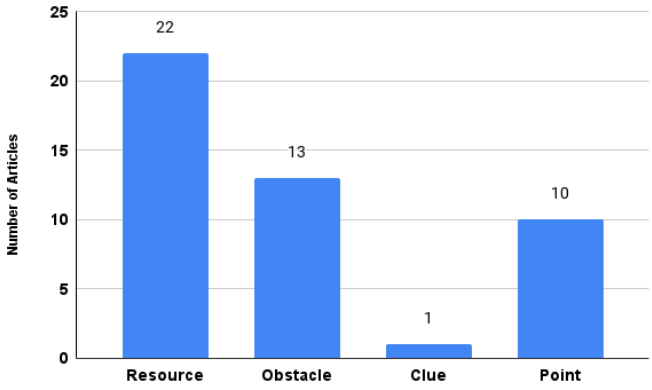

Fig. 8. Frequency of MDGD studies that support Resource, Obstacle, Clue, and Point features

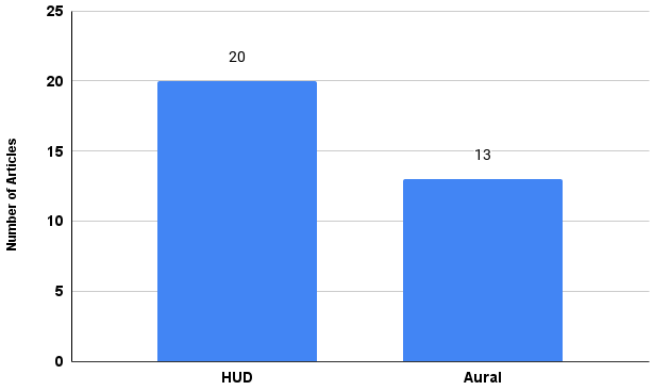

Fig. 9. Frequency of MDGD studies that support HUD and Aural features

by modeling in textual or graphical form), not all MDGD approaches support Flow of the game with an MDGD approach. For example in [57]–[59], the authors provide a graphical editor that users can use to define the communication between players and NPCs by predefined building blocks, which creates a story for player to role-play in.

A goal is composed of subgoals and can be of different types such as Move, Avoid and Survive. For example in [15], movement is defined by means of a meta-class called MovableElement. Similarly as goals, rules are also composed of subrules and can be of different types such as GameEnd, Build, Move, and Collide. In [22], a rule meta-class is defined and can be used to define EndEvaluation of the game.

2) Entertainment: Entertainment is composed of Immersion and Theme. Immersion is the feelings of the player during the game (e.g., when the player facing a Challenge) [7]. This is not something that an MDGD approach can support directly, but from an MDGD perspective, if the approach can support the definition of a challenge in some form, we can consider it as an integration of immersion into MDGD workflow. For example [39], which supports educational games, achieves this by defining a quiz.

The other part of Entertainment is Theme. A theme is composed of Avatars and a Story [7]. The game story can be presented to the player in different forms, such as a ScriptEvent or a CutScene. Not all studies support this feature. For example, in [39], you can define a video, which can be considered as a Cutscene to the player. An avatar can be the Player or an Agent (AI) in the world. Most of the approaches support the definition of Player in their meta-model, but AI is not supported in all of them. For example, in [16], users can define enemies in the textual editor with ”Enemies” and ”Enemy” keywords. Figure 7 shows the frequency of MDGD studies that support Story, Player, and Agent features.

3) Simulation: Simulation is composed of Elements in an environment that can have a Relationship with other Elements. Elements can be Resources or Obstacles. In [55], [56], users can define rewards in the modeling process. These rewards can be considered as Pickups that player has to collect. In [23], users can create walls as obstacles, which can be defined through a 2D graphical editor. Unit, Point, and Clue are also other sub-features of elements, which are not supported by most of the MDGD approaches. Figure 8 shows the frequency of MDGD studies that support resource, obstacle, clue, and point features.

Not all MDGD approaches support a way to define environment of a game. The environment can be of type World, Level, Sector, or Tile. For example, in [38], the concept of world and tile are meta-classes of the proposed meta-model.

Another aspect of simulation is relationship. Elements of a game can have relationships with other elements or the environment. For example, rewards are game elements that can be placed in a level. In most cases, this concept is

TABLE VII
SUPPORTED GAME FEATURES IN EACH MDGD APPROACH

Game Feature		MDGD Approach
Narrative	Flow	[48], [57]–[59]
	Goal	[15], [22], [25], [26], [27], [28], [30]–[32], [23], [34], [35], [16], [37], [38], [39], [14], [47], [40], [49], [41], [50], [51], [11], [54], [55], [56], [52], [53], [57]–[59]
	Rule	[15], [22], [25], [26], [27], [28], [30]–[32], [34], [35], [36], [16], [37], [38], [39], [14], [45], [46], [14], [47], [40], [49], [41], [11], [54], [55], [56], [42]–[44], [52], [53], [57]–[59]
Entertainment	Immersion	[15], [22], [25], [26], [27], [28], [23], [34], [35], [16], [39], [45], [46], [47], [40], [41], [54], [42]–[44], [52], [53], [57]–[59]
	Theme	[15], [22], [25], [26], [27], [28], [23], [12], [13], [34], [35], [36], [16], [37], [38], [39], [14], [45], [46], [47], [40], [49], [50], [51], [11], [54], [55], [56], [42]–[44], [48], [52], [53], [57]–[59]
Simulation	Element	[15], [22], [26], [27], [28], [30]–[32], [23], [34], [35], [36], [16], [37], [38], [39], [47], [49], [41], [50], [51], [11], [54], [55], [56], [42]–[44], [48], [52], [53]
	Relationship	[15], [22], [26], [27], [28], [30]–[32], [23], [34], [35], [36], [16], [37], [45], [46], [47], [40], [49], [41], [50], [51], [11], [54], [55], [56], [42]–[44], [57]–[59]
	Environment	[15], [22], [25], [26], [27], [28], [23], [33], [34], [35], [16], [37], [38], [39], [14], [45], [46], [49], [54], [55], [56], [42]–[44], [48]
Interaction	Control	[11], [55], [56]
	GamePlay	[15], [22], [34], [35], [45], [46]
	Presentation	[15], [22], [25], [26], [27], [28], [30]–[32], [23], [12], [13], [34], [35], [16], [37], [38], [39], [14], [45], [46], [47], [40], [50], [51], [11], [54], [55], [56], [42]–[44], [52], [53], [57]–[59]

automatically generated by the transformation engine. Some of the research allows us to define the relationship between game elements and the environment by means of the provided textual editor. For example, in [27], [28], you can define rewards that can appear in the level with a specific frequency.

4) Interaction: Interaction consists of Control, GamePlay, and Presentation. The concept of control is supported by [11], [55], [56]. For example, in researches [55], [56], a control mapping diagram is presented that can translate game signals into game actions. This allows users to define game actions that can occur, and the code for mapping actions to different controllers (joystick, keyboard, etc.) is generated automatically.

Each game has a GamePlay that defines how a game should be played. This can be features such as game Policy and game Configuration. For instance, board games are turn-based games that have specific policies for players. In study [15], the structure and behavior meta-model includes meta-classes like InitialTurnQueue, ChangeTurnQueue, and NextTurn, which can be used to define rules for each turn.

Another aspect of Interaction in a game is Presentation. Two important parts of a Presentation are HUD and Aural, which are crucial in computer games. Most research integrates these concepts into their meta-models. Figure 9 illustrates the frequency of MDGD studies that support HUD and Aural features.

D. RQ3: What target domains, in terms of game genre and game dimensions, are supported?

Based on the extracted data, 74% of the studies support 2D games and 30% of them support 3D games. Only [36] and [41] can support both 2D and 3D games. Another type of game is 2.5D games, but none of the approaches support 2.5D games. Game genre is another important aspect of each MDGD approach. Each MDGD approach choses one or more genres and proposes an MDGD approach to only support that specific genre. In [27], [28], authors claim that their approach covers all types of physics-based 2D games. However, this is not true since there are lots of aspects and features of 2D games, such as the ability to define custom events, that are not supported by their approach. Table VIII and IX show the supported game dimensions and genres of MDGD approaches.

TABLE VIII
SUPPORTED GAME DIMENSION OF MDGD STUDIES.

Dimension	MDGD Approach
2D (74%)	[15], [22], [25], [26], [27], [28], [29], [30]–[32], [33], [12], [13], [34], [35], [36], [16], [37], [38], [45], [46], [47], [40], [49], [41], [11], [54], [55], [56], [48], [52], [53]
3D (30%)	[23], [36], [39], [14], [41], [50], [51], [42]–[44], [57]–[59]
2.5D (0%)	–

E. RQ4: What are the gaps and deficiencies in the MDGD domain?

Based on the extracted data and quality assessment of the studies, we discovered that the majority of selected studies had three primary gaps which we will discuss next.

1) Lack of completeness of target game domain and target platform: Table IX shows that the majority of selected MDGD approaches (90%) do not support more than one game genre. When composing MDE into the game development domain, most of the studies chose a game genre and proposed an approach to support a specific game domain. Support for other game genres was left as future work. Instead of selecting a game genre, in [14] and [41], they chose to integrate the MDE domain within a specific game domain or game system (e.g. camera domain and achievement system). In this manner,

TABLE IX
CLASSIFICATION OF MDGD STUDIES BASED ON GENRE.

Dimension	MDGD Approach
Board	[15], [22]
Tower Defense	[16]
Physics-Based	[27], [28]
Touch	[34], [35]
Trivia	[34], [35]
Puzzle	[34], [35], [40]
Strategy	[34], [35]
Platform	[34], [35], [40], [55], [56]
Maze	[38]
Shooting	[38]
Racing	[38]
Fighting	[38]
Educational	[25], [12], [13], [39], [41], [42]–[44], [57]–[59]
Adventure	[26], [48]
Location-Based	[30]–[32]
Action-Adventure	[23], [52], [53]
RPG	[49], [50], [51]
Flash	[54]
Other	[29], [33], [36], [37], [14], [45], [46], [41]

we are not restricted to a specific genre. We will discuss this further in section VI.

Moreover, while some studies take a more comprehensive approach than others like [15], [16], and [57]–[59], not all of them support the whole target game domain. In most cases, the proposed meta-model lacks domain concepts. For example, some approaches like [25] and [42]–[44] that try to encompass educational games do not cover the concept of quiz in their approach. According to the data retrieved from the quality assessment of the studies, the majority of selected studies left further improvement and incorporating more domain concepts for future work.

Another aspect that was missing in most studies like [15], [26], and [27], [28] was support for multiple target platforms (e.g. iOS, Android, Windows, etc.). The majority of selected MDGD approaches (56%) only support a single target platform, with support for additional platforms left for future work. The major reason for this is that the suggested MDGD workflow does not employ an appropriate tooling environment (or none at all). More on this in section VI.

2) Lack of evaluation: One of the significant gaps in most of the studies was lack of proper evaluation of the proposed MDGD approach. The amount of automatically generated code is one of the features of an MDGD approach that needs to be evaluated [1]. This evaluation is essential because the goal of an MDE approach is to simplify development by minimizing the manual writing of software code. While some studies mentioned the percentage or line of code that is created automatically, the majority did not provide further information on this aspect.

Another crucial aspect of the MDGD approaches is comparing them with traditional approaches to writing software code (this comparison can be based on lines of code or development time). This comparison determines whether the proposed approach successfully reduces development efforts. Only [15], [34], [35], and [14] evaluated this aspect in the proposed MDGD approach.

3) Lack of proper concrete syntax: Concrete syntax is one of the essential parts of a DSL [1] and MDGD as well. A well-defined concrete syntax simplifies the modeling process. As demonstrated in Table VI, all of the studies support a way of defining models, although not all of them are adequate. Some only support tree-view editors, which become increasingly difficult to use as the meta-model becomes more complex.

In [34], [35] and [14], the concrete syntax is presented as a form-based view editor which doesn't involve a modeling process. The editor is built based on the meta-model, and models are already presented in the editor. User can then change properties of those models without worrying about the modeling process. However, this approach does not provide a flexible way to define new models.

Textual and graphical editors are better forms of concrete syntax. As shown in Table VI, most of the studies (70%) support either of those forms, and those that did not support graphical form left market for graphical editors. For example, in [30]–[32] and [16], the authors mentioned that they would provide a graphical editor in future works. Other studies, like [29], stated that the enhancing the editor was left as future work.

VI. DISCUSSION AND FUTURE DIRECTIONS

The publishing trends in the MDGD domain were increasing, and this subject was receiving more and more attention until 2016, as shown in Figure 4 in section IV-B1. However, only a few papers have been published in this field since 2017, indicating that less attention is being given to this domain. This is due to the difficulty of providing a solution that can (i) support multiple target platforms; (ii) support asset management; and (iii) build flexible games. Next, we will talk about the problems identified in the studies, along with potential solutions.

As mentioned in section V-E1, the majority of selected MDGD approaches only support a single target platform (e.g., Android, Windows, etc.). To support more than one target platform in MDE, several engines must be designed, which is difficult and time-consuming. To address this, we may employ tooling environments in the MDGD workflow. As demonstrated in Table III, almost half of the studies (44%) did not use a tooling environment in the MDGD workflow. Most of these tools have support for multiple target platforms, and using a tooling environment such as Unity Engine can help us with that. This also allows MDGD developers to do less work, as they won't have to write multiple engines to support multiple target platforms.

One of the most important aspects of game development is managing game assets, such as texture and audio files. This is also crucial for MDGD developers to support asset management in their workflow. Currently, this is achieved through the development of concrete syntax, similar to what Sánchez et

al. [16] did in their textual editor. However, this approach is inefficient and makes it challenging for users to comprehend and define game assets using the provided concrete syntax. Moreover, implementing asset management in the concrete systax is difficult and time-consuming for MDGD developers. As a result, none of the studies proposed a complete MDGD approach for game development, as mentioned in section V-E1.

Creating a program that manages game assets essentially involves building a game engine from scratch. Modern game engines are continuously evolving, with new features being added regularly. Instead of starting from scratch, we can leverage existing well-known game engines like Unity or Unreal. Only [14] accomplished this by integrating their approach into jMonkeyEngine and providing a form-based interface for specifying various game domains. However, as stated in section V-E3, a form-based view editor is not suitable for an MDGD approach. Our suggestion is to integrate the MDGD process into a game engine in a way that allows users to define models directly within the engine.

To utilize existing engine capabilities and provide a solution, MDGD engineers need to understand a game engine's architecture, be familiar with its codebases, and grasp how the engine operates. For example, consider Unreal Engine as an engine suitable for deploying MDGD tools. Unreal Engine has a module-based architecture, which is managed by the Unreal Header Tool (UHT), a component of the Unreal Build Tool (UBT). Each part of the engine is presented as a module. To create a tool based on this engine, users must understand what a module is and how to create one. Another challenge is to develop a transformation engine, similar to Acceleo for Eclipse. Although this task is difficult, once the transformation engine is available, researchers can utilize it to build their DSLs based on the engine itself. By offering a tool integrated directly into the engine, game developers can utilize the provided MDGD tool within the engine itself to construct their games and efficiently manage game assets.

As noted in section V-E1, almost all of the studies (89%) focused on a specific game genre and attempted to cover the entire domain using an MDGD method. However, none of them were complete. A complete game in any genre includes several components that a game developer must take into account during development. Additionally, different styles may exist within the same genre. This complexity makes it impossible for DSL engineers to build a solution that covers all variations, resulting in rigid and insufficient solutions. For example, none of the studies examined methods to support a player's in-game abilities or the persistence of the entire game. Instead of aiming to cover an entire genre and create a complete game, we can choose to focus on specific game component. Although this approach does not result in a complete game, it allows game designers to manage each aspect of the game individually, minimizing the time required to create that specific part. By adopting this approach, we are not restricted to a particular game genre. Furthermore, by including and supporting other game components, we can eventually provide game developers with a comperhensive framework to create game of their choice. Examples of this approach include [14], which focused on the camera, character, and scenario domains, and [41], which chose to cover the achievement system as the game component.

While this suggestions has the benefit to focus on a single component rather than a whole game, there are some potential drawbacks. Firstly, there may be limitations in the reusability of components across different game genres, as some genres may require specialized components or modifications to existing ones. Secondly, managing the complex interactions between components can be challenging, as each component may have its own dependencies and ensuring seamless coordination can become intricate. One possible solution to this might be to implement an event-based system through delegates in each component so that any changes are dispatched to other components, giving them a chance to update their state. Lastly, testing and integration pose challenges, as comprehensive testing is needed to validate both the individual functionality of components and their compatibility when integrated with other parts of the game. These drawbacks highlight the need for careful consideration of component design, interdependencies, and a robust testing strategy to ensure successful implementation of the MDGD approach.

Based on the results found in section V, MDGD has demonstrated a range of advantages and benefits that contribute to more efficient and effective game development processes. One key advantage lies in the abstraction and visualization capabilities offered by MDGD. By working at a higher level of abstraction through models and visual representations, developers can enhance communication and understanding among stakeholders, including designers and developers. This facilitates collaboration and reduces the gap between domain experts and technical implementers (e.g. gameplay, AI, audio and tools programmers), enabling a shared language for expressing game concepts and logic. Additionally, MDGD empowers rapid prototyping, allowing for iterative development cycles. With changes made to the models automatically translated into corresponding code, MDGD accelerates the development process, saving valuable time and effort. This iterative approach supports more fluid experimentation, enabling designers and developers to refine and fine-tune game mechanics and overall gameplay experience.

Another significant benefit of MDGD is the separation of concerns it facilitates. By dividing the development process into modeling and code generation, MDGD promotes reusability of game components. This separation enables developers to focus on the high-level design and logic of the game without getting lost in implementation details, which results in less development time. Additionally, MDGD often employs domain-specific languages (DSLs) tailored to game development. These DSLs provide intuitive and expressive ways to represent game concepts, enabling domain experts to directly contribute to the development process. Moreover, the power of model transformation and code generation in MDGD reduces the risk of human error and ensures consistency across the game codebase. Developers can generate code for multiple platforms from a single set of models, streamlining the deployment and adaptation of games to different target environments.

While MDGD offers numerous advantages, it also faces

several challenges and weaknesses that need to be considered. One significant challenge is the complexity and learning curve associated with MDGD approaches. Developers often need to learn new modeling languages and tools which can pose a barrier to entry and require additional time and effort for adoption. The learning curve can be particularly steep for developers who are not already familiar with modeling concepts, potentially slowing down the development process. Although it is important to note that once we have learned these tool, we can leverage its capabilities without the need for further learning, allowing us to use it efficiently going forward.

Furthermore, the expressive power of modeling languages in MDGD may be limited compared to traditional programming languages. This limitation can make it challenging to represent complex game behaviors and interactions solely through models, potentially requiring additional manual coding to achieve the desired functionality. This is shown in Table VII, where not all studies cover all game features. Additionally, the performance and optimization of generated code can be a concern in MDGD.

Lastly, the availability and maturity of MDGD tools and frameworks may vary, posing limitations and affecting the overall adoption and applicability of MDGD approaches. Some tools may lack comprehensive features or have limitations that restrict their usage in certain contexts. Addressing these challenges requires continued research and development efforts to improve tooling, enhance the expressiveness of modeling languages and ensure the availability of robust and user-friendly MDGD tools.

VII. Threats To Validity

The final pool of article selection is the main area where the validity of this systematic review is at risk. To ensure that the right studies were chosen to minimize these possible risks, we followed the guidelines of Brereton et al. [5] and Petersen et al. [6]. The elimination of inaccurate papers during the filtering step is one of the concerns that endangers our systematic review. While checking the titles of studies, we might have removed some of the related research that does not contain any of our search terms, as they might have used general terms in the title. Although adding the snowballing phase to our selection process has the advantage of lowering this risk by restoring studies that could have been accidentally removed. Another concern is the selection of inappropriate inclusion and exclusion criteria and missing potential search terms.

VIII. Conclusion

In this paper, we presented a systematic review of the studies in the MDGD domain. By identifying and categorizing the existing MDGD approaches, our goal was to assess the state-of-the-art in this field and suggest potential future research directions. From 849 research studies found through automatic searches in well-known online libraries, we finally selected and analyzed 43 primary studies that present 30 unique approaches to the MDGD domain.

Our findings show that (i) the majority of selected MDGD approaches target a specific game genre and attempt to cover the entire domain in a single approach; (ii) different game genres were selected among studies (iii) while some of the studies do not include a target environment tool in the proposed approach, others make use of them to ease their workflow; (iv) different tooling environments were used to provide a DSL and Eclipse being the most common one; (v) the most common concrete syntax was in graphical and textual form.

Currently, research in MDGD tends to concentrate on developing approaches for individual game genres. However, this approach limits the applicability of the research, as a new study is required for each unsupported game genre. Instead, the suggestion proposes concentrating on game components, which are the building blocks of games, rather than focusing on specific genres. By doing so, developers can use the presented approaches and tools to model and generate code for specific game components. This component-centric approach enables reusability, maintainability, flexibility, and efficiency, allowing developers to easily incorporate different components into various game genres without having to start from scratch.

Finally, we discussed publication trends of the studies and investigated issues in the MDGD field, indicating potential directions that researchers can choose. One crucial area is tooling and infrastructure. Continued development and refinement of robust, user-friendly modeling tools, and code generation frameworks will contribute to the wider adoption and effectiveness of MDGD. Improving the tooling landscape can provide developers with more intuitive interfaces, enhanced features, and better integration within existing game development workflows, reducing barriers to entry and streamlining the development process. This requires a tremendous amount of work for researchers, as making such tools and environments are really hard. To address this issue, we suggest researchers to use existing tools such as Unity or Unreal in their workflow. This will give them the benefits of supporting game assets and multiple target platforms.

Furthermore, enhancing the expressiveness of modeling languages used in MDGD is essential. As shown in table VII, now all game features are supported in a single study. Researchers can explore techniques to extend the capabilities of modeling languages, enabling them to represent complex game systems, behaviors, and interactions more comprehensively. This improvement will allow developers to model and express intricate game logic solely through the use of models, reducing the need for manual coding and increasing the productivity and maintainability of MDGD projects.

References

[1] M. Brambilla, J. Cabot, and M. Wimmer, *Model-Driven Software Engineering in Practice*, 2nd ed. USA: Morgan & Claypool, 2017.

[2] M. Zhu and A. I. Wang, "Model-driven game development: A literature review," *ACM Computing Surveys (CSUR)*, vol. 52, no. 6, pp. 1–32, November 2019.

[3] S. Tang and M. Hanneghan, "State-of-the-art model driven game development: A survey of technological solutions for game-based learning," *Journal of Interactive Learning Research*, vol. 22, no. 4, pp. 551–605, December 2011.

[4] A. S. Zahari, L. A. Rahim, and M. Mehat, "A review of modelling languages for adventure educational games," in *2016 3rd International Conference on Computer and Information Sciences (ICCOINS)*, August 2016, pp. 495–500.

[5] P. Brereton, B. A. Kitchenham, D. Budgen, M. Turner, and M. Khalil, "Lessons from applying the systematic literature review process within the software engineering domain," *Journal of Systems and Software*, vol. 80, no. 4, pp. 571–583, April 2007.

[6] K. Petersen, S. Vakkalanka, and L. Kuzniarz, "Guidelines for conducting systematic mapping studies in software engineering: An update," *Information and Software Technology*, vol. 64, pp. 1–18, August 2015.

[7] V. Sarinho and A. Apolinário, "A feature model proposal for computer games design," in *VII Brazilian Symposium on Computer games and Digital entertainment, Belo horizonte*, 2008, pp. 54–63.

[8] S. Kelly and J.-P. Tolvanen, *Domain-specific modeling: enabling full code generation*. John Wiley & Sons, 2008.

[9] S. Sendall and W. Kozaczynski, "Model transformation: The heart and soul of model-driven software development," *IEEE software*, vol. 20, no. 5, pp. 42–45, 2003.

[10] M. Svahnberg, J. Van Gurp, and J. Bosch, "A taxonomy of variability realization techniques," *Software: Practice and experience*, vol. 35, no. 8, pp. 705–754, July 2005.

[11] L. M. Morales, D. Méndez-Acuña, and W. Montes, "Model driven game development - case study a mtc for maze game prototyping," *Revista electrónica en construcción de software PARADIGMA*, vol. 5, pp. 1–15, 2011.

[12] M. Minovic, M. Milovanovic, M. Jovanovic, and D. Starčević, "Model driven development of user interfaces for educational games," in *2009 2nd Conference on Human System Interactions*, May 2009, pp. 611–617.

[13] M. Jovanovic, D. Starcevic, M. Minovic, and V. Stavljanin, "Motivation and multimodal interaction in model-driven educational game design," *IEEE Transactions on Systems, Man, and Cybernetics - Part A: Systems and Humans*, vol. 41, no. 4, pp. 817–824, July 2011.

[14] E. F. do Prado and D. Lucrédio, "A flexible model-driven game development approach," in *2015 IX Brazilian Symposium on Components, Architectures and Reuse Software*, Septembre 2015, pp. 130–139.

[15] M. Derakhshandi, S. Kolahdouz-Rahimi, J. Troya, and K. Lano, "A model-driven framework for developing android-based classic multiplayer 2d board games," *Autom. Softw. Eng.*, vol. 28, pp. 1–57, November 2021.

[16] K. Sánchez, K. Garcés, and C. Rubby, "A dsl for rapid prototyping of cross-platform tower defense games," in *2015 10th Computing Colombian Conference (10CCC)*, September 2015.

[17] M. Gusenbauer, "Google scholar to overshadow them all? comparing the sizes of 12 academic search engines and bibliographic databases," *Scientometrics*, vol. 118, no. 1, pp. 177–214, January 2019.

[18] E. E. Thu and N. Nwe, "Model driven development of mobile applications using drools knowledge-based rule," in *2017 IEEE 15th International Conference on Software Engineering Research, Management and Applications (SERA)*, 2017, pp. 179–185.

[19] C. Ferreira, L. F. Maia, C. Salles, F. Trinta, and W. Viana, "A model-based approach for designing location-based games," in *2017 16th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*, 2017, pp. 29–38.

[20] S. Iftikhar, M. Z. Iqbal, M. U. Khan, and W. Mahmood, "An automated model based testing approach for platform games," in *2015 ACM/IEEE 18th International Conference on Model Driven Engineering Languages and Systems (MODELS)*, 2015, pp. 426–435.

[21] A. Bucchiarone, A. Cicchetti, and A. Marconi, "Gdf: A gamification design framework powered by model-driven engineering," in *2019 ACM/IEEE 22nd International Conference on Model Driven Engineering Languages and Systems Companion (MODELS-C)*, 2019, pp. 753–758.

[22] D. Altunbay, E. Cetinkaya, and M. G. Metin, "Model driven development of board games," in *the First Turkish Symposium on Model-Driven Software Development (TMODELS)*, May 2009, pp. 1–15.

[23] C. Karamanos and N. M. Sgouros, "Automating the implementation of games based on model-driven authoring environments," in *International Conference on Entertainment Computing*, 2012.

[24] F. Jouault, F. Allilaire, J. Bézivin, I. Kurtev, and P. Valduriez, "Atl: a qvt-like transformation language," in *Companion to the 21st ACM SIGPLAN symposium on Object-oriented programming systems, languages, and applications*, October 2006, pp. 719–720.

[25] M. Funk and M. Rauterberg, "Pulp scription: A dsl for mobile html5 game applications," in *International Conference on Entertainment Computing*, September 2012, pp. 504–510.

[26] A. W. B. Furtado and A. L. M. Santos, "Using domain-specific modeling towards computer games development industrialization," in *The 6th OOPSLA workshop on domain-specific modeling (DSM06)*, October 2006, pp. 1–14.

[27] V. Guana and E. Stroulia, "Phydsl: A code-generation environment for 2d physics-based games," in *Proceedings of the IEEE Games, Entertainment, and Media Conference (IEEE GEM'14)*, October 2014.

[28] V. Guana, E. Stroulia, and V. Nguyen, "Building a game engine: A tale of modern model-driven engineering," in *2015 IEEE/ACM 4th International Workshop on Games and Software Engineering*, May 2015, pp. 15–21.

[29] R. Guerreiro, A. Rosa, V. Sousa, V. Amaral, and N. Correia, "Ubilang: Towards a domain specific modeling language for specification of ubiquitous games," in *INForum*, 2010.

[30] H. Guo, H. Traetteberg, A. I. Wang, and S. Gao, "Pergo: An ontology towards model driven pervasive game development," in *OTM Confederated International Conferences "On the Move to Meaningful Internet Systems"*, October 2014, pp. 651–654.

[31] H. Guo, H. Trætteberg, A. I. Wang, and S. Gao, "Realcoins: A case study of enhanced model driven development for pervasive games," *Int. J. Multi. Ubiquitous Eng.*, vol. 10, pp. 395–410, May 2015.

[32] ——, "A workflow for model driven game development," in *2015 IEEE 19th International Enterprise Distributed Object Computing Conference*, 2015, pp. 94–103.

[33] S. Maier and D. Volk, "Facilitating language-oriented game development by the help of language workbenches," in *Proceedings of the 2008 Conference on Future Play: Research, Play, Share*, 2008.

[34] E. R. Núñez-Valdez, Óscar Sanjuán Martínez, B. C. P. G. Bustelo, J. M. C. Lovelle, and G. I. Hernández, "Gade4all: Developing multiplatform videogames based on domain specific languages and model driven engineering," *Int. J. Interact. Multi. and Artif. Intell.*, vol. 2, pp. 33–42, June 2013.

[35] E. R. Núñez-Valdez, V. García-Díaz, J. M. C. Lovelle, Y. S. Achaerandio, and R. González-Crespo, "A model-driven approach to generate and deploy videogames on multiple platforms," *J. Ambient Intell. and Humaniz. Comput.*, vol. 8, pp. 435–447, June 2017.

[36] E. M. Reyno and J. A. C. Cubel, "A platform-independent model for videogame gameplay specification," in *DiGRA Conference*, September 2009.

[37] M. Stürner and P. Brune, "Virtual worlds on demand? model-driven development of javascript-based virtualworld ui components for mobile apps," in *2016 4th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, February 2016.

[38] R. E. Vargas, G. Arellano, H. D. Beltran, L. Zepeda-Sánchez, R. Quintero, and L. Vega, "Mda game design for video game development by genre," in *Proceedings of the MODELS-JP 2013 Co-located with the 16th International Conference on Model-driven Engineering Languages and Systems (MODELS'13)*, 2013, pp. 66–70.

[39] N. Aouadi, P. Pernelle, C. B. Amar, T. Carron, and S. Talbot, "Models and mechanisms for implementing playful scenarios," in *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*, November 2016, pp. 1–8.

[40] J. Kritz, M. de Roos, L. F. Pires, J. L. R. Moreira, and G. Guizzardi, "Ugamefeature: Automatic code generation for unity game projects," in *10th International Conference on Model-Driven Engineering and Software Development (MODELSWARD)*, January 2022, pp. 371–378.

[41] A. Matallaoui, P. Herzig, and R. Zarnekow, "Model-driven serious game development integration of the gamification modeling language gaml with unity," in *Proceedings of the Annual Hawaii International Conference on System Sciences*, January 2015, pp. 643–651.

[42] N. Thillainathan, H. Hoffmann, and J. M. Leimeister, "Shack city– a serious game for apprentices in the field of sanitation, heating and cooling (shac)," pp. 2402–2413, 2013.

[43] N. Thillainathan, H. Hoffmann, E. M. Hirdes, and J. M. Leimeister, "Enabling educators to design serious games – a serious game logic and structure modeling language," in *European Conference on Technology Enhanced Learning*, September 2013, pp. 643–644.

[44] N. Thillainathan and J. M. Leimeister, "Educators as game developers—model-driven visual programming of serious games," in *Knowledge, Information and Creativity Support Systems*, January 2016, pp. 335–349.

[45] A. W. B. Furtado, A. L. M. Santos, and G. L. Ramalho, "Sharpludus revisited: from ad hoc and monolithic digital game dsls to effectively customized dsm approaches," in *Proceedings of the compilation of the co-located workshops on DSM'11, TMC'11, AGERE! 2011, AOOPES'11, NEAT'11, & VMIL'11*, October 2011, pp. 57–62.

[46] A. W. B. Furtado, A. L. M. Santos, G. L. Ramalho, and E. S. de Almeida, "Improving digital game development with software product lines," *IEEE Software*, pp. 30–37, August 2011.
[47] F. E. Hernandez and F. R. Ortega, "Eberos gml2d: A graphical domain-specific language for modeling 2d video games," in *DSM '10: Proceedings of the 10th Workshop on Domain-Specific Modeling*, November 2010.
[48] R. Walter and M. Masuch, "How to integrate domain-specific languages into the game development process," in *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology*, November 2011, pp. 1–8.
[49] E. R. B. Marques, V. Balegas, B. F. Barroca, A. Barišić, and V. Amaral, "The rpg dsl: A case study of language engineering using mdd for generating rpg games for mobile phones," in *Proceedings of the 2012 workshop on Domain-specific modeling*, October 2012, pp. 13–18.
[50] M. McNaughton, M. Cutumisu, D. Szafron, J. Schaeffer, J. Redford, and D. Parker, "Scriptease: Generative design patterns for computer role-playing games," in *Proceedings. 19th International Conferenceon Automated Software Engineering, 2004*, September 2004, pp. 88–99.
[51] M. Cutumisu, D. Szafron, J. Schaeffer, M. McNaughton, T. Roy, C. Onuczko, and M. Carbonaro, "Generating ambient behaviors in computer role-playing games," *IEEE Intelligent Systems*, vol. 21, no. 5, pp. 19–27, October 2006.
[52] M. Zhu, A. I. Wang, and H. Trætteberg, "Engine- cooperative game modeling (ecgm): Bridge model-driven game development and game engine tool-chains," in *Proceedings of the 13th International Conference on Advances in Computer Entertainment Technology*, November 2016, pp. 1–10.
[53] M. Zhu and A. I. Wang, "Rail: A domain-specific language for generating npc behaviors in action/adventure game," in *Proceedings of the International Conference on Advances in Computer Entertainment*, 2018, pp. 868–881.
[54] A. Pleuss and H. Hussmann, "Model-driven development of interactive multimedia applications with mml," *Model-driven development of advanced user interfaces*, pp. 199–218, 2011.
[55] E. M. Reyno and J. A. C. Cubel, "Model driven game development: 2d platform game prototyping," in *GAMEON*, January 2008, pp. 5–7.
[56] ——, "Automatic prototyping in model-driven game development," *Comput. Entertain.*, vol. 7, pp. 1–9, June 2009.
[57] F. V. Broeckhoven and O. D. Troyer, "Attac-l: A modeling language for educational virtual scenarios in the context of preventing cyber bullying," in *2013 IEEE 2nd International Conference on Serious Games and Applications for Health (SeGAH)*, May 2013, pp. 1–8.
[58] O. Janssens, K. Samyny, R. V. d. Walle, and S. V. Hoecke, "Educational virtual game scenario generation for serious games," in *2014 IEEE 3nd International Conference on Serious Games and Applications for Health (SeGAH)*, May 2014, pp. 1–8.
[59] S. V. Hoecke, K. Samyn, G. Deglorie, O. Janssens, P. Lambert, and R. V. d. Walle, "Enabling control of 3d visuals, scenarios and non-linear gameplay in serious game development through model-driven authoring," in *International Conference on Serious Games, Interaction, and Simulation*, vol. 161, September 2015, pp. 103–110.