Languages of Games and Play: A Systematic Mapping Study

RIEMER VAN ROZEN*, Amsterdam University of Applied Sciences

Digital games are a powerful means for creating enticing, beautiful, educational, and often highly addictive interactive experiences that impact the lives of billions of players worldwide. We explore what informs the design and construction of good games in order to learn how to speed-up game development. In particular, we study to what extent *languages*, *notations*, *patterns* and *tools*, can offer experts theoretical foundations, systematic techniques and practical solutions they need to raise their productivity and improve the quality of games and play. Despite the growing number of publications on this topic there is currently no overview describing the state-of-the-art that relates research areas, goals and applications. As a result, efforts and successes are often one-off, lessons learned go overlooked, language reuse remains minimal, and opportunities for collaboration and synergy are lost. We present a systematic map that identifies relevant publications and gives an overview of research areas and publication venues. In addition, we categorize research perspectives along common objectives, techniques and approaches, illustrated by summaries of selected languages. Finally, we distill challenges and opportunities for future research and development.

CCS Concepts: • **General and reference → Surveys and overviews**; • **Applied computing → Computer games**; • **Software and its engineering → Domain specific languages**; *Visual languages*; *Design languages*.

ACM Reference Format:
Riemer van Rozen. 2020. Languages of Games and Play: A Systematic Mapping Study. *ACM Comput. Surv.* 1, 1, Article 1 (January 2020), 111 pages. https://doi.org/12.3456/1234567.1234567

1 INTRODUCTION

In the past decades, digital games have become a main podium for creative expression enabling new forms of play and interactive experiences that rival the works of great historical writers, painters, artists and composers. The game development industry is a vast and lucrative branch of business that eclipses traditional arts and entertainment sectors, outgrowing even the movie industry [52]. Games reach audiences around the world, unite players in common activities and give rise to subcultures and trends that impact pass-time, awareness and policies of modern societies.

However, for every outstanding success exist many games with unrealized potential and failures that preceded bankruptcy. Developing high quality games is dreadfully complicated because game design is intrinsically complex. We wish to learn what informs the design of good games in order to help speed-up the game development process for creating better games more quickly. In particular, we study to what extent *languages*, *structured notations*, *patterns* and *tools*, can offer designers and developers theoretical foundations, systematic techniques and practical solutions they need to raise their productivity and improve the quality of games and play.

*This work has been carried out at Centrum Wiskunde & Informatica under the NWO/SIA grants: Early Quality Assurance in Software Production (EQuA), Automated Game Design (AGD) and Live Game Design (LGD)

Author's address: Riemer van Rozen, r.a.van.rozen@hva.nl, Amsterdam University of Applied Sciences.

0360-0300/2020/1-ART1 $15.00
https://doi.org/12.3456/1234567.1234567

We propose the term *'languages of games and play'* for language-centric approaches for tackling challenges and solving problems related to game design and development. Despite the growing number of researchers and practitioners that propose and apply these languages, there is currently no overview of publications that relates languages, goals and applications. As a result, publications on the topic lack citations of relevant related work. In addition, lessons learned are overlooked and available methods and techniques for language development often remain unused. As a consequence, it remains difficult to compare and study games, designs and research contributions in order to build bodies of knowledge that describe best practices and industry standards.

We aim to map the state-of-the-art of languages of games and play in an understandable way, such that it is accessible to a wide audience. Our goal is to provide a means for 1) informing practitioners and researchers about the breadth of related work; 2) sharing knowledge between research areas and industry for improved results and collaboration; 3) enabling the application of available techniques; and 4) identifying opportunities for future research and development.

We pose the following research questions:

- Which publication venues include papers on languages of games and play?
- How do the various approaches compare?
- What are open research challenges and opportunities for future work?

For answering these questions we conduct a survey of languages of games and play called a *systematic mapping study*. Mapping studies provide a wide overview of a research area by identifying, categorizing and summarizing all existing research evidence that supports broad hypotheses and research questions [26]. In contrast, systematic literature reviews usually have a more narrow focus, and instead perform in-depth analyses to answer particular research questions. Both enjoy the benefits of a well-defined methodology for (re)producing high quality results and reducing bias.

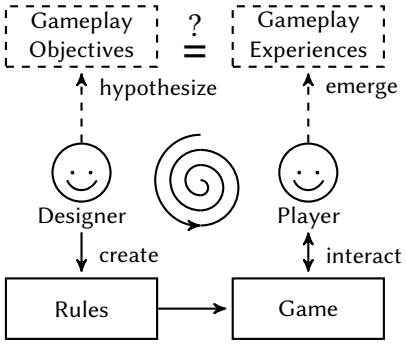
We identify and analyze relevant publications on languages of games and play. First, we motivate the need for this study by describing its scope in Section 2. Next, we describe the methodology with research questions, sources, queries and inclusion criteria, and a review protocol in Sections 3 and 4. We contribute the following:

(1) A systematic map on languages of games and play that provides an overview of research areas and publication venues, presented in Section 5.
(2) A set of fourteen complementary research perspectives on languages of games and play synthesized from summaries of over 100 distinct languages we identified in over 1400 publications, presented in Section 6.
(3) An analysis of general trends and success factors, and one unifying specific perspective on *'automated game design'*, which discusses challenges and opportunities for future research and development, presented in Section 7.

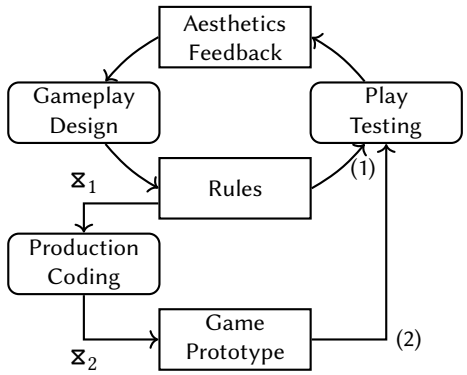
We describe related work in Section 8, discuss threats to validity in Section 9, and conclude in Section 10. Our map provides a good starting point for anyone who wishes to learn more about the topic.

2 RESEARCH VISION

A mapping study on games and play can be approached form different research perspectives, each with different goals and needs. We introduce challenges of digital game design in Section 2.1, and formulate two general hypotheses that drive this study in Section 2.2. Our specific motivation is to automate game design and investigate how domain-specific language technology, introduced in Section 2.3, can offer solutions. We clarify our position and motivate this study in Section 2.4.



(a) Game designers form hypotheses about how a game's rules realize gameplay goals, but usually repeatedly find intended and actual experiences differ



(b) Developing a high quality game entails iteratively designing, playtesting and improving its rules as a paper- (1) or a software- prototype (2)

Fig. 1. Game development aims for games with high quality player experiences

2.1 Games and Play

Games and play are inextricably intertwined concepts. Games bring about experiences such as enjoyment, persuasion and learning. There exist many different view points, explanations and definitions. We share a well-known definition of Juul, who studies games, examines similarities between them and proposes:

> *"A game is (1) a rule-based formal system with (2) a variable and quantifiable outcome, where (3) different outcomes are assigned different values [good/bad], (4) the player exerts effort in order to influence the outcome [challenge], (5) the player feels attached to the outcome, and (6) [real-world] consequences of the activity are negotiable."* [25].

Game design [17, 39], the discipline and process of iteratively designing and improving games, is an instance of a so called *wicked problem*, a problem that is *"difficult to solve in general due to incomplete, contradicting and evolving requirements"* [11, 13, 31]. We highlight challenges that illustrate its inherent complexity.

Improving a game's qualities depends on gradually improving insight, as illustrated by Figure 1a. Game designers use *paper prototyping* to explore and understand the problem at hand, abstracting away a game's details until what remains is essential. They form hypotheses about play, experiment with rules and objectives to evolve a game's design and learn what the solution can become. Designers create interaction mechanisms (a.k.a. game mechanics, or rules) offering playful affordances [38].

Players interact with games via these mechanisms during a game's execution. Playful acts result in dynamic interaction sequences. Ideally, these also represent aesthetically pleasing experiences called *gameplay*, e.g., fellowship, challenge, fantasy, narrative, discovery or self-expression [24]. However, opinions on a game's quality differ from person to person, e.g., with age, gender and beliefs.

For game designers *playtesting* is essential for verifying assumptions and learning if a game meets its objectives. More often than not, designers discover that the realized and intended gameplay differ. Unfortunately, even well prototyped games may fail to meet expectations as fully developed software. In general, it is hard to predict the outcome of modifying a game's parts, e.g., how

changing the rules affects the dynamics and aesthetics of play. As a result, steering towards new goals is difficult.

Improving a game is never truly done. The maximum number of game design iterations determines the achievable quality. Efforts on balancing, fine-tuning and polishing are limited only by time and money. Resource-wise, AAA studios have a competitive advantage over indie game developers. However, developing novel high quality games in a time-to-market manner is universally hard because game design iterations take simply too much time. Figure 1b shows an abstract game development process that illustrates the root causes of delay (shown as ⧗).

Game designers and software engineers usually live on opposite sides of the fence [27]. Both lose time when adjustments best understood by designers have to be implemented by software engineers ($⧗_1$). To evolve a game, designers have to explore alternative gameplay scenarios, constantly requiring changes.

As time progresses, more and more choices become fixed, and frequent changes to the source code become more difficult, time-consuming and error-prone ($⧗_2$). The evolution of digital games, like other software, suffers from a well-known phenomenon called *software decay* [33]. The software quality deteriorates with frequent changes to the source code made to accommodate evolving requirements. As a consequence, game designers have precious few chances to experiment with design alternatives. This seriously compromises their ability to design, prototype and playtest. Unfortunately, the complexity of game design all too often prevents development teams from timely achieving the optimal quality.

These challenges urgently require solutions. Our brief discussion indicates that rules, objectives and gameplay assumptions are artifacts that require appropriate notations for constructing high quality digital games. However, game designers lack a common vocabulary for expressing gameplay. Next, we address this need.

2.2 Languages of Games and Play

Languages of games and play are language-centric approaches for tackling challenges and solving problems related to game design and development. We propose studying existing languages and creating new ones. Two central hypotheses drive this study. We formulate a general and a specific hypothesis:

(1) *Languages*, *structured notations*, *patterns* and *tools* can offer designers and developers theoretical foundations, systematic techniques and practical solutions they need to raise their productivity and improve the quality of games and play.
(2) "Software" languages (and specifically domain-specific languages) can help automate and speed-up game design processes.

Languages of games and play exist in many shapes and forms. The next section describes one specific technical point of view that represents the departure point of this study, which also details and motivates the second more specific hypothesis.

2.3 Domain-Specific Languages

We aim to deliver solutions that automate game design and speed-up game development with so-called Domain-Specific Languages (DSLs), an approach originating in the field of Software Engineering. Van Deursen et al. define the term as follows:

> *"A Domain-Specific Language is a programming language or executable specification language that offers, through appropriate abstractions and notations, expressive power focussed on, and usually restricted to, a particular problem domain."* [51].

DSLs have several compelling benefits. They have been successfully created and applied to boost the productivity of domain-experts and raise the quality of software solutions. For instance, in areas like carving data in digital forensics [48], engineering financial products [50], and controlling lithography machines [45], to name a few. DSLs divide work and separate concerns by offering domain-experts ways to independently evolve and maintain a system's parts. Typically, DSLs raise the abstraction level and incorporate domain-specific terminology that is more recogizable to its users. Powerful language work benches enable analyses, optimizations, visualizations [14], and foreground important trade-offs, e.g., between speed and accuracy in file carving.

Naturally, there are also costs. DSLs are no silver bullet for reducing complexity. Time and effort go into developing the right language with features that are both necessary and sufficient for its users. In addition, a DSL may have steep learning curve and users require training [51]. While DSLs help users maintain products, DSLs themselves also demand maintenance and must evolve to accommodate new requirements, usage scenarios, restrictions and laws, such as new legislation on financial transparency or privacy.

2.4 Need for a Mapping Study

There are many compelling reasons to perform a mapping study on languages of games and play. This study can be approached from different research perspectives with distinct research needs and goals. Here we describe our position and motivation.

We aim to empower game designers with DSLs that automate and speed-up the game design process. We wish to learn how to facilitate the design space exploration and reduce design iteration times. We envision a set of complementary visual languages, techniques and tools that help designers boost their productivity and raise the quality of games and play. Challenges include providing abstractions and affordances for:

(1) expressing a game's parts as source code artifacts, especially interaction-bound game elements, and modifying these at any given moment
(2) evolving 'games and play' by steering changes in the source code towards new gameplay goals prototyping, play testing, balancing, fine-tuning or polishing
(3) obtaining immediate and continuous (live) feedback on a game's quality by continuously play testing the effect of changes on quantified gameplay hypotheses
(4) obtaining feed forward suggestions that focus creative efforts and assist in exploring alternative design decisions in a targeted way
(5) forming better mental models for learning to better predict the outcome on play

To know where to start automating game design, we need an extensive analysis on existing approaches. However, these efforts are currently not mapped, and opportunities and limitations are not yet well understood. As a result it is unclear which game facets are amenable to DSL development, which features can express game designs, and what the limits of formalism are. There is no telling if DSLs can deliver, and how the tradeoff between costs and benefits applies to game development.

We perform this mapping study on languages of games and play to obtain evidence to support our hypotheses in general, and suit our own specific research needs by scouting for opportunities for developing DSLs in particular.

3 METHODOLOGY

A systematic mapping study requires a precise description of its scope, research questions, search queries and databases for accurate and reproducible information extraction, categorization and comparison [26]. We apply the following methodology.

3.1 Scope

Games have been studied from different perspectives. Language-oriented approaches have been proposed by authors who published in separate fields of research using distinct vocabularies. As a result, language-centric solutions, intended for diverse domain experts and novices solve differently scoped problems related to a game's design, development, and applications. We survey the full breadth of related work.

3.2 Research questions

The research questions addressed by this study on languages of games and play are:

- RQ1 What are the research areas and publication venues where authors have published, and what does a map of the field look like?
- RQ2 Which languages have been proposed and how can these solutions be characterized in terms of 1) objectives, scope and problems addressed; 2) language design decisions, structure and notable features; 3) applications, show cases or case studies; and 4) implementation, deployment and availability?
- RQ3 What are similarities and differences between approaches, and common research perspectives sharing similar frames and goals, which languages illustrate them, and what are limitations?
- RQ4 Which developments and trends can be observed in recent work, and what are the challenges and opportunities for future language research and development?

3.3 Sources

We use the meta-repository Google Scholar (GS) to obtain primary sources because it maps repositories in which we expect to find relevant publications. GS includes traditional sources of publications such as the Association of Computing Machinery (ACM), Institute of Electrical and Electronics Engineers (IEEE), Springer and Elsevier. In addition, GS includes less-traditional sources such as games conferences that operate independently, influential books, blog posts, and dissertations. Limiting the search to fewer sources would likely make the study more easily reproducible but also reduce its relevance. A wide search over many sources is necessary for answering our research questions, and GS fits this criterion. Note that we exclusively focus on written sources, which excludes games and commercial development products.

3.4 Queries

Starting with a limited view of the field, we begin with a query to find domain-specific languages for game design and game development. We call this our *narrow query*.

```
"domain specific language" AND ("game development" OR "game design")
```

GS returns approximately 400 results, mainly in the field of software engineering and although many publications seem relevant, few articles focus on game design. Clearly, the narrow query is biased towards one specific research area and is too restricted for answering our research questions.

Part of a mapping study is identifying the distinct vocabularies experts in separate research areas use for describing similar approaches, and a more general term is ‘‘language’’. We widen the scope accordingly but unfortunately we now find many results on subjects that are off-topic. We therefore attempt to filter out irrelevant publications by formulating a *wide query*.

```
language AND ("game development" OR "game design")
    AND NOT ("sign language" OR "second language" OR  "language
        acquisition" OR "body language" OR "game based learning" OR "beer
        game" OR gamification OR gamify)
```

Table 1. Categories of publications

Category	Description
Paper or article	Peer-reviewed papers appearing in the proceedings of a symposium, workshop or conference, or a journal article
Thesis	Describes the content of a Bachelor's, Master's or PhD thesis. May contain chapters based on previously published peer-reviewed work
Textbook	Explains a subject such that it can be easily studied chapter by chapter, typically written by one or more established experts
Non-fiction	Describes topic of interest, not primarily intended to be studied
Technical report	Reports on what are usually technical challenges and solutions
Manual	Explains how to perform steps, use solutions or apply concepts
Blog post	Shares an opinion, problem or technical approach on a web page
Presentation	Introduces or explains challenges on a topic that are noteworthy or inspirational during a keynote presentation or invited talk

GS reports approximately 17.5 thousand results, more than is feasible for us to analyze. We now realize that given the wicked nature of game design and game development, no single query exists that captures all relevant works. We therefore propose a compromise that combines the results of the narrow query with the first 1000 results of the wide query[1]. We restrict the language to English. We exclude patents and citations, results for which GS typically has not seen the full source.

3.5 Inclusion and exclusion criteria

We select publications according to the following criteria. The inclusion criterion is: The publication describes a structured language-oriented approach for solving problems related to the design or the development of digital games. For instance, we include programming languages, modeling languages, DSLs, pattern languages, ontologies and structured vocabularies. Digital games (or digital representations) include computer games, videogames, and applied games (a.k.a. serious games), etc. The exclusion criterion is: Language features with a fixed structure and notation are not described, or the language does not relate directly to games, and as such does not inform the game design process. Therefore, we exclude the mathematics subject of game theory and the general theme of high performance computing. Networking and audio are not excluded a-priori.

4 REVIEW PROTOCOL

We identify and analyze relevant publications and categorize them according to the review protocol described in the following section. Each section of the protocol addresses a research question. Section 4.1 addresses RQ1, Section 4.2 addresses RQ2. The remaining questions RQ3 and RQ4 are outside the scope of this protocol, and are addressed respectively in Section 6 and Section 7.

4.1 Research areas and publication venues

For each included publication we record the available bibliographical information in BibTeX, e.g., about its authors, title, editors, year, publication venue, acronym, publisher, journal, volume, number, ISBN, ISSN and DOI. When records are incomplete or missing, as is often the case, we insert the information by hand.

In addition, we add a mapping study identifier that denotes its rank in the query results, a number or not found (–), and appears in the narrow (n) or wide (w) query results. For instance, 12n indicates the publication ranked 12 in the GS results of the narrow query, and –w indicates a publication that

[1] GS limits the number of results to one thousand, but one can obtain more when filtering by year.

Table 2. Categories of research (adapted from Wieringa et al. [54])

Category	Description
Evaluation research	Investigates a problem or implementation of a technique *in practice* for gaining *empirical* knowledge about causal relationships between phenomena or logical relationships among propositions
Proposal of solution	Proposes a novel solution technique and argues for its relevance without a thorough validation.
Validation research	Investigates properties of a proposed solution that has not yet been used in practice
Philosophical paper	Sketches a new way of looking at a problem, a new conceptual framework, etc.
Opinion paper	Provides an author's opinion about what is wrong or good about a topic of interest
Experience report	Explains steps taken and lessons learned from experiences gained during a project
Tutorial	Explains and demonstrates how something works, usually by means of illustrative examples

conforms to the wide query, but is not ranked in the top one thousand results. In some cases, we include publications for clarity that conform to neither query, stating which keyword is missing, e.g., ~~gd~~ indicates the keywords ‘‘`game design`’’ and ‘‘`game development`’’ are both missing.

Each publication is of a certain type, as shown in Table 1. In addition, we analyze research categories shown in Table 2, like Petersen et al. [37]. This table extends categories proposed by Wieringa et al. for categorizing peer reviewed research in requirements engineering with the last two [54]. These are general categories that indicate how reliable and mature a source is, without going into detail.

We construct a visual map of the field by leveraging citation data that relates publications, and categorize publication venues to research areas. First, we extract citation information from the GS research results. Next, we generate a citation graph whose nodes are publications and edges are citations between them. Finally, we visualize this graph using Gephi, an interactive graph visualization framework[2]. Gephi's force map algorithm draws together publications with citations between them, forming clusters that roughly correspond to research areas.

We count the number of publications in different journals, conferences, workshops and symposia. We identify research areas by grouping venues according to disciplines and shared topics. We briefly describe each area, and zoom in on the related section of the map for illustration. We summarize venues with two or more publications.

4.2 Language analysis and summary

We wish to learn how languages compare, what they have in common, what separates them, and what makes them unique. For each included publication we extract the name of the language, or a description in case no name is provided. We summarize each language concisely by analyzing related publications in a style similar to an annotated bibliography. Table 3 highlight the facets we analyze.

Claims regarding the scope and applications typically refer to game genres, such as First Person Shooter (FPS), Role Playing Game (RPG) or 2D Platform Game. Although game genre qualifiers are course grained, and not suitable for comparing games in detail [2], they do offer authors ways to indicate the topic of the solution and sketch contours of its scope. In addition, we categorize languages objectives using the categories shown in Table 4.

Non-exclusive objectives position languages as: *communication means* for sharing knowledge between experts; *illustration means* for explaining or clarifying problems or solution by example; *maintenance tool* for maintaining and modifying a game's parts over time; *productivity raiser* for

[2]https://gephi.org (visited June 6th 2019)

Table 3. Languages facets to summarize and analyze

Facet	Element	Description
Brief description	Problem	Problem statement, game topic
	Objectives	Goals the authors formulate, challenges addressed
	Solution	Solutions proposed, claims on language application and scope, game genre
	Category	Solution category and application area (Table 4)
Design	Pattern	Language design pattern (Table 5)
	Features	Language features (elements shown in Table 6)
	Examples	Snippets of text, code, diagrams or models
Implementation		How is the language implemented, e.g., interpreter, compiler (these details are usually not described)
Validation	Products	Games, prototypes and show cases that are constructed using the language
Availability	Web site	URL of a web site providing information on the language, notation or toolset
	Distribution	URL of a binary distribution or source code repository
	Source license	License under which the source code is available

Table 4. Language objectives – solution scope, category and application area

Dimension	Category	Description of intent
Scope	Application-specific	Solution is specific for a game or application
	Genre-specific	Solution that is reusable for a specific game genre
	Generic	Generic solution or separated concern
Solution	Framework	Analysis or mental framework for studying, understanding, comparing, categorizing games that does not directly support game development, e.g., ontologies, design patterns, or simulations
	Tool	Authoring tool that facilitates creating a game's parts as models or programs for design or development , e.g., visual environments, programming languages or DSLs
	Engine	Game engine, reusable building block or software library that integrates models fully into game software
Area	Research	Research vehicle primary intended for performing research in a specific area
	Educative	Platform primarily intended for teaching a subject to a group of people or example meant to illustrate, educate or inform
	Practice	Solution primarily intended for practitioners, supporting game design or game development

increasing the productivity of its users; *quality raiser* for improving a game's quality; and *reuse promotor* for making parts of a game's code or design reusable.

We highlight language design decisions and notable features, including mentions of language reuse and formal semantics. When possible, we use the language design patterns for DSLs proposed by Mernik et al. shown in Table 5 in our description [34]. We analyze language features related to notation, elements and user interface described in Table 6. We record how a language is implemented, e.g. as an interpreter or a compiler, and what the host language or formalism is.

Furthermore, we assess applications and availability to form an idea about its status, deployment and maturity. We list notable applications, show cases and case studies that have been used to validate or evaluate the language in practice. Finally, we report which languages are actually

Table 5. Language design patterns (adapted from Mernik et al. [34])

Dimension	Category	Description
Reuse	Piggyback	Partially uses an existing language, a form of exploitation
	Specialization	Restricts an existing language, a form of exploitation
	Extension	Extends an existing language, a form of exploitation
	Invention	Designs a language from scratch without language reuse
Description	Formal	Formally describes a language using an existing semantics definition method such as attribute grammars, rewrite rules, or abstract state machines
	Informal	Informally explains a language without formal methods

Table 6. Language features (these features are not mutually exclusive)

Dimension	Feature	Description
Notation	Textual	The language has a textual notation
	Visual	The language has a visual notation
Elements	Scopes	Scopes and bounds may be used to separate elements and limit their valid context
	Conditionality	Conditionality features enable or disable other language elements or events
	Recurrence	Recurrence features are elements that can happen again, e.g., in iterations
	Modularity	Modularity features enable composition and/or reuse of language elements
	Domain-specific	Domain-specific features may be especially created for a special purpose are unique to the language
User Interface	Feedback	Provides a feedback feature enabling understanding
	Mixed-initiative	Provides feedback & feed-forward, alternating between user input and computer generated alternatives
	Live	Provides immediate and continuous feedback, e.g., a live programming environment

available, and if applicable, we provide links to manuals, teaching materials, source repositories and license agreements. This concludes the protocol. We present the results in the following section.

5 RESEARCH AREAS

Here we present a systematic map of languages of games and play. We performed a search on GS with our narrow and wide queries between the 2nd and 8th of March 2018, and obtained citation data until March 21st. Figure 2 shows a year-by-year count of papers included in this study from both queries and publications we added by hand. Figure 3 shows the citation graph of publications in the search results. Each dot with text represents a publication shown with first author name and year. Publications are included (green) or excluded (red) by applying the criteria from the search protocol. Publications not connected to the graph are omitted in this figure.

The languages of games and play we have identified originate from the fields of software engineering, artificial intelligence, humanities, social sciences, education, and game studies, with some cross-disciplinary overlap and diffuse areas. Figure 4 illustrates the diversity of publication areas and topics we have selected. The reader is invited to spin the outer wheel of research topics around the publication areas. We describe research areas one by one. We briefly introduce each area, give an overview of venues and link related research perspectives, which are detailed in Section 6. For conciseness, we only describe venues when the number of identified publications is at least two, as specified by the search protocol.

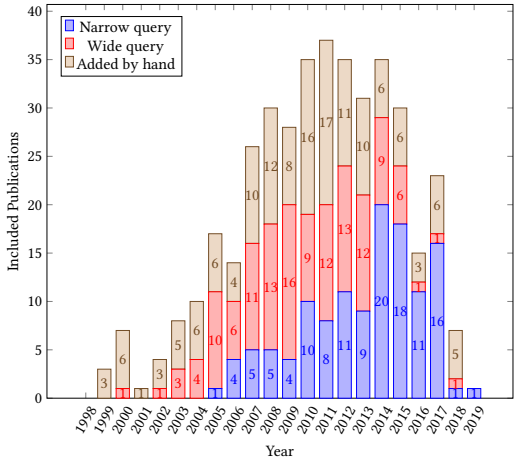


Fig. 2. Amount of publications included in the study per year

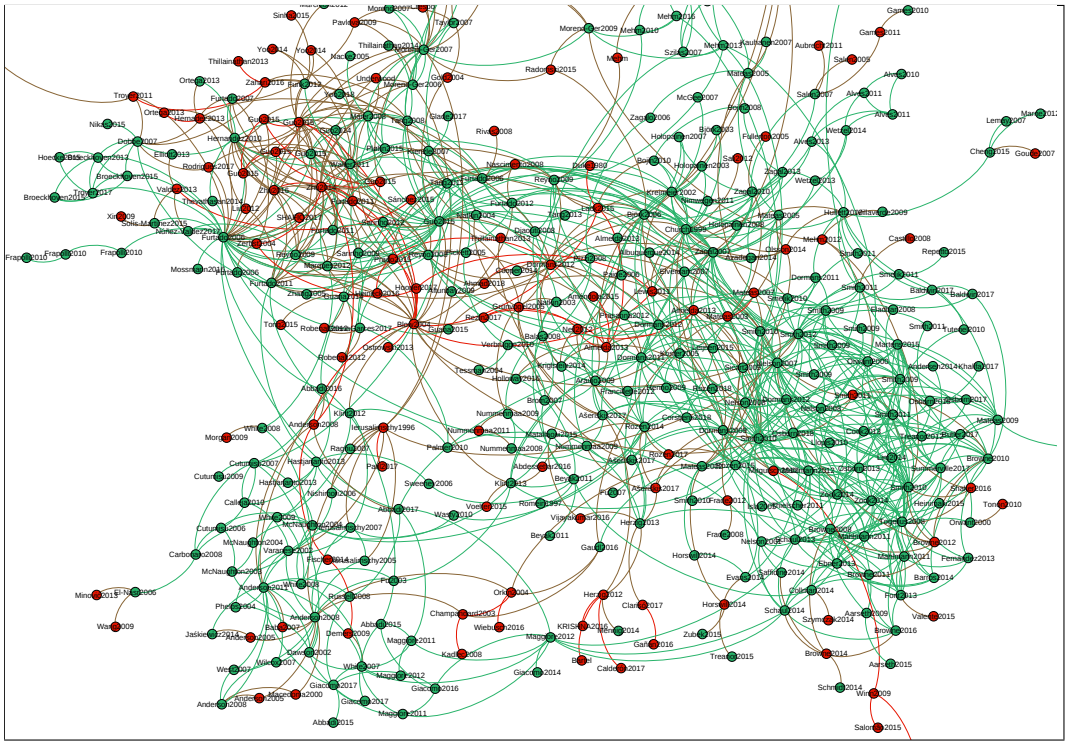


Fig. 3. Systematic map shown as a citation graph. On the left: Software Engineering, Modelware (mainly the top half) and Grammarware (mainly the bottom half). On the right: Analysis frameworks, pattern languages, ontologies (mainly the top half) and AI in Games (mainly the bottom half).

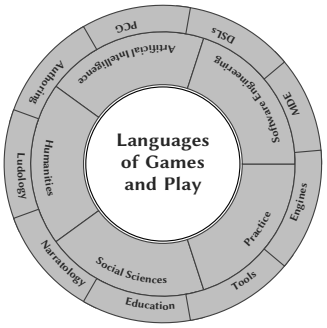


Fig. 4. Language-centric approaches crosscut areas, disciplines and topics

Table 7. Publication venues in the field of Software Engineering and Programming Languages

Venue	Acronym	Years	Ct.
International Conference on Systems, Programming, Languages, and Applications: Software for Humanity (conference umbrella)	SPLASH	1986–	
Conference on Object-Oriented Programming Systems, Languages and Applications	OOPSLA	1986–	1
Symposium on New Ideas in Programming and Reflections on Software	Onward!	2002–	2
International Conference on Generative Programming and Component Engineering	GPCE	2002–	1
International Conference on Software Language Engineering	SLE	2008–	3
Workshop on Domain-Specific Modeling	DSM	2001–	4
International Conference on Software Engineering	ICSE	1975–	2
Workshop on Games and Software Engineering	GAS	2011–2016	2
International Conference on Automated Software Engineering	ASE	1990–	2
Symposium on Principles of Programming Languages	POPL	1973–	2
Science of Computer Programming		2000–	2
ACM Sigplan Notices		1966–	2
Communications of the ACM	CACM	1958–	3
ACM Queue	Queue	2003–	2

5.1 Software Engineering and Programming Languages

Software Engineering (SE) researchers study the game domain by developing and applying structured methods, languages, techniques and tools for engineering better game software. Lämmel covers several subjects of Software Language Engineering in his textbook on Software Languages: Syntax, Semantics, and Metaprogramming [29]. Compilers: Principles Techniques and Tools (a.ka. the *"dragon book"*) by Aho et al., first published in 1986, is still regarded as a classic foundational textbook [3].

We identify contributions from Programming Language (PL) research in particular, as shown in Table 7. Figure 3 shows related publications on the left. The ACM Special Interest Group on Programming Languages (SIGPLAN) *"explores programming language concepts and tools, focusing on design, implementation, practice, and theory"*.

The main source of publications is the International Conference on Systems, Programming, Languages and Applications: Software for Humanity (SPLASH), a large conference 'umbrella' of

Table 8. Publication venues in the field of Artificial Intelligence and Games

Venue	Acronym	Years	Ct.
AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment	AIIDE	2005–	14
Workshop on Experimental AI in Games	EXAG	2014–	2
IEEE Conference on Computational Intelligence and Games	CIG	2005–2018	8
IEEE Conference on Games	CoG	2019–	–
International Conference on Computational Creativity	ICCC	2010–	2
EvoStar – The Leading European Event on Bio-Inspired Computation (conference umbrella)	EvoStar	1998–	
International Conference on the Applications of Evolutionary Computation – Games track (EvoGames)	Evo-Applications	2010–	3
IEEE Transactions on Computational Intelligence and AI in Games	TCIAIG	2009–2017	12
IEEE Transactions on Games	T-G	2017–	–

colocated events, which includes: 1) Workshop on Domain-Specific Modeling (DSM); 2) International Conference on Software Language Engineering (SLE); 3) Conference Object-Oriented Programming Systems, Languages and Applications (which is a track of SPLASH since 2010) (OOPSLA); 4) Symposium on New Ideas in Programming and Reflections on Software (Onward!); and 5) International Conference on Generative Programming and Component Engineering (GPCE). SLE research is traditionally split between modelware and grammarware, which respectively revolve around meta-models and grammars [36].

In addition, the search revealed two publications at the International Conference on Software Engineering (ICSE). and two more at the colocated workshop on Games and Software Engineering (GAS), which was organized five times.

Other conferences include the International Conference on Automated Software Engineering (ASE), and the International Conference on Model Driven Engineering Languages and Systems (MoDELS). In addition, we find two invited talks at the Symposium on Principles of Programming languages (POPL) intended to inspire PL research.

Several journals stand out. The monthly SIGPLAN Notices includes special issues from associated conferences, including SPLASH, SLE, Onward!, GPCE and POPL Communications of the ACM is a journal that covers a wider computer science space and articles from ACM Queue are included in its practitioners section In addition, we find two publications in special issues of Elsevier's Science of Computer Programming

We highlight the following related perspectives: Automated Game design, a multi-disciplinary area that includes SE, in Section 7.3; Applied (or serious) game design, in particular DSLs for expressing subject matter, in Section 6.3, Script- and Programming Languages for game development, in Section 6.12; Modeling Languages and model-driven engineering for game development, in Section 6.13; and Metaprogramming, primarily illustrative examples explaining the power of generic language technology, in Section 6.14.

5.2 Artificial Intelligence and Games

The Artificial Intelligence (AI) community has studied how games can benefit from intelligent, usually algorithmic approaches, yielding efficient algorithms, techniques and tools. In their textbook on AI and Games, Yannakakis and Togelius describe the theory, use and application of algorithms and techniques [56].

Table 9. Articles and presentations from Game Development Practice

Venue	Acronym	Years	Ct.
Game Developers Conference (UBM Technology Group)	GDC	1988–	4
Game Developer Magazine (UBM Technology Group)		1994–2013	3
Gamasutra (UBM Technology Group)		1997–	3

When languages are created, it is often in the context of intelligent systems (or expert systems), or content generators. Classical AI favored logic programming in Prolog for knowledge engineering, the construction of intelligent systems, which explains why some modern solutions are also based on this paradigm. Notable approaches include logic (Prolog, Answer Set Programming) and Machine Learning. Figure 3 shows related publications, mainly clustered together in the bottom half of the graph. Conferences, symposia and workshops include the following.

The Association for the Advancement of Artificial Intelligence (AAAI) *"aims to promote research in, and responsible use of, artificial intelligence"*. This includes the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE) and colocated workshops, such as Experimental AI in Games (EXAG) which are annually organized in North America.

IEEE Conference on Computational Intelligence and Games (CIG) covers *"advances in and applications of machine learning, search, reasoning, optimization and other CI/AI methods related to various types of games."* We find contributions related to methods for general game playing such as game description languages and generation of level, strategies and game rules. The IEEE Conference on Games (CoG), evolved from CIG and wided the scope to cover among other topics, game technology, game design, and game education.

The prime journal is IEEE Transactions on Computational Intelligence and AI in Games (TCIAIG) It recently widened its technological scope and was renamed to Transactions on Games (T-G).

We highlight the following related perspectives: Automated Game design, a multi-disciplinary research topic with many contributions from AI and games, in Section 7.3; Game mechanics, frameworks and systems providing analysis, generation and explanations of a game's rules, in Section 6.4; Virtual worlds and game levels, spaces whose structure and composition can be described by languages, tools and generative techniques, in Section 6.5; Behavior languages, the design of algorithms, tools and engines for non-player character behavior, in Section 6.6; Narratives and storytelling, in particular technical language-centric approaches, in Section 6.7; Game analytics and metrics, in Section 6.8; and General gameplaying and Game Description Languages, formalisms for a diverse test-bed for AI and general game playing, in Section 6.11.

5.3 Game Development Practice

The Game Developers Conference (GDC) owned by the UBM Technology Group is a large annual event and not a publishing venue. However, it hosts presentations by practitioners, some of whom share presentation slides identified by this study The GDCVault[3] contains audio and video recordings of these presentations and all volumes of Game Developer Magazine[4], which ran until 2013 Gamasutra[5] is a web site that continues to publish articles, and hosts selected articles from Game Developer Magazine In particular, post-mortems in which developers share experiences about challenges, solutions, decisions (the good and the bad) during a game's life cycle offer glimpses

[3]https://gdcvault.com (visited August 20th 2019)
[4]https://gdcvault.com/gdmag (visited August 20th 2019)
[5]http://www.gamasutra.com (visited August 20th 2019)

Table 10. Multi-disciplinary publication venues on Game Studies, Education and Storytelling

Venue	Acronym	Years	Ct.
Conference of the International Simulation and Gaming Association	ISAGA	1970–	3
Simulation & Gaming	S&G	1970–	2
Game Studies		2001–	2
European Conference on Games Based Learning	ECGBL	2007–	2
Computers & Education (Elsevier)		2000–	2
International Conference on Interactive Digital Storytelling	ICIDS	2008–	2
Conference on Technologies for Interactive Digital Storytelling and Entertainment	TIDSE	2003, 4, 6	2
International Conference on Virtual Storytelling	ICVS	2001, 3, 5, 7	2

of game development practice [19]. Given the enormous size of the game industry, the practical accounts we identified are few. These languages likely represent the tip of the iceberg. We reflect on this issue in Section 9.

5.4 Social Sciences and Humanities and Storytelling

Game scholars have extensively studied, analyzed and critiqued games. They provide insight into how games work, what constitutes play, and how games impact culture and society. In game studies, ludologists have proposed vocabularies, ontologies and pattern languages aimed at understanding and critiquing games and play in a cultural context.

Aside from studies, game education has yielded textbooks on game design and development practice. Schell introduces the art of game design, offering theory, approaches and conceptual lenses that help designers think and practice [39]. Fullerton describes a play centric approach to creating games highlighting the disciplines prototyping and play testing [17]. Salen and Zimmerman describe game design fundamentals with focus on core concepts, rules, play and culture [38].

The International Conference on Interactive Digital Storytelling is the result of merging between its predecessors Technologies for Interactive Digital Storytelling and Entertainment and Conference on Virtual Storytelling We also mention the workshops on Games and Natural Language Processing and Intelligent Narrative Technologies

The International Simulation and Gaming Association (ISAGA) has organized an annual conference since 1970 ISAGA can trace its origins back to the now famous book Homo Ludens [23], and aside from game studies also has an education focus. Related is the journal Games & Simulation . In addition, we identify articles in the Journal of Game studies .

We highlight the following related perspectives: Ontological approaches and typologies, in Section 6.1; Pattern languages and design patterns, in Section 6.2; Narratives and storytelling, in Section 6.7; Educative languages, in Section 6.9; and Gamification, in Section 6.10.

5.5 Multi-disciplinary Games Research

Researchers and practitioners from different fields meet at multi-disciplinary conferences on games and entertainment computing. They exchange points of view, and apply and mix diverse expertise on game design, AI and language technology, which results in synergy and inter-disciplinary advances. Although challenges, objectives approaches may differ, multi-disciplinary venues are the main source of publications included in this study, and perhaps the best source of nuanced approaches. Table 11 shows the publications venues we identified. We briefly describe each one.

Table 11. Multi-disciplinary publication venues on Games and Entertainment Computing

Venue	Acronym	Years	Ct.
International Conference on the Foundations of Digital Games	FDG	2006–	12
Workshop on Procedural Content Generation in Games	PCG	2010–	9
Workshop on Design Patterns in Games	DPG	2012–2015	5
International Conference on Entertainment Computing	ICEC	2002–	5
Workshop on Game Development and Model-Driven Software Development	GD & MDSD	2011, 2012	3
Digital Games Research Association International Conference	DiGRA	2003–	7
International Conference on Intelligent Games and Simulation	GAME-ON	2000–	6
International North American Conference on Intelligent Games and Simulation	GAME-ON-NA	2005–2011+	3
International Conference on Computer Games	CGAMES	2004–2015	4
Brazilian Symposium on Computer Games and Digital Entertainment	SBGames	2002–	4
International Conference on Computers and Games	CG	1998–	2
International Conference on Advances in Computer Entertainment Technology	ACE	2004–2018	5
International Academic Conference on the Future of Game Design and Technology	Future Play	2002–2010	3
ACM Computers in Entertainment	CIE	2003–2018	2

The International Conference on the Foundations of Digital Games (FDG) is a multi-disciplinary conference that alternates between Europe and the USA. Two colocated workshops are of special note. The first, Design Patterns in Games (DPG) includes pattern languages and gameplay design patterns. DPG was last organized in 2015, but might be revived. The second, Procedural Content Generation in Games (PCG) is concerned with generative methods for games, often using AI techniques.

The International Conference on Entertainment Computing (ICEC) is organized annually. The continent of the venue varies. The colocated Workshop on Game Development and Model-Driven Software Development (GD&MDSD) was organized twice. The Digital Games Research Association (DiGRA) organizes its annual DiGRA International Conference, which is a mix of game studies, humanities and technology.

The International Conference on Intelligent Games and Simulation (GAMEON-ON) focuses on structured methods for programming of games that benefit industry and academia. GAME-ON is organized annually in Europe. but also occurs on different continents, e.g., North America (GAME-ON-NA). Another conference branched off from GAME-ON in 2004, as explained by Spronck [43]. It was first known as Computer Games: Conference on Artificial Intelligence, Design and Education (CGAIDE) and later became the International Conference on Computer Games (CGames), which was last organized in 2015.

The International Conference on Computers and Games (CG) is a venue that is not organized every year. The Brazilian Symposium on Computer Games and Digital Entertainment (SBGames) is a national event with international visibility, and also a source of several DSLs.

Some venues we identified are discontinued. For instance, the Conference on Advances in Computer Entertainment Technology (ACE) was a technology oriented multi-disciplinary conference. In 2018, most members of its steering committee resigned when they could no longer guarantee an impartial review process, and the conference closed down after a community boycot that condemned the conduct of the event's owner. The International Academic Conference on the Future of

Table 12. Ontological Languages

Nr.	Language	Domain	Notation
1	Game Typology	Game studies	tables and visual diagrams
2	Game Ontology Project	Game studies	tables /pattern-language
4	Pervasive Games Ontology	Pervasive games	class diagram
3	Ontology of Journalism	Journalism	Web Ontology Language

Game Design and Technology (Future Play) was organized from 2002 until 2010. The journal, ACM Computers in Entertainment (CIE) ceased in 2018.

In the next section, we discuss a series of research perspectives. Each of these can be considered a multi-disciplinary point of view.

6 RESEARCH PERSPECTIVES

We present a series of fourteen complementary research perspectives on languages of games and play. Each perspective sheds light on what informs the design and construction of good games. Together they form an overview that provides answers to our research questions RQ2 and RQ3. We acknowledge that different decompositions would have been possible, and ours is merely one of many ways to aggregate the results of this study. Our structure represents a best-effort that adheres to the phrasings and research frames of the authors whose works we summarize.

We derive the perspectives from the search results as follows. We group summaries of similar languages. We distill common research frames, highlight challenges and describe theoretical foundations, systematic techniques and practical solutions. For conciseness, we only illustrate these views with a selection of representative language summaries.

6.1 Ontologies and Typologies

In search of a common vocabulary for game studies, scholars have defined ontologies that relate written symbols, abstract concepts and real-world objects. In general, an ontology defines a set of named concepts, their properties and the relations between them, usually with the goal of categorizing objects of interest in a specific subject area or problem domain for understanding its meaning. In game studies in particular, ontologies and typologies are used to describe, categorize, analyze, understand and critique games and play.

Aarseth explains that ontologies are essential for game studies to reach a consensus on the meaning of words, concepts and relations between them [1]. He scrutinizes different ontological models and describes challenges in their construction. For instance, choosing the level of description, i.e. how fine-grained the ontological elements (or meta-categories) should be, is difficult because there is no natural halting point in adding nuances, details or patterns for highlighting differences. In addition, choosing generality or specificity limits applications since general-purpose ontologies may not be as useful as ontologies constructed for a specific purpose.

There are many languages for constructing ontologies. For instance the W3C Web Ontology Language (OWL) is an XML based notation with good tool support for describing semantic relationships[6]. In some cases languages of games and play use ontologies for describing or guiding parts of the solution. Table 12 lists ontological languages we describe.

[6]https://www.w3.org/OWL (Visited June 1st 2019)

Table 13. Pattern-languages for analyzing games and forming gameplay hypotheses

Nr.	Language	Informs the design of
5	Formal Abstract Design Tools	gameplay goals / pattern languages
6	Game Design Patterns	gameplay goals / pattern languages
7	Gameplay Design Pattern	gameplay goals
8	Mechanics Dynamic Aesthetics	digital game systems
11	Collaboration Patterns	collaborative gameplay goals
15	Flow Experience Patterns	flow experience goals
16	Patterns Language for Serious Games Design	applied gameplay goals

Table 14. Pattern-languages offering authorial affordances over designing games and play

Nr.	Language	Game Artifact	Affects
9	Pattern Language for Sound Design	sound	sound-supported experiences
10	Verbs	abstract player actions	gameplay affordances
12	Pattern Cards for Mixed Reality Games	mixed reality game rules	mixed-reality experience
13	Design Patterns for FPS Games	structure of FPS levels	progression, experiences
14	3D Level Patterns	structure of 3D levels	progression, experiences
17	Operational Logics	depends on concrete representations	inner game workings, external player communication
26	Machinations	game-economic mechanics, feed-back loops	gameplay affordances, strategies and trade-offs

6.2 Pattern Languages and Design Patterns

A pattern language describes best practices with empirically proven good results as a reusable solution to commonly recurring problems in a particular area of interest. The approach originates form Alexander et al. who describe a pattern language for towns, buildings and construction [4]. Often presented in table form or a template, sequential sections highlight different facets of the problem, proposed solution, examples, and related contextual information. In Software Engineering, Object Oriented (OO) design patterns are a well-known means to create, explain and understand software designs, design decisions and implementations [18]. In contrast, in game studies and humanities, game design patterns are a means to analyze and explain player experiences, also referred to as gameplay design patterns. The key difference between these kinds of patterns is that the former are prescriptive for structuring software and the latter are analytic regarding gameplay effects. We categorize pattern languages in two categories. The first, shown in Table 13, lists languages for analyzing, categorizing, understanding and critiquing gameplay. The second, shown in Table 14, lists languages for predicting the effect of changes to a game's design, and making informed design decisions, e.g., level design patterns and game mechanics patterns. Programming patterns that directly affect a game's mechanics, narratives, levels, and behaviors are discussed in Sections 6.4, 6.5, 6.6 and 6.7.

6.3 Applied Game Design

Applied (or serious) games have a primary purpose other than entertainment and usually require designs that incorporate subject matter knowledge. Diverse experts from education, psychology and even medical doctors can help imporve game designs, e.g., for learning[7], overcoming traumas and speeding-up recovery. The challenge is integrating domain knowledge in a game's design to

[7]Please note that we excluded the term *'game based learning'* from the wide query due to the large amount of false positives.

Table 15. Languages for domain-experts to help design applied games scenarios

Nr.	Language	Subject matter expert	Objectives
18	StoryTec	educator, non-programmer	educate through stories that integrate learning goals
19	GameDNA	psychologist	assess cognitive processes
20	ATTAC-L	pedagogical expert	educate, prevent cyber bullying
21	EngAGe DSL	educator	improve feedback to learners
22	VR-MED	medical expert	teach family medicine

Table 16. Game Mechanics Languages

Nr.	Language	Mechanics	Analysis	Generation
23	Petri Nets	game-economic, story	yes	?
24	Game Space Definitions	'stock' logics	yes	
25	Biped and Ludocore	combinatorial	yes	yes
26	Machinations	game-economic	yes	no
27	Micro-Machinations	game-economic	yes	yes
28	Game-o-Matic	combinatorial	yes	yes
29	Mechanic Miner	avatar-centric	yes	yes
30	Gamelan and Modular Critics	combinatorial	yes	?
31	PDDL Mechanics	combinatorial, avatar-centric	yes	yes
32	Sygnus and Gemini	combinatorial	yes	yes

achieve specific gameplay goals such that players (for instance patients or students) learn, reflect or modify behaviors. Naturally, there are ethical and privacy implications and restrictions of studying player choices, especially if the game also serves as a diagnostic tool. Dörner et al. provide an overview of foundations, concepts and practice of serious games for prospective developers and users [12]. Here, we identify languages, mainly DSLs, intended for helping domain-experts design and vary scenarios of applied games, e.g., for learning and assessment, as shown in Table 15.

6.4 Game Mechanics

Although most agree that *game mechanics* are rules that affect gameplay, there are many different explanations, theories on how this works, e.g., [16, 25, 38, 39, 41]. In the previous sections we have described frameworks (Section 6.1) and design patterns (Section 6.2) for understanding, analyzing, and creating game mechanics. This section can be regarded as a *proceduralist view* that applies formalizations and generative techniques to program game mechanics, and analyze effects.

Table 16 shows languages, generators and tools for game mechanics. These works explore the limits of formalism, and study to what extent models of mechanics (and players) can be leveraged for predicting and improving a game's quality. Each language attempts to relate mechanics to aesthetics in different ways. Combinatorial rule spaces expressed with logical notations use constraints for exploring the design space and homing in on desired qualities, for instance using Answer Set-Programming for exploring the design space. Examples include Biped and Ludocore (Language 25) or Gamelan and Modular Critics (Language 30).

Meaningful relationships between real-world subjects, e.g., derived from WordNet and ConceptNet, can be used to instantiate the structure of mechanics, e.g. Game Space Defintions (Language 24). Using Rhetorical arguments intended for adding meaning structure the mechanics of news games or micro-games, e.g., to convince players with political or cultural statements. Examples are Game-o-Matic (Language 28), and Sygnus and Gemini (Language 32). Sicart critiques *procedural rhetorics*, and

Table 17. Tools and Languages for Mixed-Initiative Design of Levels and Virtual Worlds

Nr.	Language or Tool	Content
33	Semantic Scene Description Language	classes of concepts and relationships
34	SketchaWorld	3D worlds
35	Tanagra	platform game levels
36	Ludoscope	grammar-based transformation pipelines of 2D levels
37	The Sentient Sketchbook	tile map sketches of 2D levels
38	Evolutionary Dungeon Designer	2D level maps and patterns

presents opposing arguments [42]. Nelson clarifies a more general position on proceduralism [35], and Treanor and Mateas present an account of proceduralist meaning [47].

Avatar-centric mechanics encoded in ASP, rewrite rules or Java describe the physics of characters in 2D levels, such as moving, jumping and bouncing, e.g., Mechanic Miner (Language 29), Planning Domain Description Language Mechanics (Language 31) and PuzzleScript (Language 91).

Game-economic mechanics described in graph notations express how in-game resources flow and which choices players have. They foreground feed-back loops that represent investments and trade-offs. Examples are the well-known Petri Nets (Language 23), the design framework Machinations (Language 26) or its cousin the programming language Micro-Machinations (Language 27).

6.5 Virtual Worlds and Levels

Virtual worlds are spaces in games or simulations that through various integrated audiovisual content and interactive mechanisms support exploration, communication or play. Game worlds and levels can be populated by and player avatars, virtual characters, stories, missions, quests, etc. Procedural Level Generation is a form of PCG that focuses on generating game levels, spaces that integrate missions, quests and (lock and key) puzzles, e.g., for dungeon crawlers and platformers. Van der Linden et al. survey Procedural Dungeon Generation [49]. We identify relatively few language-centric approaches using our protocol, since most authors in this perspective call their solutions 'tools'.

Table 17 shows tools and languages for designing levels and virtual worlds. Each of these tools support creating and improving (this is usually called authoring) content a mixed-initiative style. In *mixed-initiative* approaches, intelligent services (the tool) and the designer collaborate and take turns to achieve the designer's goals [22]. Typically, designers receive visual computer-generated suggestions, and based on gradually improving insight, make decisions that refine the content iteration by iteration, in a conversational style. However, due to a lack of direct manipulation of generated content, it can be challenging to assure all possible results represent meaningful and high quality content. In this context, *semantic scenes* are structures for describing meaningful and consistent content that can guide generators. As in other perspectives, authors also apply patterns, constraints and metrics for generating and analyzing level qualities. Metrics are discussed separately in Section 6.8.

6.6 Behaviors

Defining behaviors of Non Player Characters (NPCs) has been an active topic of technically oriented research. A Behavior Definition Language (BDL) is a programming language that offers a notation that controls powerful AI features for describing believable virtual entities that inhabit game worlds [8]. Many BDLs are implemented as reusable software libraries that complement game engines. According to Anderson, many BDLs are DSLs that also maintain the flexibility of programming languages [8]. Challenges include developing appropriate notations and features,

Table 18. Languages for domain-experts to help design behaviors

Nr.	Language	Description
39	ABL	ABL is a reactive planning language for authoring believable agents with rich personality
40	SEAL	Simple Entity Annotation Language (SEAL) is A C-like script language for describing NPC behaviors
41	BEcool	BEcool is a visual graph language with sensors and actuators for describing expressive virtual agents
42	Behavior Trees	Behavior Trees is a visual language for authoring AI behaviors
43	BTNs	Behavior Transition Networks is a visual notation of hierarchical state machines for describing behaviors
44	POSH#	framework for creating behavior-based AI for robust and intuitive agent development
94	Statecharts	Modeling formalism for describing behaviors
103	RAIL	Reactive AI Language (RAIL) is a metamodel-based DSL for modeling behaviors in adventure games

authoring for dramatic realism, improving scalability of parallel behaviors, and raising the fault tolerance. Table 18 shows a limited selection of formalisms including Reactive Planning Languages, Behavior Trees, (Hierarchical) Finite State Machines, and Statecharts. Of course, many languages describe behaviors in one way or another. Our selection represents a wider set of languages.

6.7 Narratives and Storytelling

Storytelling, a field on its own, is concerned with writing, telling and sharing stories by means of narratives to convey ideas and experiences to an intended audience. Similar to games, stories in a cultural context, can be used for entertainment, education, cultural values, etc. The perspective we describe here is a technical interpretation that envisions programming languages for creating narratives and programming interactive stories using generative techniques. Here, we refer to the creation process as *authoring*, since the users of these languages are first and foremost authors.

Various researchers have focused on Interactive Fiction (IF), interactive drama, and the story components of games, e.g., quests, missions and stories of adventure games or educational games. Branching narratives can be expressed as graphs, where choices represent alternative sequences of events or paths. Moreover, emergent stories with generative and dramatic components requires integrating social values and knowledge of virtual personas. Challenges include checking the correctness of the paths by analyzing constraints and providing insight with appropriate visualizations and debugging facilities, e.g, into the causality of emergent scenarios. Storyboards are a sequences of images (or illustrations) and text (e.g. dialogues) used for analyzing stories of various kinds of media, including games. Kybartas and Bidarra survey story generation techniques [28]. We identify several languages intended for authoring, generating and analyzing narratives, summarized in Table 19.

6.8 Analytics and Metrics

Data science combines techniques, approaches and tools from statistics, data mining, data analysis and machine learning. Data scientists leverage the available data to extract knowledge, gain insights and predict trends. The game industry increasingly relies on game analytics for developing high quality games. One technique is using *metrics*, algorithms quantifying system properties, as measures of quality. Designers can use these metrics to test gameplay hypotheses and assess the gameplay quality by studying how metrics evolve over time. For instance, by relating player models or 'personas' to how game mechanics are used (Language 55). PlaySpecs can be used to analyze sequences of player actions (Language 57). Launchpad uses metrics to assess platform level

Table 19. Languages for authoring, analyzing and generating narratives, stories and dramas

Nr.	Language	Expresses
39	ABL	Reactive planning language used for interactive drama
46	<e-Game>	Storyboards for educational adventure games, formally analyzed
47	ScriptEase	Patterns of behaviors, quests and stories, interactive user interface
18	StoryTec	Educational stories and related learning goals, part of a tool set
48	Ceptre	Story worlds and experimental game mechanics, offers logical proofs
49	SAGA	Stories whose compilers target different platforms
50	(P)NFG	Structured computer narratives and IF, playability and correctness
51	Wander	Number games, represents an early historical account
52	Storyboards	Generating storyboards of game levels, leveraging a planner
53	Versu	Interactive text-based drama, including social conventions
54	Tracery	Stories and art, example of a 'casual creator'
107	Ficticious	Demonstrates the use of DSLs for Interactive Fiction

college students		Squeak (Lang. 62) StarlogoTNG (Lang. 64)	
high school students			
middle school schildren	Scratch (Lang. 63) Alice (Lang. 59)		AgentSheets and AgentCubes (Lang. 65) Gamestar Mechanic (Lang. 60)
young children	Kodu (Lang. 61)		
	computational thinking	**programming**	**creating & designing games**

Fig. 5. Relating languages to education levels and learning goals

quality (Language 56). In contrast MAD and SAnR work directly on the engine source code of grammar-based level generators, relating gameplay- and software quality (Language 58). Fenton and Bieman describe a rigorous approach for software metrics based on measurement theory [15]. Research challenges include evaluating the quality of content generators [40], identifying suitable metrics for different types of content, and relating metrics to player models and experience.

6.9 Education

Here we describe educative languages that are end-user solutions aimed at improving learning experiences, e.g., for helping students or children learn programming, computational thinking and game design in a playful and explorative manner. Figure 5 shows examples, and roughly relates languages to educational activities, goals and (minimum) education levels. Educational languages usually come with an ample amount of study material. In addition, these languages may include learning programmes and web sites that cater for active online communities.

Languages for learning usually pay extra attention to usability. Of special note are the "block-based languages" that enable users to fit syntactic constructs together like puzzle pieces. These do not permit syntax mistakes that can be especially frustrating to novices, and instead ensure every adjustment is meaningful and educational.

In addition, several languages use a Logo-style positional movement where one can imagine moving around. Logo is an educational programming language that is well-known for its 'turtle', which can be steered using commands for drawing vector graphics.

Table 20. Game Description Languages (GDLs) for General Game Playing

Nr.	Language	Game domain	Examples of represented games
69	Multigame	mainly board games	Chess, Checkers
70	(Stanford) GDL	combinatorial games	various combinatorial games
71	'Rule Sets'	pac-man-like games	generated games
72	Ludi GDL	combinatorial games	e.g. Javalath
73	Strategy GDL	strategy games	Rock Paper Scissors, Dune II
74	Card GDL	card games	Texas hold'em, Blakcjack, Uno
75	Video GDL	video games	a variety of classic 2D games
76	Recycle	card games	Agram, Pairs, War

6.10 Gamification

Gamification aims to apply or retrofit standard game designs to a new or existing system to improve user experiences[8]. For instance, score, competition and reward systems have been used in areas like online marketing to stimulate participation with a product or service. We identify languages intended to gamify information systems in general, e.g., GAML (Lang. 66), GLiSMo (Lang. 67) and UAREI (Lang. 68). While these languages are technically reusable, they lack subject matter concepts that help domain experts solve design problems. Recently the term *playification* has been used to describe gamification that facilitates play more effectively by means of tailor-made game designs.

6.11 General Game Playing

A key research challenge in AI is developing generally applicable intelligent techniques capable of solving a wide range of complex problems. General game playing, for instance, refers to algorithms that can play many games well.

Game Description Languages (GDLs) are notations with expressive power over restricted game domains intended for evaluating the performance of AI techniques called *general game players* against a wide variety of manually created or generated games. GDLS are meant to cover a varied and representative game space.

The systems that execute GDL programs are used for validating if these techniques are indeed more widely applicable, e.g., by letting them compete. General game players can be search-based [46], e.g., Monte-Carlo Tree Search (MCTS) or leverage Machine Learning, e.g., genetic algorithms or neural networks. In a guest editorial of a special issue on "general games", Browne et al. summarize the state-of-the art, challenges and directions for future research [10]. We show representative GDLs in Table 20 whose domains reflect a shifting research interest of the community over the years. Notably not identified by this study is Zillions of Games[9].

General game playing has the advantage of developing and applying the state-of-the-art AI to digital games, offering advanced tools, simulations and analyses. GDLs and are not necessarily suited for fine-tuning rules and improving gameplay as in automated game design Section 7.3. Therefore, not all GDLs are a-priori well-suited for developing games, especially not those intended as research platforms. For instance, the language features of VGDL are course grained and the Stanford GDL is low-level and verbose. GDLs for restricted game domains, such as board- and card games, represent a concise and expressive middle ground.

[8]Please note that we excluded the term `'gamification'` from the wide query due to the large amount of false positives.
[9]http://www.zillions-of-games.com (visited April 3rd 2019)

Table 21. Generic script and programming languages applied to game development

Nr.	Language	Application domain
62	Squeak	programming system based on Smalltalk applied in teaching game design
77	Python	programming language, applied for scripting in games
78	Lua	programming language, scripting in games
79	vision on game programming	reflections on features of game programming languages and Haskell
80	DisCo	language and system for creating, executing and analyzing formal specifications
81	Design by contract	generic approach that uses pre- and post-conditions for checking function calls

6.12 Script and Programming

Here we describe a programming language perspective on game development. For creating a programming language, one usually constructs a *grammar* using the Extended Backus-Naur form (EBNF) or a similar notation, for parsing the textual source code of programs [3]. Here, source code (or textual model) refers to end-user notations called *concrete syntax*. The result of parsing is a parse tree that is often represented using suitable intermediate representations referred to as *abstract syntax*, which usually omits white-space and comment. In the resulting tree, references between definitions and uses must be resolved. This processes of *reference resolution* yields a graph that forms an input to analyzers, code generators and interpreters that further transform or run programs.

Game programming usually follows a bottom-up approach that composes game systems from reusable parts. Specialized software libraries called *game engines* offer developers reusable Application Programming Interfaces (APIs) for solving challenge in 3D modeling, physics, directional audio or networking. Many commercial game engines have been developed as reusable platforms for game development, e.g., Unity 3D, Unreal, CryoEngine, etc.[10].

One approach separates game engines from game-specific source code by using a generic interpreters as-is, e.g. Python (Language 77) or Lua (Language 78). Table 21 also shows other examples. Another approach leverages general purpose languages by adding domain-specific language extensions, e.g., as an *internal DSL* that reuses the syntax and semantics of the host language. For instance, Scalable Games Language (Language 85) extends SQL and 4Blocks (Language 87) and Sound Scene DSLs (Language 89) are Haskell-based.

In contrast, purpose-built languages, also known as *external DSLs*, have separate parsers, compilers and/or interpreters. PuzzleScript (Language 91) and Micro-Machinations (Language 27) are examples of external DSLs. Table 22 shows examples of DSLs for game development. For conciseness, we do not list all textual DSLs that we already describe in other sections. Notably not identified is DarkBasic [20].

6.13 Model-Driven Engineering

Model-driven game development revolves around abstract *models* that describe game content or work processes, often using visual diagrammatic representations. Modeling languages are based on the principles, techniques and tools from an area called Model-Driven Engineering (MDE). These models are step-by-step translated, transformed and combined into a resulting model or source code that integrates with the game software. We identify applications of generic modeling languages in Table 23, and Domain-Specific Modeling Languages in Table 24.

Metamodeling represents the abstract syntax of models as a graph of Unified Modeling Language (UML) classes and references between them. Because metamodeling is often based on the Eclipse

[10]These are not identified by this study.

Table 22. Domain-specific languages for game development

Nr.	Language	Application domain
82	GameMaker	2D game development with C-like scripting
83	Extensible Graphical Game Generator	game programming
84	Mogemoge	2D games
85	Scalable Games Language	scripting for games
86	Network Scripting Language	scripting and networking
87	4Blocks DSL	DSL for Tetris games (Haskell-based)
88	Casanova	game programming (integrated game engine)
89	Sound scene DSL	sound scene DSL (Haskell-based)
90	MUDDLE (historical account)	multi-user dungeon games (MUDs)
91	PuzzleScript	puzzle games

Table 23. Generic modeling languages applied to game design and development

Nr.	Language	Application domain
92	UML – Metamodeling	generic formalism for meta-modeling, e.g., applied to 2D platfrom games
93	UML – Class and State Diagrams	generic formalism for object-oriented analysis and design applied in model-driven game development
94	Statecharts	variants of a formalis that describes behaviors as state machines, applied to Game AI and dialogue in games
95	Feature Models	visual variability modeling formalism applied to managing game (and game engine) variability

Table 24. Domain-specific modeling languages for game development

Nr.	Language	Application domain
96	SharpLudus	RPG games, mobile touch-based games, 2D arcade games
97	Eberos GML2D	2D Games
98	FlexibleRules	digital board game creation and customization
99	PhyDSL	2D mobile physics-based games
100	Pong Designer	Pong-like games
101	Board Game DSL	board games
102	RougeGame Language	visibility Rogue-like dungeon maps
103	Reactive AI Language	behaviors in adventure games

Modeling Framework (EMF) and Ecore (the EMF model engine), it enjoys the advantage of generic reusable tools for model transformation, analysis and productivity, e.g., for adding a textual concrete syntax (Xtext, EMFText), or graphical ones (e.g., using GMF, Graphiti) [36]. In a recent literature review, Zhu and Wang present a model-driven perspective on game development [57].

6.14 Metaprogramming

The metaprogramming perspective considers game development as an application domain for generic language technology. Metaprogramming refers to techniques, tools and approaches for creating metaprograms that read and transform the source code of other programs, e.g., compilers, interpreters and integrated development environments. Applying these techniques to game development promises to raise productivity, improve quality and reduce maintenance costs.

Constructing languages and tools by means of metaprogramming requires appropriate meta-tooling. *Language work benches* are tools that provide high-level mechanisms for the implementation

Table 25. Applications of metaprogramming languages and language work benches

Nr.	Language	Metaprogramming language or work bench	URL
27	Micro-Machinations	Rascal	https://www.rascal-mpl.org
66	GAML	Xtext	https://www.eclipse.org/Xtext
96	SharpLudus	Microsoft DSL tools	https://visualstudio.microsoft.com
99	PhyDSL	Xtext	https://www.eclipse.org/Xtext
104	Whimsy	C++	(various versions exist)
105	Level editors	DiaMeta	http://www2.cs.unibw.de/tools/DiaGen
107	Ficticious	Ginger	(not found)
106	Text aventures	Racket	https://racket-lang.org
108	Dialog Script	Xtext	https://www.eclipse.org/Xtext

of software languages. Erdweg et al. describe the state of the art in language workbenches [14]. Examples of metaprogramming languages and language work benches include Epsilon, Gemoc Studio, Meta-Edit+, MPS, Racket, Rascal, Spoofax and Xtext. Several authors illustrate metaprogramming techniques and apply approaches to example languages. Table 25 shows examples of language of games and play created by means of language work benches.

The strength of this perspective is the application of state-of-the-art in language engineering and its weakness is that, with some exceptions, many illustrations remain toy examples that are never extensively validated.

7 CHALLENGES AND OPPORTUNITIES

Here we discuss research trends, synthesize insights and describe challenges and opportunities for future research and development. First, we discuss the results in general in Section 7.1. Next, we compare and analyze success factors in Section 7.2. Finally, we synthesize one additional perspective on languages of games and play in Section 7.3. Our Automated Game Design perspectives is a specific language-centric discussion on challenges and opportunities for research and development. This section answers research question RQ4.

7.1 General Analysis

Our area of interest 'languages of games and play' is a well-studied research topic with a growing number of publications. Figure 2 shows that most papers we included were published after the turn of the millennium. Around this time, roughly between 1998 and 2006, most of the interdisciplinary game publishing venues we identified also came into existence. Since 2005, there is a gradual increase in the term 'domain-specific language'. Two factors explain the declining number of publications in this study after 2015. First, the query date limits the search results. Second, GS orders the results of the wide query to show older results first. Therefore, it is likely we missed newer results that could have been included.

Our analysis of the citation graph, shown in Figure 3, reveals a low cohesion between publications. We observe clusters that represent distinct technological spaces, tight-knit communities, fragmented sub-topics and diffuse areas. Therefore, authors may not find or recognize related work in a wider research context. As a result, relevant literature has gone uncited, efforts and successes have often been one-off, lessons learnt have gone overlooked, and several studies and areas have remained isolated. We view this mapping study as an opportunity to relate relevant primary sources to help frame research problems, reuse available approaches, and benefit from documented experiences. Our map serves to navigate between related perspectives and technological spaces. As time progresses,

Table 26. Examples of multi-year, multi-disciplinary work on Languages of Games and Play

(AI: Artificial Intelligence, SE: Software Engineering, Edu: Education, G: Games)

Nr.	Language	Ct.	Years	Areas	Perspectives
65	AgentSheets	6	1995–2012	SE+Edu	visual DSLs, education
39	ABL	5	2002–2008	AI+G+SE	programming, behaviors, interactive drama
7	Gameplay Design Patterns	6	2003–2013	G+Edu	pattern language
47	ScriptEase	8	2003–2013	SE+G	pattern language, visual DSL
96	SharpLudus	6	2006–2012	SE+G	model-driven engineering, visual DSL
46	<e-game>	6	2006–2012	SE+Edu	storyboards, education, visual DSL
25	Biped and Ludocore	9	2008–2012	AI+G	automated game design, mechanics
26	Machinations	6	2009–2012	G	pattern language, automated game design
27	Micro-Machinations	3	2013–2015	SE+G	automated game design, mechanics, programming, visual DSL
88	Casanova	11	2011–2017	SE+G	game programming

Table 27. Highlighting the differences between applied and forgotten languages

	Alive languages	Dead languages
Language experts	multiple	one
Publication count	multiple	one or two
Publication areas	multiple	one
Validation	applied and validated in practice	not applied, toy examples
Availability	sources or wiki pages are released and maintained up to a point,	no source code is available
Examples	tutorials, workshops and study materials are available	not available

the map can be extended and reshaped for charting new research trajectories that continue to explore the limits of formalism.

7.2 Success Factors

Our analysis reveals a ”grave yard” of dead language prototypes. Few languages ever grow to maturity. Many languages remain solution proposals that are now no longer maintained, available or in use. This lack of reuse is unfortunate, since creating one language often requires years of research, design and development. Naturally, this leads to the question why so many prototypes were abandoned. Here we discuss observations and insights about shared success factors. Table 26 shows examples of languages that stand out as multi-faceted research with a relatively high publication count. We explain succes factors summarized in Table 27.

Languages of games and play represent a considerable effort and a long-term investment. Therefore, success requires a multi-year research trajectory, perhaps spanning multiple research grants and PhD projects. Publishing in different research areas helps answer different related questions and sheds light on challenges and solutions from different perspectives. Involving multiple researchers, language developers and practitioners creates co-ownership and continuity.

Traditionally, academia and the game industry have not always seen eye to eye. However, collaborating in applied research projects is essential for validating research in practice. To that end, some research departments include labs and game studios, and host in-house designers. Of course, working with innovative indie game developers or AAA studios on industrial case studies costs time and effort. The main benefit is that case studies can show case approaches and lead to better and more applicable solutions. Stakeholders can formulate common goals, agree to make research results open source, and protect intellectual property of businesses with Non-Disclosure

Agreements (NDAs). As a selling point, students participating in these projects might become employees who bring expertise and help to create new and innovative game products.

Naturally, making solutions available is necessary in order to apply them. Open source software might include reusable script engines, content generators, visual tools or programming environments. In addition, for learning to apply solutions effectively, users may require Wiki pages, blogs, example materials, tutorials and workshops.

Some languages, in particular educational online languages, have thriving user communities that create significant impact. Their users apply solutions in practice, which increases the research visibility and grows a network. Naturally, these benefits come at great cost, e.g., time spent on tools, demos, maintenance and legacy support. However, when users become stakeholders invested in validation, they also assist in building data sets. Ultimately, empirical evidence is necessary for scientific research.

7.3 Automated Game Design

Here, we synthesize a unifying perspective on languages of games and play by relating research perspectives from the previous section to challenges and opportunities for future research and development.

Automated Game Design (AGD) aims to speed-up and improve iterative game design processes by automating design processes. Here we discuss how languages, structured notations, patterns and tools can help game design experts raise their productivity and improve the quality of games and play. We distill research challenges and relate this perspective to other perspectives on languages of games and play.

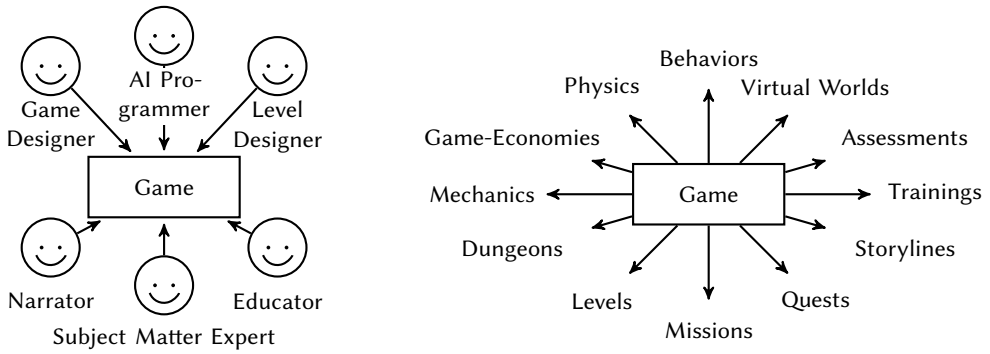
Various languages, techniques and tools have been proposed for creating, generating, analyzing and improving a game's parts. These design tools offer interactive user interfaces that support prototyping, sketching designs, automating play tests and exploring design spaces, usually in a mixed-initiative, conversational and collaborative manner. Moreover, these tools are often intended to support game design as an explorative, playful or enjoyable activity. AGD usually shifts design and implementation efforts away from manual, repetitive, time consuming and error-prone tasks. That way, designers can more efficiently create, improve and maintain growing amounts of *game content*. We define content as follows:

> "Game content refers to every asset of a digital game that can be separately (or together) viewed, understood, modeled, generated, recombined and improved to affect audio-visual and interactive player experiences."

Visual content includes textures, models, sprites, imagery, objects that form composite structures such as trees, landscapes, cities and nature [40]. Audio content includes music, voices and effects. Procedural Content Generation (PCG) refers to generative techniques that produce and transform game content [21, 28, 40, 49]. Game engines and software libraries, reusable components for the construction of digital games, are not content [40].

The focus of this study is on *'interaction-bound content'*, content that expresses how players interact and communicate, which especially affects player experiences. Figure 6a illustrates the diverse experts that might contribute to a game's design. Figure 6b illustrates types of content (or facets) a game might have. Modifying these content facets evolves a game along multiple related axes or dimensions.

The problem is that these dimensions overlap, often in non-trivial ways, which results in a web of criss-crossing interconnected content dependencies that may differ from game to game. This makes it exceptionally difficult to align a particular interpretation of a dimension with a reusable form of content representation that separates a game design concern. As a result, improving a game's



(a) Examples of experts that might contribute to a game's design

(b) Examples of interaction-bound content types whose evolution represents gameplay dimensions that span game design spaces

Fig. 6. Game design experts contribute content to evolve games in different dimensions

qualities along one dimension is often difficult without negatively affecting another. Therefore, design experts require *orchestrated* content generators [30] for composing high quality games from different kinds of inter-connected content, intricately interwoven to support meaningful experiences, in a reusable manner. Next, we relate key challenges of AGD to perspectives on languages of games and play.

Frameworks and pattern catalogues for studying games describe *what* games are. These perspectives inform automating game design by providing context, theory and structured frameworks for common vocabularies and notations.

- Game designers lack a common *vocabulary*, which hampers specification, communication and agreement among developers and designers [27]. However, automating game design requires formalizing game concepts, e.g., by performing a *domain analysis* that identifies concepts, names, meanings and relationships. Useful frameworks and resources are *ontologies and typologies* that help to describe, understand and characterize games, and distinguish what makes games unique. Section 6.1 highlighted this perspective.
- Game designers require design tools, reusable *patterns*, and techniques that help them analyze and predict how modifications to a game's design will affect the gameplay. Section 6.2 highlighted pattern languages and game design patterns that describe best practices, recurring structures of content and gameplay, and represent steps towards standardization and reuse.

Other perspectives are content-centric. Related publications typically envision design experts who contribute design artifacts by means of languages and tools. We describe the following perspectives:

- Games may require integrating *subject matter knowledge*. The challenge is providing languages and tools that enable domain experts such as as educators and psychologists to describe scenarios and participate in game design processes. Section 6.3 highlighted a perspective on applied (or serious) game design.
- Many games integrate *game mechanics*, rules that offer playful affordances and bring about interesting player experiences. Game designers require formalisms to express game mechanics, e.g., game economies or avatar physics. Additionally, they require tools for analyzing behaviors, generating rules, balancing strategies, introducing trade-offs, managing feedback-loops and automating play testing. Section 6.4 highlighted a perspective on game mechanics.
- Many games include generated *spaces* such as virtual worlds and game levels. Designers require tools to assist in populating these spaces with various kinds of content, e.g., to

create varied and interesting game levels, dungeons, missions and quests. Section 6.5 gave a perspective on spaces.
- Game designs may integrate *behaviors* of in-game entities such as non-player characters that require dramatic realism or challenge. Designers and AI programmers require formalisms for expressing these behaviors. Section 6.6 illustrated a perspective on behavior languages with different strengths and applications.
- Games designs may integrate *stories* that allow players to progress through stages of a plot. Designers and narrators require languages and tools to expressing these stories. Section 6.7 illustrated a perspective on techniques that express narratives and story plot using textual notations and graphs.

Technical views on *how* to automate game design with available techniques and approaches originate mainly from the fields of AI and games, software engineering, and education. We relate the following challenges to technical perspectives on languages of games and play:

- Raising the quality of game content requires automated iterative analyses, especially when dealing with generated content. Section 6.8 highlighted a perspective on metrics, which can be used to relate content and player actions to gameplay.
- Usability, understandability and ease of use are essential qualities for game design tools. Designers require appropriate visual notations and timely feedback to comprehend concepts, master language features and learn to apply user interfaces effectively. We described a perspective on educational end-user environments in Section 6.9.
- During a digital game's life span, designers may need to make significant changes to its design. Designers require tools that enable modifying a game's design during its entire evolution. Section 6.10 gave a perspective on gamification, which studies techniques for redesigning games and retrofitting game designs.
- Powerful, cutting-edge *AI techniques* are constantly being researched, developed and improved. The challenge is leveraging these for automate game design, e.g., for analysis, generation and testing. Section 6.11 described a perspective on a field called *general game playing*, which uses games as a test-bed for AI.
- Developing high quality games in a time-to-market manner requires *programming and maintaining* growing amounts of source code, and rapidly evolving that code towards new gameplay goals. Section 6.12 highlighted a perspective on DSLs, script languages and programming environments, which have been created to speed up development and improve the quality and maintainability.
- Developing *visual languages* for game design from scratch is a difficult and time-consuming endeavour. Model-driven engineering offers several reusable formalisms, techniques and approaches. We highlighted a perspective on how design can be automated by means of visual models and tools in Section 6.13.
- Developing and maintaining DSLs, generators and tools costs a considerable amount of time and effort. Language developers require appropriate *meta-tooling* and metaprogramming techniques to alleviate this effort, that enable rapid prototyping. Several authors advocate the use of generic language technology by demonstrating its power in illustrative examples. We highlighted this perspective in Section 6.14.

We describe two additional open research challenges. First, a game's quality is limited by the number of game design iterations. Producing high quality games more quickly requires the duration of game reducing iterations. An open challenge is leveraging *live programming* techniques for providing *live* (immediate and continuous) *feedback* on changes to a program. This may be the key to forming more accurate mental models and better predicting behavioral effects and gameplay

outcomes. Second, the composition of content alone does not explain how a game's simulation is communicated to its players. To fully understand how a game works, designers might require content creation strategies that relate content representations to a set of communication strategies or Operational Logics (Language 17).

8 RELATED WORK

The research perspectives we have described in Section 6 relate work on languages of games and play. We have already mentioned several surveys on more specific topics that align with those perspectives. Here we briefly discuss more general related surveys, literature reviews and mapping studies. In addition, we give an overview of related PhD dissertations in Appendix A.

Ampatzoglou et al. perform a systematic review on software engineering research for computer games [7]. They identify topics, research approaches and empirical research methods.

In addition, two surveys relate to the model-driven-engineering perspective presented in Section 6.13. Tang and Hanneghan examine the state-of-the-art in model-driven game development from a game-based learning perspective [44]. Their overview describes model transformations and several game design languages. Zhu and Wang present a literature review that analyzes 26 model-driven approaches [57]. Their protocol zooms in on target game domains (genres), domain frameworks, modeling languages, tooling, and evaluation methods.

Beckmann et al. perform an exploratory literature study on 'live' tooling in the game industry [9]. They analyze articles on Gamasutra and videos of the Game Developers Conference, and relate identified tools to degrees of live programming.

Almeida and da Silva survey game design methods and tools [6], and synthesize requirements from 32 selected publications [5]. They present a map of design frameworks and visual modeling formalisms [6], which overlaps with our perspectives on Ontologies (Section 6.1), Pattern Languages (Section 6.2) and Game Mechanics (Section 6.4).

Several vision papers align with this study. Walter and Masuch discuss how to integrate DSLs into the game development process [53]. Mehm et al. take an authoring tools perspective when discussing research trends [32].

9 THREATS TO VALIDITY

Systematic mapping studies are intended to create an *unbiased* and *complete* overview of a subject. With that in mind, we have applied methodology guidelines [26], and designed a reproducible and an unambiguous protocol. However, the results of this study are a compromise. By definition the word 'game' is a concept whose 'essence' cannot be captured in words [55]. Therefore this study can never be fully complete, unambiguous, and unbiased. Here we address threats to validity.

Scoping the area of interest. We formulated two queries to obtain evidence for our hypotheses. Our narrow query, aimed at our second hypothesis, is biased towards software engineering where the term 'domain-specific language' is common. To obtain a more nuanced and complete overview that also includes other research fields, we formulated a wide query aimed at our first hypothesis, with focus on languages in general. However, more than 16K GS results was more than is feasible for us to analyze. We compromised and chose to limit our analysis to its top 1K results. In addition, we filtered terms that are often, but not always, off-topic. As a result, despite our best efforts, we may have overlooked relevant publications.

Breaking protocol. We cited publications not conforming to either of our queries to clarify the origins, descriptions and applications of a language. In addition, we included several papers that conform to the wide query, but did not appear in the top 1K results. We have clearly marked these in the language summaries of Appendix B, and added the bibliographical data in a separate library,

as can be seen in Figure 2. While this makes our overview more complete, it also reintroduces the selection bias we wished to avoid in the first place.

Pilot error. We have summarized and related publications from a wide array research fields, areas and topics. Unfortunately, despite our best efforts, we have inevitably overlooked or mischaracterized contributions. This study is intended as an inclusive, constructive and 'living' document that we hope to discuss, improve and extend over time.

Synthesizing perspectives. We synthesized fifteen perspectives on languages of games and play from over one hundred language summaries. Our decomposition of the topic of interest is a best-effort interpretation that relies on our personal experience. We acknowledge that it is possible to formulate other research perspectives that extend the collection. Different authors, who have distinct research needs and goals, might want tot shed light on a problem from a different angle. They could choose finer granularity to zoom in on an area, and reuse different subsets of languages that cross-cut topics in different ways.

Missing in action: game development practice. We have mapped the state-of-the-art in languages of games and play for a wide audience, which in our view, should include practitioners. Unfortunately, because we identified relatively few practical sources, we have not fully delivered on this promise. We acknowledge this is a limitation of our research method, which does not include non-written sources such as games, development kits, engines and tools. Of course, GS primarily contains academic sources, and the game development industry is not in the business of publishing papers. In addition, fierce competition and time-to-market pressure have lead to a degree of industrial secrecy. By not sharing information, many businesses are simply protecting their intellectual property and competitive edge.

Using Google Scholar. Our choice for GS is motivated by its high recall. Using GS we obtained publications from independent venues we did not know existed. However, Google owns the information records on GS, maps the interests of its users, and does not provide bulk access[11]. This complicates systematic studies in general, which require an off-line analysis in order to 'stand on the shoulders of giants'. As a result, it is not straightforward to reproduce this study.

Applying bibliometrics. We obtained citation data from GS and constructed the citation graph using a combination of Python scripts and Gephi. Given the right tools, we could have extracted the citation data directly from the PDFs. In addition, we used Gephi's built-in layout algorithms to obtain a suitable image. However, the same data can produce different graph layouts as well. Mapping studies are complicated by a lack of tools for bibliographic analysis and bibliometrics. Standardizing and automating mapping studies can help save precious time and improve the quality of literature reviews and surveys in general.

10 CONCLUSION

We have presented an overview of the state-of-the-art in languages of games and play that relates research areas, goals and applications. We identified and summarized over one hundred languages, and synthesized fifteen research perspectives (or angles) on the topic, each illustrated by selected language summaries. The results show that there is evidence to support both of our research hypotheses. First, languages, structured notations, patterns and tools can offer designers and developers theoretical foundations that offer experts systematic techniques and practical solutions they need to raise their productivity and improve the quality of fames and play. Second, we obtained evidence that DSLs can help in automated game design, and described illustrative examples that suggest how to achieve this. We also mapped related approaches and described perspectives other than our own departure point with distinct approaches and motivations. Instead of clarifying a

[11] https://scholar.google.com/intl/en/scholar/help.html (visited August 23rd 2019)

single perspective, our map leads in many related research directions, each representing possible departure points for related studies. Ultimately, our map provides a good starting point for anyone who wishes to study and learn more about languages of games and play.

Future Work. We foresee the following future work. First, we plan to create a Wiki on languages of games and play that we can grow and maintain related work as a 'living document', like Gameplay Design Patterns and (Language 7) and Game Ontology Project (Language 2). Second, we see opportunities for additional mapping studies and literature reviews that intersect with this study and zoom in on specific related areas, such as automated game design, mixed-initiative approaches, procedural content generation and live programming [9].

REFERENCES

[1] E. Aarseth. "Define Real, Moron! Some Remarks on Game Ontologies". In: *DIGAREC Keynote-Lectures 2009/10*. DIGAREC 6. Potsdam UP, 2011.

[2] E. Aarseth et al. "A Multi-Dimensional Typology of Games". In: *Proceedings of the 2003 DiGRA International Conference: Level Up, DiGRA 2003, Utrecht, The Netherlands, November 4–6, 2003*. Utrecht University, 2003.

[3] A. V. Aho et al. *Compilers: Principles, Techniques, and Tools*. Addison-Wesley, 1986.

[4] C. Alexander et al. *A Pattern Language - Towns, Buildings, Construction*. Oxford University Press, 1977.

[5] M. S. Almeida and F. S. da Silva. "Requirements for Game Design Tools: a Systematic Survey". In: *Proceedings of the 12th Brazilian Symposium on Games and Digital Entertainment, São Paulo, Brazil, October 16–18, 2013, SBGames 2013*. 2013.

[6] M. S. O. Almeida and F. S. C. da Silva. "A Systematic Review of Game Design Methods and Tools". In: *Entertainment Computing – Proceedings of the 12th International Conference, ICEC 2013, São Paulo, Brazil, October 16–18, 2013*. Vol. 8215. LNCS. Springer, 2013.

[7] A. Ampatzoglou and I. Stamelos. "Software Engineering Research for Computer Games: A Systematic Review". In: *Information & Software Technology* 52.9 (2010).

[8] E. F. Anderson. "On the Definition of Non-Player Character Behaviour for Real-Time Simulated Virtual Environments". PhD thesis. Bournemouth University, Apr. 2008.

[9] T. Beckmann et al. "An Exploratory Literature Study on Live-Tooling in the Game Industry". In: *Workshop on Live Programming (LIVE) 2019, Athens, Greece, October 20, 2019*. 2019.

[10] C. Browne et al. "Guest Editorial: General Games". In: *IEEE Transactions on Computational Intelligence and AI in Games* 6.4 (Dec. 2014).

[11] R. Coyne. "Wicked problems revisited". In: *Design studies* 26.1 (2005).

[12] R. Dörner et al. *Serious Games*. Springer, 2016.

[13] M. P. Eladhari and E. M. I. Ollila. "Design for Research Results: Experimental Prototyping and Play Testing". In: *Simulation & Gaming* 43.3 (2012).

[14] S. Erdweg et al. "The State of the Art in Language Workbenches – Conclusions from the Language Workbench Challenge". In: *Software Language Engineering – Proceedings of the 6th International Conference, SLE 2013, Indianapolis, IN, USA, October 26–28, 2013*. Vol. 8225. LNCS. Springer, 2013.

[15] N. Fenton and J. Bieman. *Software Metrics: A Rigorous and Practical Approach*. 3rd ed. CRC press, 2014.

[16] T. Fullerton et al. *Game Design Workshop: Designing, Prototyping, and Playtesting Games*. CMP Books, 2004.

[17] T. Fullerton et al. *Game Design Workshop: A Playcentric Approach to Creating Innovative Games*. 2nd ed. Morgan Kaufmann, 2008.

[18] E. Gamma et al. *Design Patterns: Elements of Reusable Object-Oriented Software*. Addison-Wesley, 1994.

[19] *Postmortems from GameDeveloper*. CMP Books, 2003.

[20] J. S. Harbour and J. R. Smith. *DarkBASIC Pro Game Programming*. 2nd ed. Thomson Course Technology, 2007.

[21] M. Hendrikx et al. "Procedural Content Generation for Games: A Survey". In: *ACM Transactions on Multimedia Computing, Communications and Applications* 9.1 (Feb. 2013).

[22] E. Horvitz. "Principles of Mixed-Initiative User Interfaces". In: *Proceeding of the CHI '99 Conference on Human Factors in Computing Systems: The CHI is the Limit, Pittsburgh, PA, USA, May 15–20, 1999*. ACM, 1999.

[23] J. Huizinga. *Homo Ludens: Proeve Ener Bepaling van het Spelelement der Cultuur*. Wolters-Noordhoff, 1938.

[24] R. Hunicke et al. "MDA: A Formal Approach to Game Design and Game Research". In: *Proceedings of the AAAI workshop on Challenges in Game Artificial Intelligence*. AAAI, 2004.

[25] J. Juul. *Half-real: Video Games between Real Rules and Fictional Worlds*. MIT press, 2011.

[26] B. Kitchenham and S. Charters. *Guidelines for performing Systematic Literature Reviews in Software Engineering*. Tech. rep. Keele University and Durham University Joint Report, 2007.

[27] P. Klint and R. van Rozen. "Micro-Machinations: a DSL for Game Economies". In: *Software Language Engineering – Proceedings of the 6th International Conference on Software Language engineering, SLE 2013, Indianapolis, IN, USA, October 26–28, 2013*. Vol. 8225. LNCS. Springer, 2013.

[28] B. Kybartas and R. Bidarra. "A Survey on Story Generation Techniques for Authoring Computational Narratives". In: *IEEE Transactions on Computational Intelligence and AI in Games* 9.3 (Sept. 2017).

[29] R. Lämmel. *Software Languages: Syntax, Semantics, and Metaprogramming*. Springer, 2018.

[30] A. Liapis et al. "Orchestrating Game Generation". In: *IEEE Transactions on Games* 11.1 (2019).

[31] M. Mateas and A. Stern. "Build It to Understand It: Ludology Meets Narratology in Game Design Space". In: *Proceedings of the 2005 DiGRA International Conference: Changing Views: Worlds in Play, DiGRA 2005, Vancouver, Canada, June 16–20, 2005*. Digital Games Research Association, 2005.

[32] F. Mehm et al. "Future Trends in Game Authoring Tools". In: *Entertainment Computing – Proceedings of the 11th International Conference on Entertainment Computing, ICEC 2012, as part of the 2nd Workshop on Game Development and Model-Driven Software Development, GD&MDSD 2012, Bremen, Germany, September 26–29, 2012*. Springer, 2012.

[33] T. Mens. "Introduction and Roadmap: History and Challenges of Software Evolution". In: *Software Evolution*. Springer, 2008.

[34] M. Mernik et al. "When and How to Develop Domain-Specific Languages". In: *ACM Computing Surveys* 37.4 (Dec. 2005).

[35] M. J. Nelson. *Sicart's 'Against Procedurality' – A reply*. Kmjn.org. May 2012.

[36] R. F. Paige et al. "Metamodelling for Grammarware Researchers". In: *Software Language Engineering, 5th International Conference, SLE 2012, Dresden, Germany, September 26–28, 2012, Revised Selected Papers*. Vol. 7745. LNCS. Springer, 2013.

[37] K. Petersen et al. "Systematic Mapping Studies in Software Engineering". In: *12th International Conference on Evaluation and Assessment in Software Engineering, EASE 2008, University of Bari, Italy, June 26–27, 2008*. Workshops in Computing. BCS, 2008.

[38] K. Salen and E. Zimmerman. *Rules of Play - Game Design Fundamentals*. The MIT Press, 2003.

[39] J. Schell. *The Art of Game Design: A Book of Lenses*. AK Peters/CRC Press, 2014.

[40] N. Shaker et al. *Procedural Content Generation in Games: A Textbook and an Overview of Current Research*. Computational Synthesis and Creative Systems. Springer, 2016.
[41] M. Sicart. "Defining Game Mechanics". In: *Game Studies* 8.2 (Dec. 2008).
[42] M. Sicart. "Against Procedurality". In: *Game Studies* 11.3 (Dec. 2011).
[43] P. Spronck. "CGAIDE AND GAME-ON 2004". In: *ICGA Journal* 27.4 (Dec. 2004).
[44] S. Tang and M. Hanneghan. "State-of-the-Art Model Driven Game Development: A Survey of Technological Solutions for Game-Based Learning". In: *Journal of Interactive Learning Research* 22.4 (Dec. 2011).
[45] U. Tikhonova et al. "Applying Model Transformation and Event-B for Specifying an Industrial DSL". In: *Proceedings of the 10th International Workshop on Model Driven Engineering, Verification and Validation MoDeVVa 2013, co-located with 16th International Conference on Model Driven Engineering Languages and Systems (MoDELS 2013), Miami, Florida, USA, October 1st, 2013*. Vol. 1069. CEUR Workshop Proceedings. CEUR-WS.org, 2013.
[46] J. Togelius et al. "Search-Based Procedural Content Generation: A Taxonomy and Survey". In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.3 (2011).
[47] M. Treanor and M. Mateas. "An Account of Proceduralist Meaning". In: *Proceedings of the 2013 DiGRA International Conference: DeFragging Game Studies, DiGRA 2013, Atlanta, GA, USA, August 26–29, 2013*. Digital Games Research Association, 2013.
[48] J. van den Bos and T. van der Storm. "Bringing Domain-Specific Languages to Digital Forensics". In: *Proceedings of the 33rd International Conference on Software Engineering, ICSE 2011, Waikiki, Honolulu , HI, USA, May 21-28, 2011*. ACM, 2011.
[49] R. van der Linden et al. "Procedural Generation of Dungeons". In: *IEEE Transactions on Computational Intelligence and AI in Games* 6.1 (Mar. 2014).
[50] A. van Deursen. "Domain-Specific Languages versus Object-Oriented Frameworks: A Financial Engineering Case Study". In: *Proceedings Smalltalk and Java in Industry and Academia, STJA'97, Erfurt, September 1997*. Ilmenau Technical University, 1997.
[51] A. van Deursen et al. "Domain-Specific Languages: An Annotated Bibliography". In: *ACM SIGPLAN NOTICES* 35 (2000).
[52] C. van Grinsven et al. *Games Monitor The Netherlands 2018 – Full Report*. Tech. rep. 2019.
[53] 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, ACE 2011, Lisbon, Portugal, November 8–11, 2011*. ACM, 2011.
[54] R. Wieringa et al. "Requirements Engineering Paper Classification and Evaluation Criteria: A Proposal and a Discussion". In: *Requirements Engineering* 11.1 (Mar. 2006).
[55] L. Wittgenstein. *Philosophical Investigations*. Blackwell, 1953.
[56] G. N. Yannakakis and J. Togelius. *Artificial Intelligence and Games*. Springer, 2018.
[57] M. Zhu and A. I. Wang. "Model-driven Game Development: A Literature Review". In: *ACM Computing Surveys* 52.6 (Nov. 2019).

A RELATED DISSERTATIONS

Several dissertations also describe one or more languages of games and play, usually as part of a literature study chapter. These chapters have a more narrow focus than this study, and few are systematic studies. Table 28 shows a selection of PhD dissertations. Authors of dissertations approach the topic from various angles. For instance, Maggiore includes libraries and systems when discussing languages [150]. Neil discusses several existing game design tools, mainly academic prototypes, and evaluates their application in supporting practical game design activities [187]. We refer to our citation data for additional PhD, Master and Bachelor dissertations.

Table 28. Several PhD Dissertations related to Languages of Games and Play

author	thesis	query	research angle	language
Abbadi	[2]	195n	DSL for general game development	88: Casanova
Ahmadi	[7]	158n	Game based learning	65: AgentSheets
Anderson	[14]	247n	Behaviors and Game AI	40: SEAL
Ašeriškis	[17]	241n	Gamification	68: UAREI
Borghini	[36]	263n	Assessment systems in game based learning	21: EngAGe DSL
Browne	[40]	–		72: Ludi
Dormans	[76]	97w	Game mechanics and level generation	26: Machinations 36: Ludoscope
Furtado	[100]	93n	Domain-Specific Modeling Languages	96: SharpLudus
Guo	[109]	97n	Modeling Pervasive Games	4: PerGO
Gaudl	[103]	274n	Real-Time Game AI	44: Posh #
Guana	[106]	209n	Modeling Games	99: PhyDSL
Holloway	[117]	272n	Modeling storylines	
Mahlmann	[151]	289n		73: SGDL
Mayer	[166]	390n		
Martens	[157]	–	Programming narratives in Linear Logic	48: Ceptre
Neil	[187]	–	Evaluates game design tools	
Mehm	[171]	238n	authoring tools for the educational domain	18: StoryTec
Osborn	[200]	–	Operationalizing Operational Logics	17: Operational Logics 57: Playspecs 30: Gamelan
Smith	[237]	494w	Mechanizing exploratory game design	25: Biped and Ludocore
Zhu	[291]	165n		103: RAIL
Zook	[292]	–w	Automated iterative game design	31: PDDL

B LANGUAGE SUMMARIES

Here we give summaries of languages of games and play. In addition, we give a complete bibliography of all publications we included in this mapping study.

B.1 Ontologies and Typologies

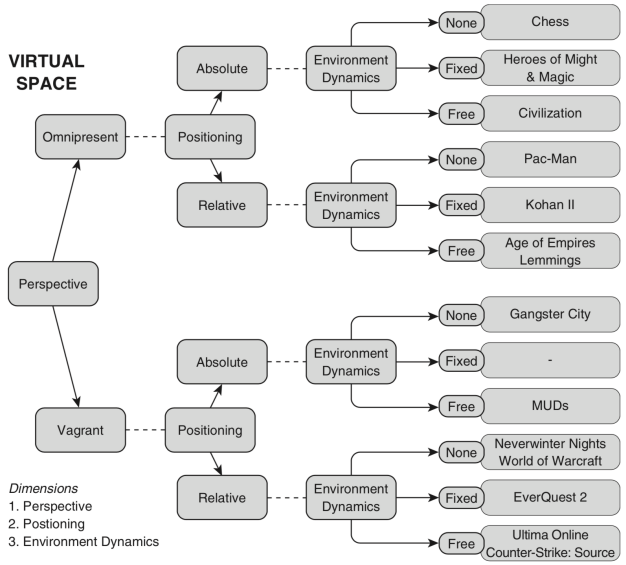


Fig. 7. Game Typology – The Metacategory Space Containing the Dimensions Perspective, Positioning, and Environment Dynamics (appears in Elverdam and Aarseth [82])

Language 1 ***Game Typology*** *generic / framework / research*

Elverdam and Aarseth discuss a multi-dimensional typology of games [82], an extension of prior work [1]. Instead of using game genres for classifying games, which may be arbitrary, contradictory, or overlapping, they propose using an open-ended ontological model. It has a metacategories Player Composition, Player Relation, Struggle, Game State, and Time (Internal and External) and Space (Virtual and Physical). Virtual Space has the dimensions Perspective, Positioning, and Environment Dynamics. Figure 7 depicts its Perspective dimension in a visual notation. Read from left to right (and from abstract to concrete), dimensions (shown as rounded rectangles) have alternatives (marked by arrows) and sub-dimensions (horizontal dashed lines), which are also categories. On the far right are game instances, distinguished by the typology.

publication	query	publication type	research category	note
[1]	~~language~~	conference paper	proposal of solution	
[82]	16w	journal article	proposal of solution	

Language 2 ***Game Ontology Project*** *generic / framework / practice*

Zagal et al. present the Game Ontology Project (GOP), a framework offering a unified game design vocabulary to describe, analyse and study existing games and facilitate the design of new ones [288]. The ontology abstracts away representational details of games such as setting, genre and player knowledge, and its aim is to characterize the game design space. The top-level elements are Interface, Rules, Entity Manipulation and Goals. GOP is available online in Wiki form[1].

[1]https://www.gameontology.com/ (last visited November 17th 2018)

publication	query	publication type	research category	note
[288]	15w	conference paper	proposal of solution	
[289]	15w	book chapter	proposal of solution	reprint
[287]	601w	book	validation research	Ludoliteracy

Language 3 ***Ontology of Journalism*** *genre-specific / tool / practice*

Dowd explores how the design of persuasive learning systems for journalists can be guided by an ontology for journalism [80]. The ontology describes vocabulary, concepts and emotions in the journalism domain, including social media and crowdsourcing. Robojourno is a synthetic player that helps journalists reflect on their core values by responding to emotional inputs. Defined as a Finite State Machine, it leverages links between roles, actions, before- and after-intentions and Hoare logic, e.g., "`{Curiosity} publish story {satisfied}`" and "`{Competitive} scoop please {smug}`".

publication	query	publication type	research category	note
[80]	166w	conference paper	proposal of solution	

Language 4 ***Pervasive Games Ontology*** *genre-specific / tool / practice*

Guo and Trætteberg propose a Pervasive Games Ontology (PerGO) for structuring and accelerating the process of analyzing the game domain, specifically aimed at pervasive games, i.e. games extending into the real world [110]. Guo et al. evaluate a model-driven development methodology for a location-based game called RealCoins [112], and propose a domain-specific modeling workflow [111]. Pløhn et al. extend PerGO and perform a case study on a game called Nuclear Mayhem [213].

publication	query	publication type	research category	note
[110]	9n	workshop paper	proposal of solution	
[112]	30n	journal paper	validation research	
[111]	50n	conference paper	validation research	
[109]	97n	PhD Thesis	evaluation research	
[213]	107n	journal paper	validation research	

B.2 Pattern Languages and Design Patterns

Language 5 *Formal Abstract Design Tools* *generic / framework / practice*

Church proposes Formal Abstract Design Tools (FADT), a kind of pattern language for game design, with broad categories intention, perceivable consequence and story. This influential publication has influenced later works, because it offered a new frame for research and practice for building on past discoveries, sharing concepts behind successes, and applying lessons learnt from one domain (or genre) to another.

publication	query	publication type	research category	note
[54]	–w	magazine article	proposal of solution	
[55]	–w	magazine article	proposal of solution	reprint

Language 6 *Game Design Patterns* *generic / framework / practice*

Kreimeier proposes a pattern language for game design aimed at promoting reuse that describes problem, solution, consequence, examples and references. Example patterns include Privileged Move, for restricting actions and Weenie, for reorienting players. This work inspired later approaches.

publication	query	publication type	research category	note
[137]	67w	article	proposal for solution	

Language 7 *Gameplay Design Patterns* *generic / framework / practice*

Björk et al. propose a framework for game design patterns, which was later renamed *gameplay design patterns*, to support the design, analysis, and comparison of games. The pattern language describes components of games and interaction patterns that express how players or a computer use these components to affect aspects of gameplay. Language elements include name, description, usage, consequences, relations, relations and history. Holopainen et al. describe teaching gameplay design patterns using a tool called CAGE, which visualizes design goals as a graph of related design facets [120]. Holopainen and Björk further expand the gameplay design patterns collection through exploring how games support motivation [119]. Zagal et al. discuss dark design patterns that cause negative player experiences and whose intent is questionable and perhaps even unethical, and how to identify them [290]. A pattern catalogue is available online in Wiki form, along with a list of related publications[1].

[1] http://www.gameplaydesignpatterns.org (Last visited November 19th 2018)

(a) Emergent gameplay (b) Interaction through game design and play

Fig. 8. MDA perspectives (adapted from Hunicke et al. [122])

publication	query	publication type	research category	note
[118]	150w	lecture notes	lecture	
[33]	313w	conference paper	proposal of solution	
[32]	927w	book chapter	proposal of solution	reprint
[120]	605w	conference paper	proposal of solution	
[119]	839w	conference paper	validation research	
[290]	830w	conference paper	proposal of solution	

Language 8 ***Mechanics Dynamics Aesthetics*** *generic / framework / research*

Hunicke et al. present the Mechanics Dynamics and Aesthetic (MDA) framework for understanding games, bridging the gap between design, development, game criticism, and technical game research [122]. The framework is schematically shown in Figure 8. Players interact with games via mechanisms (a.k.a. mechanics, or rules) created by game designers. During a game's execution, playful acts result in dynamic interaction sequences. Ideally, these are also aesthetically pleasing experiences called *gameplay*, e.g., fellowship, challenge, fantasy, narrative, discovery or self-expression.

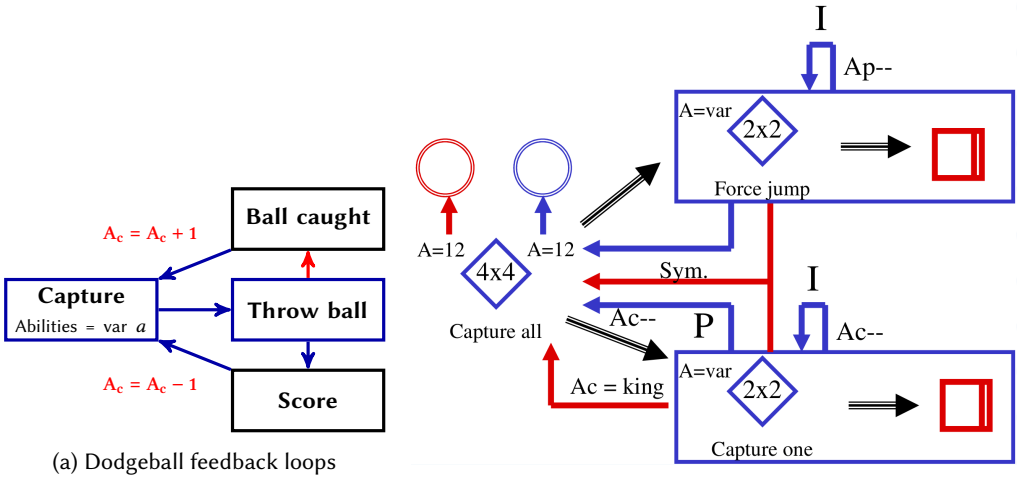
publication	query	publication type	research category	note
[122]	~~language~~	workshop paper	philosophical paper	

Language 9 ***Pattern Language for Sound Design*** *generic / tool / practice*

Alves and Roque propose a sound pattern language for empowering game developers in sound design. They present a collection of illustrative patterns in an accessible format based on best practices [10]. In addition, they propose and evaluate a deck of cards for sound design [11, 12] and report experiences [13]. Examples of sound patterns include Achievement, Failure, Anticipation, Directionality and Hurry Up! The patterns are available in Wiki form[1].

[1] http://www.soundingames.com (Last visited March 21st 2019)

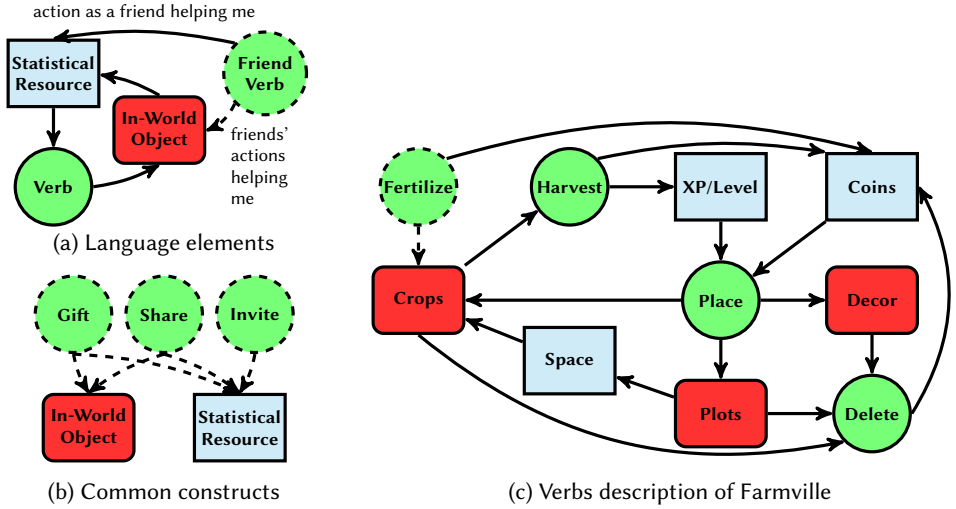
publication	query	publication type	research category	note
[10]	65w	conference paper	proposal of solution	
[11]	575w	conference paper	evaluation research	
[12]	820w	conference paper	evaluation research	
[13]	–w	workshop paper	experience report	



(a) Dodgeball feedback loops

(b) An attempt by Koster to model Checkers

Fig. 9. A Grammar of Gameplay (diagrams appear in Koster [134])



(a) Language elements

(b) Common constructs

(c) Verbs description of Farmville

Fig. 10. The Verbs language fits on a napkin (diagrams adapted from Koster [136])

Language 10 *Verbs* *generic / framework / practice*

Koster proposes "A Grammar of Gameplay" [134], an informal notation for gameplay design that describes atomic player activities and affordances by using *verbs* (or *ludemes*). The notation is inspired by "A Theory of Fun" [135]. It consist of atoms shown as rectangles enclosing a verb, which can be labeled with additional information, e.g., board game size (e.g., 8*x*8), conditions (e.g., *var* < 256) and time passing (vertical bar on the right). The arrows denote a sub-relation between general and specific atoms, and also express success and failure outcomes (marked in

blue and red respectively) that help in analyzing feedback loops, e.g., in Dodgeball as shown in Figure 9.
Bojin studies game design epistemologies from a language philosophy perspective that applies Wittgenstein's language games [34]. The author examines ludemes with respect to limits of formal language for expressing experiences [35]. Koster challenges academics to explore these limits, and shows several excerpts from an updated and simplified Verbs language depicted in Figure 10 [136]. The language consists of verbs (enclosed by a green circle), friend verbs (dashed line) statistical resources (light blue rectangle) and in-world objects (red rounded rectangle).

publication	query	publication type	research category	note
[134]	–w	presentation slides	proposal of solution	Grammar of G.
[34]	17w	journal article	philosophical paper	language games
[35]	377w	conference paper	philosophical paper	Grammar of G.
[136]	–w	presentation slides	discussion piece	Verbs

Language 11 ***Collaboration Patterns*** *genre-specific / tool / practice*

Azadegan and Harteveld examine how Collaboration Engineering (CE) can help study the design of collaborative games, and how such games support collaboration through their game mechanics [21]. They analyse two games using facilitation techniques from CE called *Thinklets*. Thinklets are modular sets of rules intended for creating predictable patterns of collaboration that desribe how people interact and work together towards a common goal. Thinklets specify preferred actions that, given constraints and capabilities, are appropriate for specific roles.

publication	query	publication type	research category	note
[21]	944w	workshop paper	proposal of solution	

Language 12 ***Mixed Reality Pattern Cards*** *genre-specific / tool / practice*

Wetzel introduces a set of playing cards as a pattern language for designing Mixed Reality games, and presents initial findings on their application [280]. These cards are available for purchase[1].

[1] https://www.pervasiveplayground.com/mixed-reality-game-cards/ (visited May 10th 2019)

publication	query	publication type	research category	note
[279]	871w	workshop paper	proposal of solution	
[280]	133w	workshop paper	proposal of solution	

Language 13 ***Design Patterns in FPS Games*** *genre-specific / tool / practice*

Hullett and Whitehead present level design patterns for creating varied and interesting First Person Shooter (FPS) games [121]. The pattern language associates building blocks of level design to resulting gameplay, and consists of the sections description, affordances, consequences and examples. Categories and patterns include Positional Advantage (Sniper location, Gallery

and Choke Point), Large-scale Combat (Arena and Stronghold), Alternate Gameplay (Turret and Vehicle Selection) and Alternate Routes (Split Level, Hidden Area and Flanking Route).

publication	query	publication type	research category	note
[121]	–w	conference paper	proposal of solution	

Language 14 ***Structural Composition Patterns*** *genre-specific / tool / practice*

Winters and Zhu propose guiding the spacial navigation of players in 3D adventure games with structural composition patterns [286]. They analyze the games Uncharted 3, Dear Esther, and Journey, and derive five patterns that direct a player's attention: Contrasting Shape, Framed Structure, Directional Line, Shifting Elevation and Structural Exaggeration. They perform user tests on simulated 3D environments that encode the patterns, interview the players, and show that especially Shifting Elevation and Directional Line patterns influence player movement.

publication	query	publication type	research category	note
[285]	201w	poster abstract	proposal of solution	
[286]	–w	conference paper	validation research	

Language 15 ***Flow Experience Patterns*** *generic / framework / practice*

Lemay proposes a pattern language for analyzing and understanding how a video game's elements can help trigger and maintain the most positive and intense player experiences [138]. Flow patterns describe a name, problem statement, context, solution, forces affecting the problem, and examples. They relate to the facets sensation, emotion, cognition, behavior and social interaction.

publication	query	publication type	research category	note
[138]	39w	conference paper	proposal of solution	

Language 16 ***Serious Games Design Patterns*** *generic / framework / practice*

Marne proposes a pattern library for serious games design aimed at improved understanding and cooperation between project team members, organizations and stakeholders[1]. Reified Knowledge is an example pattern described in detail.

[1] http://seriousgames.lip6.fr/site/spip.php?page=design_patterns&lang=en (visited April 26th 2019)

publication	query	publication type	research category	note
[156]	46w	conference paper	proposal of solution	

Language 17 ***Operational Logics*** *generic / framework / practice*

Mateas and Wardrip-Fruin propose a framework for game analysis called Operational Logics (OLs). They define the term as follows: *"An operational logic defines an authoring (representational) strategy, supported by abstract processes or lower-level logics, for specifying the behaviors a system must exhibit in order to be understood as representing a specified domain to a specified audience."* [163]. Thus, OLs describe both how a game functions internally and how its simulation is communicated to players. Osborn et al. expand on this work by refining and operationalizing several OLs [200, 203]. Example logics are: Collision-, Resource-, Persistence- and Character-State Logics. Logics are shown as a pattern language with the fields: Communicative role, Abstract process, Abstract operations, Presentation, Required concepts and Provided concepts. Osborn et al. propose a new field of research called Automated Game Design Learning (AGDL) for learning game designs expressed as OLs through simulating play [202].

publication	query	publication type	research category	note
[163]	–w	conference paper	philosophical paper	
[203]	–w	conference paper	proposal of solution	pattern language
[202]	–w	conference paper	philosophical paper	AGDL
[200]	–w	PhD thesis	validation research	

B.3 Applied Game Design

Language 18 ***StoryTec*** *generic / tool / educative*

Mehm et al. present the StoryTec system, an authoring tool for non-programmers for creating Digital Education Games (DEGs) [172]. Story descriptions consist of scenes. These are visually modeled using story units that have dramatic functions, e.g., derived from the Heroes Journey template. The arrows between the units define possible paths a story can take. Step-by-step authors add details to units, such as selecting virtual characters, props that participate, and actions that modify the story state. In later work, Mehm et al. present Bat Cave, a prototyping tool for evaluation and testing DEGs [173]. Here, authors can define so-called Narrative Game-Based Learning Objects (NGBLOBs) that express a scene's learning context, gaming context and storytelling function. The narrative engine executes story descriptions in an XML format. This engine is extended to handle NGBLOBs and adapt DEGs to a model of learner knowledge. In a book chapter on serious games, Mehm gives an overview of authoring processes and tools, which includes StoryTec [174].

publication	query	publication type	research category	note
[172]	471w	workshop paper	proposal of solution	StoryTec
[173]	128n	conference paper	validation research	Bat Cave
[171]	238n	PhD thesis	evaluation research	both
[174]	152n	textbook chapter	introduction and overview	overview

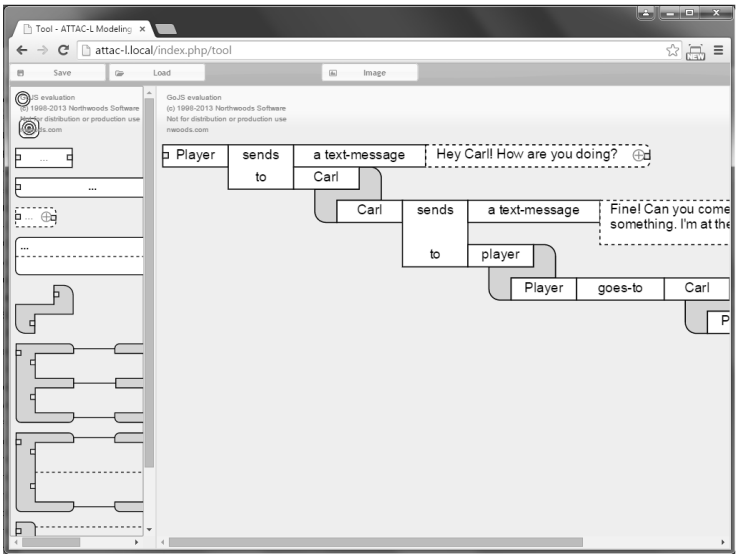


Fig. 11. Example game narrative in ATTAC-L (appears in Broeckhoven et al. [263])

Language 19 *GameDNA* *generic / tool / practice*

Van Nimwegen et al. describe Game Discourse Notation and Analysis (GameDNA), a graphical modeling language intended to develop serious games for assessment more effectively [266]. GameDNA is designed to improve visualization methods for the assessment of a player's cognitive processes and metal states during gameplay. Its notation is composed of two levels. The first describes narrative story elements that form the story plot. The second describes the discourse between the player and the system, and a players' corresponding mental actions. Modeling elements include player perceptual actions (see), mental actions (decide), physical actions (perform) and system actions (react/feedback) that are connected via triggers and loops.

publication	query	publication type	research category	note
[266]	–w	conference paper	proposal of solution	

Language 20 *ATTAC-L* *genre-specific / tool / educative*

Van Broeckhoven and de Troyer propose ATTAC-L: a visual modeling language for describing educational virtual scenarios that help prevent cyber bullying [262]. In addition, they apply controlled natural language to improve collaboration [263]. ATTAC-L helps pedagogical experts compose scenarios from *story bricks*, nouns and verbs that can be combined in sequence, as choices or as concurrent events. Figure 11 shows a screen shot of its web-based editor. Friendly Attac is a related resarch project[1].

[1] http://www.friendlyattac.be – does not share software (visited April 10th 2019)

publication	query	publication type	research category	note
[262]	15n	conference paper	proposal of solution	
[263]	206n	conference paper	proposal of solution	
[264]	693w	conference paper	validation research	
[265]	253w	conference paper	proposal of solution	
[66]	123n	book chapter	validation research	

Language 21 ***EngAGe DSL*** *generic / engine / educative*

Chaudy et al. aim to improve the effectiveness of game-based learning,. They propose an Engine for Assessment in Games (EngAGe) and a DSL that helps teachers take ownership of the feedback provided in serious games [53]. They describe the DSL as a Feature Model (see Language 95) and implement a prototype in Xtext. Features include serious game kind, player data, learning goals, feedback messages, a feedback model, and actions related to evidence and reactions. Chaudy's website gives an overview of related research[1]

[1]http://www.yaellechaudy.com (visited July 15th 2019).

publication	query	publication type	research category	note
[53]	199n	conference paper	proposal of solution	
[36]	163n	PhD thesis	evaluation research	

Language 22 ***VR-MED*** *genre-specific / tool / educative*

Mossmann et al. present a prototype of VR-MED, a visual DSL that expresses game scenarios for teaching family medicine. Developers and health-care professionals can use VR-MED for creating simple games based on textual medical cases.

publication	query	publication type	research category	note
[182]	88n	conference paper	validation research	

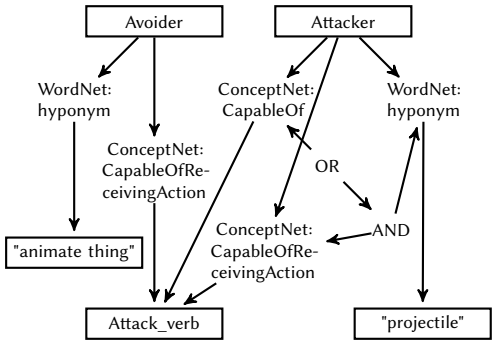
B.4 Game Mechanics

Language 23 ***Petri Nets*** *generic / engine / practice*

Petri Nets are a visual notation originally intended for describing chemical processes, which has been extensibly studied and applied to numerous other fields, including game design. Petri Nets are directed graphs with two node types: transitions (shown as bars) and places (shown as circles) that respectively model events and variable amounts of resources. The arrows between the nodes describe pre- and post-conditions of events. The operational semantics and mathematical properties of different classes of Petri Nets are well known, and there is ample tool support. Murata provides a historical introduction, comprehensive overview, and tutorial [183].
Verbrugge proposes representing Narrative Flow Graphs (NFGs) (Language 50) as Petri Nets [271]. Natkin and Vega propose describing and analyzing the non-deterministic structure of

```
noun Avoider
noun Attacker
verb Attack_verb: shoot, attack, damage,
     chase, injure, hit
constraint:
  (ConceptNet CapableOfReceivingAction ?
       Avoider ?Attack_verb)
constraint:
  (WordNet hyponym ?Avoider "animate_thing")
constraint:
  (or
    (and (WordNet hyponym ?Attacker "
         projectile")
         (ConceptNet
              CapableOfReceivingAction ?
              Attacker ?Attack_verb))
    (ConceptNet CapableOf ?Attacker ?
         Attack_verb))
```

(a) Definition of an attacker-avoidance game space



(b) Graphical view

Fig. 12. Example of a game space (Adapted from Nelson and Mateas [189])

the game narration with Petri Nets [185]. Natkin et al. propose a methodology for spatiotemporal game design with directed hypergraphs that relate missions modeled as Petri Nets to spaces for validating mission properties such as reachability [186].
Brom and Abonyi propose authoring non-linear story plots featuring intelligent virtual humans with Petri Nets [37]. Brom et al. describe a Petri Nets dialect that supports token ageing and its application in the design of Europe 2045, an on-line multiplayer strategy game for teaching high-school in economics, politics, and media studies [38]. Balas et al. extend the approach by combining timed colored Petri Nets and non-deterministic FSMs for developing Karo, a social simulation intended for teaching. Araújo and Roque describe an approach for modeling game systems and flow with Petri Nets for analyzing and simulating behaviors [16]. Ortega et al. propose Petri Nets for modeling multi-touch games systems [197].

publication	query	publication type	research category	note
[271]	–w	conference paper	proposal of solution	story plot
[185]	~~language~~	conference paper	proposal of solution	game design
[186]	118w	conference paper	proposal of solution	game design
[37]	~~language~~	workshop paper	proposal of solution	story plot
[38]	~~language~~	conference paper	validation research	story plot
[23]	~~gd~~	conference paper	validation research	story plot
[16]	207w	conference paper	proposal of solution	game design
[197]	382w	conference paper	proposal of solution	multi-touch

Language 24 ***Game Space Definitions*** *genre-specific / tool / practice*

Nelson and Mateas describe an approach for automated game design that describes game mechanics of micro-games such as news games. They use WordNet and ConceptNet for relating

design goals to *nouns and verbs* that instantiate predefined (or stock) mechanics. They demonstrate the approach by generating WarioWare-style games [188]. Extending the approach, they propose an interactive game design assistant that helps novice designers create games. Authoring and understanding happens in a mixed-initiative fashion, by alternating user-directed decisions and computer-generated suggestions [189]. Figure 12 shows an example definition of an attacked-avoider game space.

publication	query	publication type	research category	note
[188]	135w	conference paper	position paper	
[189]	328w	conference paper	proposal of solution	

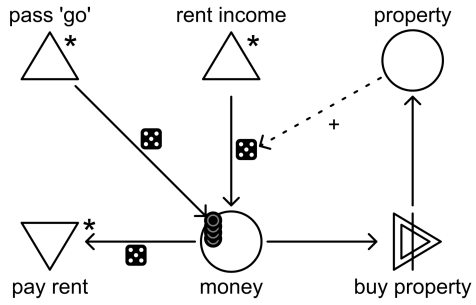
Language 25 *Biped and Ludocore* *generic / engine / practice*

Smith et al. propose a light-weight game sketching approach with computational support for a class of physical prototypes that use board-game-like elements to represent complex videogames [239]. The Biped system supports manual and automated play testing of prototypes that are formalized using the *game sketching language*, a subset of Prolog for describing the game state, player actions and state update rules. A sketch can be played as an interactive visual representation that maps specifications to board game primitives. Connected spaces and tokens permit user actions such as clicking and dragging. Designers can also analyze its properties by specifying scenarios and conditions, obtaining feedback as logical game traces from an analysis engine that leverages Answer Set Programming (ASP). The running example is DrillBot 6000. In later work, they present Ludocore, a logical game engine that formalizes the event calculus and drives Biped [235]. Smith and Mateas add a notation for pattern-based gameplay analysis [238]. They describe a design space approach that leverages ASP for PCG [234].

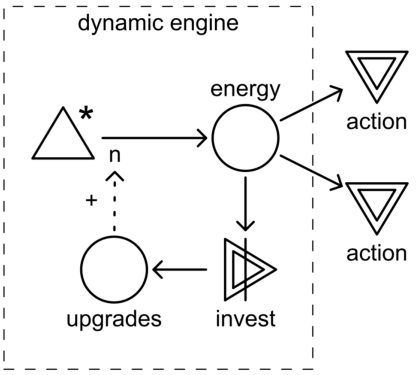
publication	query	publication type	research category	note
[190]	–w	conference paper	proposal of solution	event calculus
[239]	537w	conference paper	proposal of solution	Biped
[240]	880w	ext. abstract (demo)	proposal of solution	Biped
[236]	72n	PhD thesis proposal	proposal of solution	Biped
[235]	–w	conference paper	validation research	Ludocore
[233]	–w	conference paper	proposal of solution	ASP for PCG
[238]	52w	conference paper	proposal of solution	Ludocore
[234]	–w	journal article	validation research	ASP for PCG
[237]	494w	PhD thesis	validation research	all of the above

Language 26 *Machinations* *generic / framework / practice*

Dormans proposes the Machinations framework as a common game design vocabulary for visualizing elemental feedback loops associated to emergent gameplay [72]. Models (or diagrams) are directed graphs resembling Petri Nets (Language 23). When set in motion through play, activated nodes act by pushing or pulling economic resources along its edges. Figure 13a demonstrates a feedback loop in Monopoly, where buying property raises a player's income, which can again be invested. Figure 13b shows an example pattern from the pattern catalogue, described as a pattern language. Designers can use models for simulating and balancing games before they are



(a) A Machinations diagram of Monopoly that demonstrates a feedback loop in its economy



(b) The Dynamic Engine pattern expresses a trade-off between spending energy on long-term investment and short-term gain (action nodes)

Fig. 13. Machinations diagram and pattern (adapted from Dormans [76])

built [75], and for describing emergent physics [77]. The original Machinations tool was Flash-based and now discontinued, but still available for download together with a set of examples[1]. A commercial web-based tool that uses JavaScript is under development[2].

[1]https://machinations.io/FAQ.html – Technical 'old version' (visited April 23rd 2019)
[2]https://machinations.io (visited April 23rd 2019)

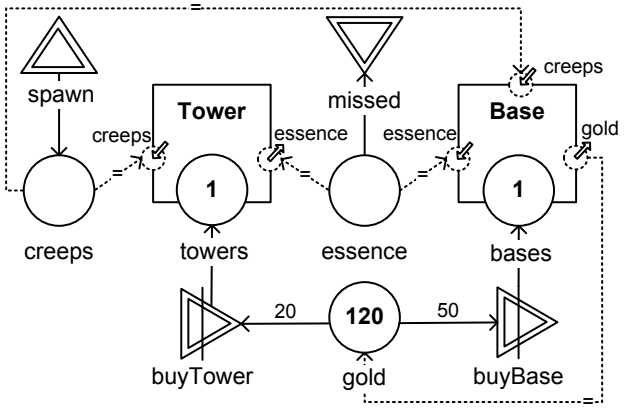
publication	query	publication type	research category	note
[72]	–w	conference paper	proposal of solution	
[74]	–w	workshop paper	proposal of solution	
[75]	–w	workshop paper	validation research	
[76]	97w	PhD thesis	validation research	
[77]	–w	workshop paper	proposal of solution	
[5]	–w	textbook	validation research	

Language 27 *Micro-Machinations*

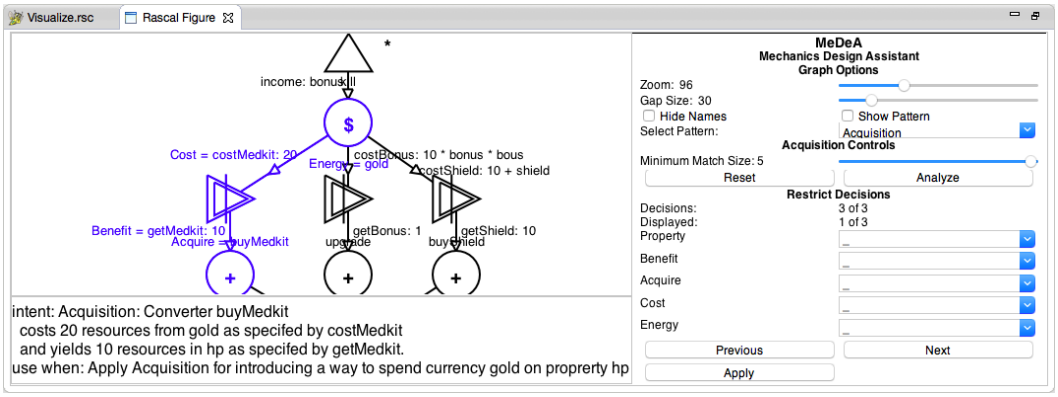
generic / engine / practice

Micro-Machinations (MM) is a textual and visual programming language, a continuation of Machinations (Language 26) that addresses several technical shortcomings of its evolutionary predecessor. Klint and van Rozen present MM, a DSL for game-economies (or game-economic mechanics) that speeds up the game development process by improving game designer productivity and design quality. MM formalizes an extended subset of Machinations features, notably adding modularity and a textual storage format. For accurately predicting a game's behavior, they provide MM Analysis in Rascal (MM AiR), a visual framework for analyzing the reachability and invariant properties[1]. In addition, van Rozen and Dormans propose a *live programming* approach for rapidly prototyping, adapting and fine-tuning game mechanics, which includes an embeddable MM library written in C++[2]. Finally, van Rozen presents a pattern-based Mechanics

(a) AdapTower is a tower defense game whose embedded mechanics can be modified at run time. Towers prevent creeps from passing and bases catch essence for buying more. (adapted from [268])



(b) AdapTower's visual Micro-Machinations definition contains Tower and Base sub-modules (adapted from [268])



(c) A Mechanics Design Assistant for pattern-based analysis and generation (adapted from [267])

Fig. 14. Micro-Machinations is an embeddable and modular script language for live programming of game mechanics whose qualities can be predicted using design patterns

Design Assistant (MeDeA) featuring pattern-based editing using an extensible and programmable *pattern palette*. MeDeA can recognize and explain patterns, and generate model extensions[3]. Figure 14 depicts an example of a game's mechanics and shows a screenshot of MeDeA. MM AiR uses the Spin model checker[4]. MM AiR and MeDeA leverage meta-programming features of Rascal[5] e.g., pattern matching and visualization. A new version of MM for live programming that is based on C#, edit scripts and novel model migration techniques is ongoing work[6].

[1]https://github.com/vrozen/MM-AiR (visited May 14th 2019)
[2]https://github.com/vrozen/MM-Lib (visited May 14th 2019)
[3]https://github.com/vrozen/MeDeA (visited May 14th 2019)

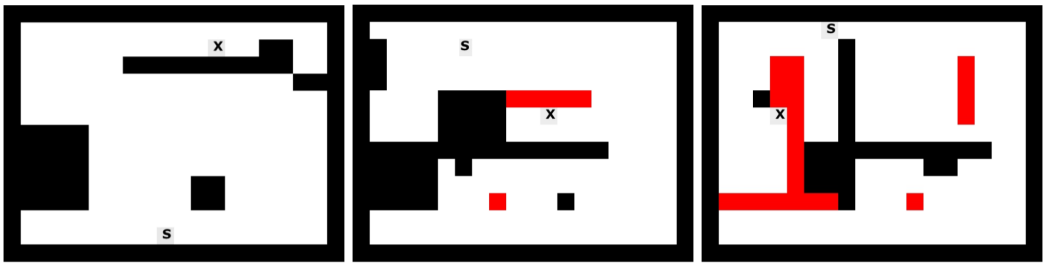


Fig. 15. Mechanic Miner: Levels for 'gravity inversion' (appears in Cook et al. [59])

[4]http://spinroot.com (visited May 14th 2019)
[5]https://www.rascal-mpl.org (visited May 14th 2019)
[6]http://livegamedesign.github.io (visited May 14th 2019)

publication	query	publication type	research category	note
[132]	508w	conference paper	validation research	MM AiR
[268]	17n	conference paper	validation research	MM Lib
[267]	76n	conference paper	validation research	MeDeA

Language 28 *Game-o-Matic* *generic / tool / practice*

Treanor et al. propose generating arcade-style videogames that represent ideas with so-called *micro-rhetorics*. Micro-rhetorics are parameterized structures with a unique id, a verb and entity roles (parameters). For instance, "`A avoids B`" consists of a subject A, a predicate B and a verb avoids. For each verb, Game-o-Matic randomly selects a representative micro-rhetoric from its library that form partial game descriptions. These are completed with recipes that modify the rules and completes the game's mechanics, adding win and lose conditions and concrete structures for player interaction. Proceduralist Readings [258] is a related framework (see Language 32).

publication	query	publication type	research category	note
[260]	~~language~~	conference paper	proposal of solution	
[259]	~~language~~	workshop paper	proposal of solution	

Language 29 *Mechanic Miner* *generic / tool / practice*

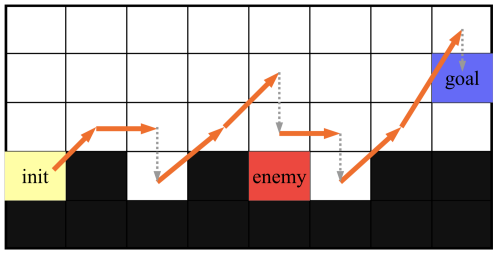
Cook et al. introduce Mechanic Miner, *"an evolutionary system for discovering simple game mechanics for puzzle platform games"* [59]. Mechanic Miner inspects and modifies the mechanics using Java reflection. Levels are generated as tile maps with accessible (white), solid (black) and deadly (red) cells. The objective is to use the mechanics and toggle effect on and off to navigate the player (s) to the destination (x). The playability of the resulting game is evaluated using a solver that attempts sequences of button presses until it finds a combination that

```
<Jump,
  {<Relative, 1, Equal(Ypos(e), Ypos(Block)
      +1)>,
   <Relative, 1, Equal(Xpos(e), Xpos(Block))
       >},
  {<Relative, 1, Update(Xpos(e), 1)>,
   <Relative, 1, Update(Ypos(e), 1)>}>
```

(a) 'Jump' mechanic where e is an entity that has x and y coordinates `Xpos(e)` and `Ypos(e)`

```
<DoubleJump,
  {<Relative, 1, Equal(Ypos(e),Ypos(Block)+1)
      >,
   <Relative, 1, Equal(Xpos(e),Xpos(Block))>,
   <Absolute,-1, Equal(Performed(Jump),e)>},
  {<Relative, 1, Update(Xpos(e),1)>,
   <Relative, 1, Update(Ypos(e),2)>}>
```

(b) 'Double Jump' requires first performing 'Jump'



(c) Platformer level with a superimposed playtrace that uses a generated mechanics (shown as arrows) and gravity (shown as dotted arrows)

Fig. 16. Generated PDDL Mechanics of a Platformer (adapted from Zook and Riedl [293])

completes a level, which must include the mechanic. Figure 15 shows an example level generated for mechanic named 'gravity inversion', which modifies how the game engine handles gravity: `INVERTSIGN player.acceleration.y;`. Other examples include , 'teleportation': `HALVE player.y;` and 'bouncing': `ADD 1 player.elasticity;` In a large user study on a selection of generated puzzle mechanics called A Puzzling Present[1], they evaluate enjoyability and difficulty, which entailed play testing followed by questionnaires [60].

[1] http://www.gamesbyangelina.org/downloads/app.html (visited August 7th 2019)

publication	query	publication type	research category	note
[59]	−w	conference paper	proposal of solution	
[60]	284w	conference paper	evaluation research	

Language 30 *Gamelan and Modular Critics* *generic / tool / practice*

Osborn et al. propose a framework for automated game design with computational support for play testing. The game definition language (Gamelan) models turn-taking games, such as board- and card games, and game design critics quantify gameplay qualities for automated analysis. Gamelan games consist of rules and procedures. Rules are side-effect-free relations that can only succeed or fail. In contrast, logical functions are expressions defined over a game's current state and can succeed with different parameter bindings. Examples of critics are: no rules should go unused, players get equal turns (unfair play), repeating similar actions should be a losing strategy (dull), and rankings of players should shift frequently over the course of the game (unsuspenseful). Core Gamelan is implemented in XSB Prolog. As a demonstration they detect known design problems in a card game called Dominion.

publication	query	publication type	research category	note
[201]	400w	conference paper	proposal of solution	

Language 31 *Planning Domain Definition Language* *generic / tool / practice*

Zook and Riedl present an approach for generating game mechanics, with special focus on turn-based domains with deterministic actions and avatar-centric mechanics. Game mechanics are expressed in a subset of the Planning Domain Definition Language (PDDL) with game-specific extensions. The first, *time-indexing* enables preconditions to refer to earlier times for expressing delayed effects. The second, *coordinate frames of reference* distinguishes global world-state from avatar-relative terms for expressing avatar perception. Figure 16 shows an example of jumping mechanics for a platform game level. They leverage ASP to generate mechanics that conform to a set of design constraints. A planner, also implemented in ASP, analyzes the playability of those mechanics. Instead of generating mechanics from scratch, they iteratively adapt and refine mechanics, favoring fewer mechanics. The approach is demonstrated by generating combat mechanics of a role-playing game, and avatar-mechanics of a platform game, and a hybrid of the two.

publication	query	publication type	research category	note
[294]	–w	workshop paper	proposal of solution	
[293]	114w	conference paper	proposal of solution	

Language 32 *Sygnus and Gemini* *generic / tool / practice*

Summerville et al. propose Gemini, a system for analysis and generation of a game's mechanics [247]. They formalize Proceduralist Readings [258], a framework for interpreting and understanding the meaning of games by providing rhetorical affordances. Sygnus is a language based on ASP predicates that not only describes mechanics, processes and interactions, but also aesthetic and cultural expressions. As a result,'meaning' can be directly derived from the source code. The Gemini system statically analyzes Sygnus programs, producing readings as reasoning chains.

publication	query	publication type	research category	note
[258]	–w	conference paper	proposal of solution	
[247]	173n	journal article	proposal of solution	Gemini

B.5 Virtual Worlds and Levels

Language 33 *Semantic Scene Description Language* *generic / engine / practice*

Tutenel et al. propose a Semantic Scene Description Language for guiding how generators produce consistent and meaningful content [261]. Using its visual notation, designers can express abstract *scene classes*, descriptions of scenes consisting of objects (or components), the relationship between them, and time- and context-specific variations, e.g., dining area, office, street, dungeon, forest, etc. Concrete scenes are situated instances whose constraints are solved and generated as an integral part of a larger whole. Kessing et al. propose Entika, a framework for designers that offers an editor for authoring semantic game worlds and an engine for semantic layout solving [130].

publication	query	publication type	research category	note
[261]	~~gd~~	conference paper	proposal of solution	
[130]	–w	workshop paper	validation research	Entika

Language 34 *SketchaWorld* *generic / tool / practice*

Smelik et al. aim to simplify modeling virtual worlds by combining semantics-based modeling and PCG techniques in a declarative modeling approach [231]. SketchaWorld is a tool for designers for rapidly sketching 3D worlds, which integrates two novel techniques: interactive procedural sketching and virtual world consistency maintenance.

publication	query	publication type	research category	note
[232]	~~language~~	workshop paper	proposal of solution	
[231]	–w	journal article	validation research	

Language 35 *Tanagra* *genre-specific / engine / practice*

Smith et al. describe Tanagra, a *mixed-initiative* level design tool for 2D platformers. In response to changes to the pacing of the level generates levels with corresponding "beat patterns" (sequences of obstacles) and verified playability, using constraint programming and reactive planning, Tanagra integrates reactive planning language ABL (Language 39) and numerical constraint solving.

publication	query	publication type	research category	note
[243]	–w	conference paper	proposal of solution	
[244]	783w	extended abstract	tool demonstration	
[241]	–w	journal article	proposal of solution	

Language 36 *Ludoscope* *generic / engine / practice*

Lindenmayer systems (or L-systems) are generative grammars that were originally intended for describing plant growth patterns [142] and are now also used for PCG. Dormans investigates strategies for generating levels for action adventure games, and proposes mission and spaces as two separate structures. He analyzes a Zelda game level, and generates its missions and spaces using transformative grammars [73]. Ludoscope is a tool for designing procedurally generated game levels based on these principles. In Ludoscope, level transformation pipelines step-by-step transform level content, gradually adding detail, dungeons, enemies, encounters, missions, etc. These pipelines consist of grammar rules that work on content represented as tile maps, graphs, Voronoi Diagrams and Strings. Karavolos et al. explore applying Ludoscope in the design of two distinct pipelines that generate dungeons and platform levels [129]. Van Rozen and Heijn propose two techniques for analyzing the quality of level generation grammars called MAD and SAnR (see Language 58).

publication	query	publication type	research category	note
[73]	–w	workshop paper	proposal of solution	grammars
[78]	–w	journal article	proposal of solution	grammars
[74]	–w	workshop paper	proposal of solution	grammars
[79]	~~language~~	workshop paper	validation research	Ludoscope
[129]	~~language~~	conference paper	validation research	Ludoscope
[269]	–n	workshop paper	proposal of solution	Ludoscope Lite

Language 37 *The Sentient Sketchbook* *genre-specific / tool / practice*

Liapis et al. introduce The Sentient Sketchbook, a tool intended to support level designers in rapidly creating abstract game levels, which represents levels as low-resolution tile map sketches. The tool supports a mixed-initiative design and refinement process, allowing designers to choose level suggestions generated using genetic algorithms. The running example discusses a strategy game where players control a base and require resources to build units. Its tile maps consist of tiles that are passable, impassable, player bases or resources. Designers can analyze the maps for playability and visualize gameplay properties using built-in metrics that calculate properties such as navigable space, resource safety, safe areas and unused space. The tool is available on its website as a Java application or an online version, along with a list of related publications[1].

[1] http://www.sentientsketchbook.com (visited August 12th 2019)

publication	query	publication type	research category	note
[139]	~~language~~	conference paper	evaluation research	

Language 38 *Evolutionary Dungeon Designer* *genre-specific / tool / practice*

Baldwin et al. present the latest iteration of the Evolutionary Dungeon Designer (EDD), a mixed-initiative level design tool that assists level designers in collaboratively creating game content [25]. EDD uses evolutionary search algorithms and patterns for generating dungeon

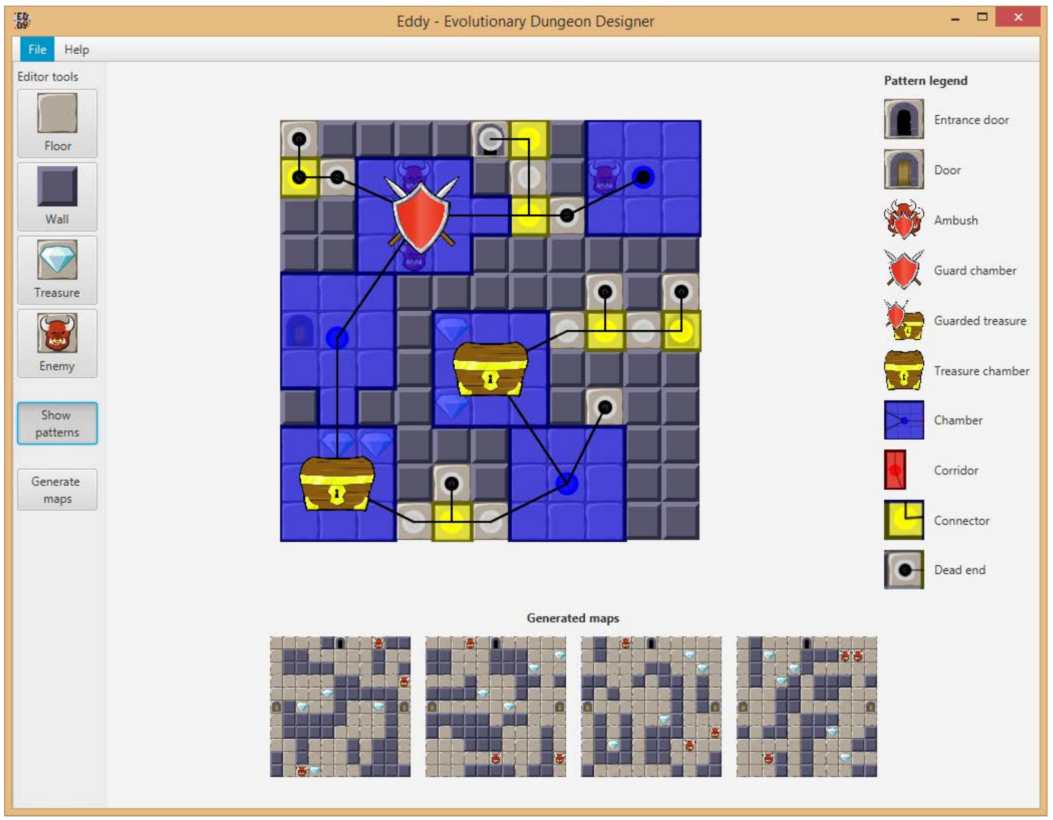


Fig. 17. Room editing view with pattern overlay (appears in Baldwin et al. [24])

levels with desirable properties. EDD detects patterns in levels and displays on instances superimposed on the tile map, as shown in Figure 17. Spacial micro-patterns consist of paths and multiple tiles: corridor (red tiles), connector (yellow tile) and chamber (blue tiles). Inventorial micro-patterns placed on one tile are door, treasure and enemy. Meso-patterns are composed from spacial combinations of micro-patterns. Examples, each shown using descriptive icons, are: treasure chamber, guard chamber, ambush, dead end, and guarded treasure. Meso-patterns are detected using breadth first search of a pattern graph, starting at a room's entrance. The level quality is estimated with several fitness functions that guide the search.

publication	query	publication type	research category	note
[24]	–w	workshop paper	validation research	
[25]	206w	conference paper	validation research	

```
joint sequential behavior OfferDrink(){
  team Trip, Grace;
  // individual behavior for initial
       offer subgoal
  with (post-to OfferDrinkMemory)
    iInitialDrinkOffer();
  subgoal
    iLookAtPlayerAndWait(0.5);
  with (synchronize) subgoal
    jSuggestMartini();
  // react to Grace's line about fancy
       shakers
  with (synchronize) subgoal
    jFancyCocktailShakers();
}
```

(a) Trip's *'offer drink'* behavior

```
joint sequential behavior OfferDrink(){
  team Trip, Grace;
  // wait for Trip's first line
  with (success_test
    { OfferDrinkMemory
      (CompletedGoalWME
       name == iInitialDrinkOffer
       status == SUCCEEDED
    )}) wait;
  subgoal
    iLookAtPlayerAndWait(0.5);
  // react to Martini suggestion
  with (synchronize) subgoal
    jSuggestMartini();
  with (synchronize) subgoal
    jFancyCocktailShakers();
}
```

(b) Grace's *'offer drink'* behavior

(c) Trip and Grace in Façade

Fig. 18. ABL scripts adapted from Mateas and Stern [162] (a, b) and [165] (c)

B.6 Behaviors

Language 39 ***ABL*** *generic / engine / practice*

Mateas and Stern describe A Behavior Language (ABL), pronounced "able", a reactive planning language for authoring believable agents expressing rich personality built on Hap [162]. ABL extends Hap with atomic behaviors, reflection, private memories and goal spawning. ABL was notably used in the interactive drama Façade[1] [164, 165] and Tanagra, which is described as Language 35. Figure 18 shows example scripts that demonstrate synchronization with individual `'i'` and joint `'j'` subgoals. Simpkins et al. extend ABL (as A^{2}BL) with reinforcement learning [229]. Its Java sources are released under the GNU GPL[2].

[1] https://www.playablstudios.com/facade/ (visited May 9th 2019)
[2] https://www.cc.gatech.edu/~simpkins/research/afabl/abl.html (visited May 9th 2019)

publication	query	publication type	research category	note
[162]	~~gd~~	journal article	proposal of solution	ABL
[164]	63w	conference paper	philosophical paper	ABL
[165]	773w	conference paper	proposal of solution	ABL
[229]	–w	conference paper	proposal of solution	A^{2}BL
[230]	–w	journal article	proposal of solution	A^{2}BL, reprint

Language 40 ***Simple Entity Annotation Language*** *generic / engine / practice*

Anderson proposes a Simple Entity Annotation Language (SEAL), a behavior language that resembles C for scripting believable NPC behaviors with domain-specific features and datatypes, e.g., state machines. Figure 19 shows an example script with two entities, states, events, and associated behavior functions.

```
entity defender {
  scalar manGun;
  scalar turret;
  scalar gunAvailable = 0;

  event unused {
    manGun = getGlobal("use");
    if(manGun!=NULL) {
      turret=getEntity(manGun);
      gunAvailable = 1;
      trigger lock @ turret;
    }
  };

  state fsm {
    patrolling(), NULL;
    defending(), patrolling();
    fsm(), NULL;
  };

  event enemy_detected {
    setstate fsm::defending;
  };

  fsm::patrolling() {
    while(1) {
      /* execute 'patrolling'
           behaviour */
      ...
    }
  }

  fsm::defending {
    /* if gun-turret available */
    if(gunAvailable){
      /* man gun turret */
      manGun();
      trigger unlock @ turret;
      gunAvailable = 0;
      ...
    } else {
      /* execute default defence
           behaviour */
      ...
    }
  }

  fsm::fsm() {
    setstate patrolling;
  }

  defender() {
    setstate fsm;
  }
};
```

(a) Defender entity

```
entity turret {
  event lock {
    /* stop advertising */
    setSilent();
  };

  event unlock {
    /* advertise */
    setBroadcast();
  };
  ...

  global void use() {
   /* action to fire the gun */
   action fire();

   ...
   fire();
   ...
  }
  ...
  turret(){}
};
```

(b) Turret entity

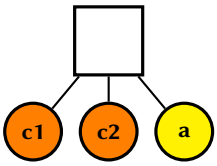
Fig. 19. SEAL script of a defender manning a turret (adapted from Anderson [14]

publication	query	publication type	research category	note
[15]	20n	conference paper	proposal of solution	
[14]	247n	PhD thesis	proposal of solution	

Language 41 *BEcool* *generic / engine / practice*

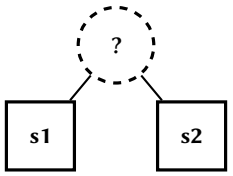
Szilas presents BEcool, a behavior engine for authoring high performance expressive virtual agents. Behaviors are directed graphs with nodes for animations and arrows for transitions, arrows' labels for environment's sensing (events) and dashed arrows for event-based animation triggering. BEcool supports sequencing, branching, parallelism and inter-characters behaviors [249].

publication	query	publication type	research category	note
[249]	~~gd~~	conference paper	proposal of solution	



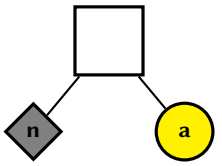
A sequence is a node, shown as a rectangle, whose success depends on each of its child nodes succeeding in sequence, where any failure ends its execution in failure. Here, conditions c1 and c2 must succeed for action a to happen.

(a) Sequence node



A selector is a node shown as a dashed circle whose success depends on trying to execute its child nodes one-by-one until one succeeds, ending the selection in success. Here, sequences s1 and s2 are alternative plans.

(b) Selector node



A lookup node, shown as a diamond, serves as a modular construct for looking up a sub-tree by name. Here, if action a happens depends on the success of the sub-tree represented by lookup n.

(c) Lookup node

Fig. 20. Behavior Trees (visual variant adapted from Champandard [50, 51])

Language 42 ***Behavior Trees*** *generic / engine / practice*

Behavior Trees (BTs) is a visual notation for authoring AI behaviors that is said to be understandable, easy to use, and scale well in parallel [50, 126, 127]. Behaviors are modeled by hierarchies of nodes that represent plans whose details are specified in a top-down fashion. More general plans appear near the top, whereas the leafs at the bottom represent conditions and "atomic" actions exposing lower-level logic such as moves. Figure 20 shows three node types: sequence, selector and lookup, which can be composed with conditions and actions in authoring complex modular behaviors.
Lim et al. apply evolutionary techniques in developing competitive AI-bots that can play video games, in particular for the real-time strategy game DEFCON. Martens et al. present a formal operational semantics, a type system and an implementation [159]. Variants of BTs are commonly used in practice, e.g., in Halo 2 and Spore [50, 127], and several implementations and engine plugins are available, e.g., BTs are a built-in feature of Unreal Engine 4[1].

[1]https://docs.unrealengine.com/en-us/Engine/AI/BehaviorTrees (visited May 9th 2019)

publication	query	publication type	research category	note
[127]	–	presentation (audio)	practice	
[126]	~~language~~	article	experience report	
[50]	–	presentation (video)	tutorial / practice	
[141]	~~language~~	conference paper	validation research	
[51]	–	presentation (video)	tutorial / practice	
[159]	–w	unpublished	validation research	

Language 43 ***Behavior Transition Networks*** *generic / engine / practice*

Fu et al. describe a visual framework for designers, developers and subject-matter experts that simplifies authoring behavior as Behavior Transition Networks (BTNs), an extension of finite state machines [93]. In addition to current states and transitions, BTNs support hierarchical decomposition, variables, communication to other BTNs and code invocation. Figure 21 shows an example aimed at specifying realistic behavior of NPCs in a first person shooter. SimBionic is

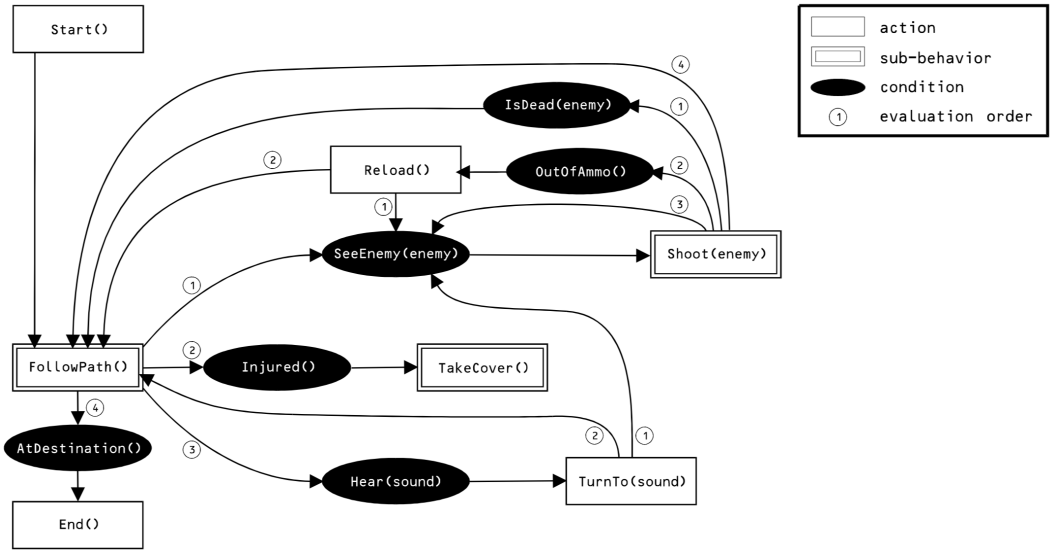


Fig. 21. Behavior Transition Network of combat patrol behavior (appears in Fu et al. [93])

a visual editor and a run-time engine for embedding behaviors [94] that is available under the 3-clause BSD licence[1].

[1]http://www.simbionic.com/ (visited March 28th 2019)

publication	query	publication type	research category	note
[92]	–w	journal article	proposal of solution	BrainFrame
[93]	~~gd~~	conference paper	proposal of solution	BTNs
[94]	~~gd~~	conference paper	proposal of solution	Simbionic

Language **44** ***POSH#*** *generic / engine / practice*

Gaudl describes POSH# a C# framework for creating behavior-based AI for robust and intuitive agent development[1] ABODEstar is an IDE for behavior oriented design.

[1]https://github.com/suegy/posh-sharp (Visited November 24th 2018)

publication	query	publication type	research category	note
[104]	–w	conference paper		POSH
[105]	–w	conference paper		POSH, ABL
[103]	274n	PhD thesis		POSH

```
<!ELEMENT condition (%basic-condition;|either
    )+>
<!ELEMENT active EMPTY>
<!ATTLIST active flag NMTOKEN #REQUIRED>
<!ELEMENT inactive EMPTY>
<!ATTLIST inactive flag NMTOKEN #REQUIRED>
<!ELEMENT either (%basic-condition;)+>
<!ENTITY %basic-condition "(active|inactive)"
    >
```

(a) Condition Description

```
<condition>
  <active flag="FirstTaskInitiated"/>
  <either>
    <inactive flag="UsedSandSack1Container"/>
    <inactive flag="UsedSandSack2Container"/>
  </either>
<condition>
```

(b) Condition Example

```
<!ELEMENT effects ((activate|consume-object|
    speak-player|speak-char)*,trigger-
    cutscene?)>
<!ELEMENT activate EMPTY>
<!ATTLIST activate flag NMTOKEN #REQUIRED>
<!ELEMENT consume-object EMPTY>
<!ELEMENT speak-player (#PCDATA)>
<!ELEMENT speak-char (#PCDATA)>
<!ELEMENT trigger-cutscene EMPTY>
<!ATTLIST trigger-cutscene idTarget IDREF #
    REQUIRED>
```

(c) Effects Description

```
<effects>
  <speak-player>Aaaahhhh!!!</speak-player>
  <activate flag="PlayerDamaged"/>
  <trigger-cutscene idTarget="Ambulance"/>
</effects>
```

(d) Effects Example

```
<!ELEMENT resources (condition?, asset+)>
<!ATTLIST resources id ID #IMPLIED>
<!ELEMENT asset EMPTY>
<!ATTLIST asset type CDATA #REQUIRED uri
    CDATA #REQUIRED>
```

(e) Resources Description

```
<resources>
  <asset type="image/jpeg" uri="images/
      backround1.jpg"/>
  <asset type="audio/mpeg" uri="sounds/working1
      .mp3"/>
</resources>
```

(f) Resources Example

Fig. 22. <e-Game> Examples (adapted from Moreno-Ger et al. [179])

Language **45** ***DSL for AI in real-time*** *generic / tool / practice*

Hastjarjanto et al. introduce a DSL for modeling the decision making process of the AI in real-time video games, an embedded DSL in Haskell.

publication	query	publication type	research category	note
[114]	138n	workshop paper	proposal of solution	
[113]	155n	Master's thesis	proposal of solution	

B.7 Narratives and Storytelling

Language 46 <e-Game> *genre-specific / engine / educative*

Moreno-Ger et al. introduce <e-Game>, a textual DSL for describing storyboards of adventure games [178, 179], which is extended and applied for game based learning [45, 180]. <e-Game> and <e-Adventure> have an operational semantics, which enables formal analysis and supports model checking [181]. They describe the structure of storyboards using XML Schemas, which are abbreviated as Document Type Definitions (DTDs). Figure 22 shows DTDs of conditions (a), effects (c) and resources (e) and <e-Game> examples (b, d and f). The language also describes scenes, objects, characters, conversations and actions in a similar way. Marchiori et al. provide a Writing Environment for Educational Video games (WEEV), a visual language and tool for educational adventure game authoring that builds on prior work.
The <e-Adventure> project web site[1] refers to SourceForge[2] for the distribution, which includes several games and Java sources, released under LGPL.

[1]http://e-adventure.e-ucm.es (visited January 7th 2019)
[2]https://sourceforge.net/projects/e-adventure/ (visited January 7th 2019)

publication	query	publication type	research category	note
[178]	11n	conference paper	proposal of solution	<e-Game>
[179]	125w	journal article	proposal of solution	<e-Game>
[180]	220n	conference paper	proposal of solution	<e-Adventure>
[45]	227n	journal article	proposal of solution	<e-Adventure>
[181]	122n	journal article	validation research	<e-Adventure>
[155]	126n	journal article	proposal of solution	WEEV

Language 47 ScriptEase *genre-specific / tool / practice*

McNaughton et al. propose ScriptEase, a tool for game designers intended to reduce the effort in defining complex AI behaviours for Role Playing Games [168]. ScriptEase is an IDE for pattern-based script design that offers drop-down lists, checkboxes, etc. for pattern instantiation, adaptation and use. For instance, the Guard pattern specifies a guard, a guarded object and a list of situations, which consists of a name, conditions and actions. Other patterns include Patrol and Encounter.
Cutumisu et al. propose four metrics to evaluate the effectiveness of pattern catalogues [64], and Carbonaro et al. evaluate ScriptEase in teaching [49]. Schenk et al. present ScriptEase II, which adds support for game-specific generators and a drag-and-drop interface that simplifies the story component [228]. Figure 23 shows a screen shot of the prototype. Its Java sources are available on GitHub[1]. The main example and generator target is Neverwinter Nights, a game developed at Bioware.

[1]https://github.com/UA-ScriptEase/scriptease (visited March 21st 2019)

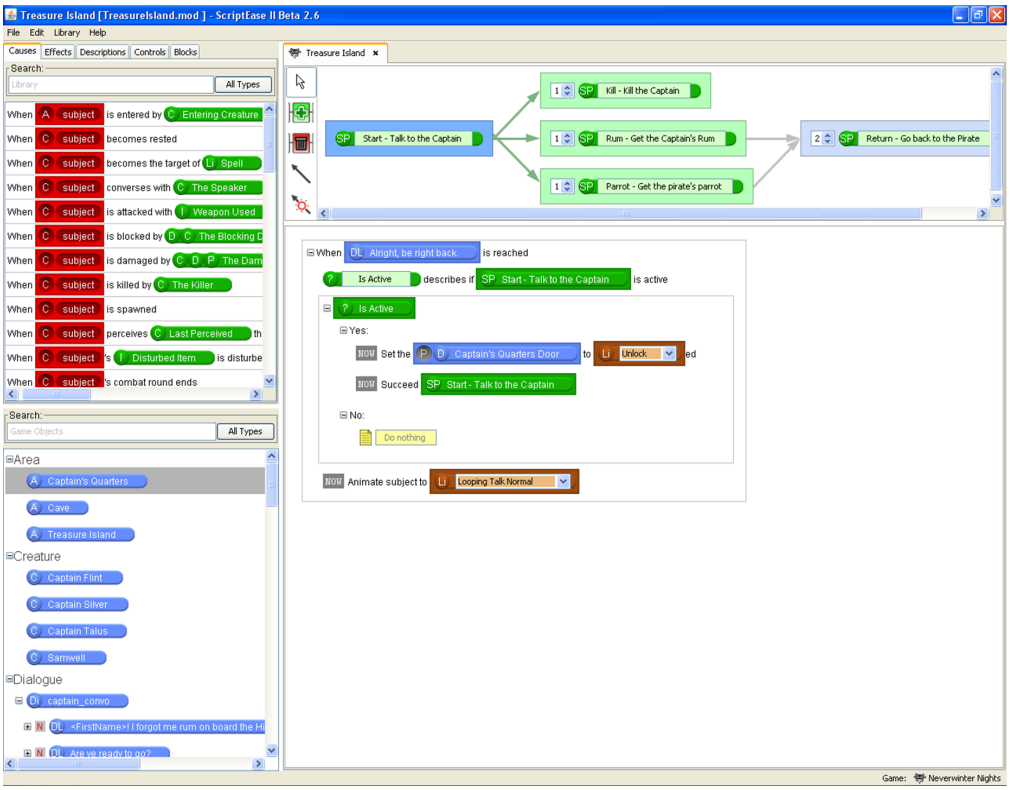


Fig. 23. ScriptEase II – Treasure Island story (appears in Schenk et al. [228])

```
do/flirt/conflict
: eros Flirter Flirtee * eros Other Flirtee
  -o {eros Flirter Flirtee * eros Flirtee Flirter anger Other Flirter * anger Other
      Flirtee}.
```

Fig. 24. Narrative action that expresses the effect of flirting in Linear Logic (adapted from Martens et al. [158])

publication	query	publication type	research category	note
[168]	–w	conference paper	proposal of solution	ScriptEase
[170]	–w	conference paper	proposal of solution	ScriptEase
[169]	~~language~~	tool demo	proposal of solution	ScriptEase
[64]	–w	conference paper	validation research	ScriptEase
[63]	145n	journal article	evaluation research	ScriptEase
[49]	–w	journal article	evaluation research	ScriptEase
[62]	–w	PhD thesis	evaluation research	ScriptEase
[228]	–w	conference paper	proposal of solution	ScriptEase II

Language 48 *Ceptre* *generic / tool / practice*

Martens et al. investigate the use of Linear Logic (LL) programming for expressing story worlds, settings in which the effect of narrative actions may create emergent behaviors [158]. Narrative actions modify story states consisting of logical predicates. Figure 24 shows a rule that expresses how flirting between a Flirter and a Flirtee may result in an angry lover (Other). Describing interactive stories using LL enables an interpretation of stories as logical proofs. To demonstrate this, the authors describe an example (dramatic) story world in the language Celf, and discuss how the approach enables generation, analysis, and interactive interpretation of stories.
Martens et al. present Ceptre, a language for rapid prototyping of experimental game mechanics that builds on prior work [157]. Game designers and researchers can use Ceptre to create, analyze and debug 'core systems' and relate logical proofs to gameplay. Ceptre adds interactivity and modules called *stages* for structuring independent components. Stages run until no more actions are available (a quiescence state) allowing a transfer of control to another stage. They present two case studies. The first is an updated interactive drama. The second specifies actions and effects of a dungeon-crawler-like game. Ceptre and a tutorial are available on Github[1].

[1] https://github.com/chrisamaphone/ceptre-tutorial (visited August 14th 2019)

publication	query	publication type	research category	note
[158]	~~gd~~	workshop paper	proposal of solution	linear logic
[157]	147w	conference paper	proposal of solution	Ceptre

Language 49 *SAGA* *genre-specific / engine / practice*

Beyak and Carette describe SAGA, a DSL for story management meant to augment the productivity of artistic teams who create multi-platform narrative-driven RPGs [30, 31]. SAGA (Story as an Acyclic Graph Assembly) describes story states and transitions as graphs. A meta-language called AbstractCode is simplifies translating SAGA programs to different target platforms, e.g., C++, C# and Java. Figure 25 shows an example story graph and its associated story description. The Haskell implementation of SAGA is available online[1].

[1] http://www.cas.mcmaster.ca/~carette/SAGA/ (visited August 14th 2019)

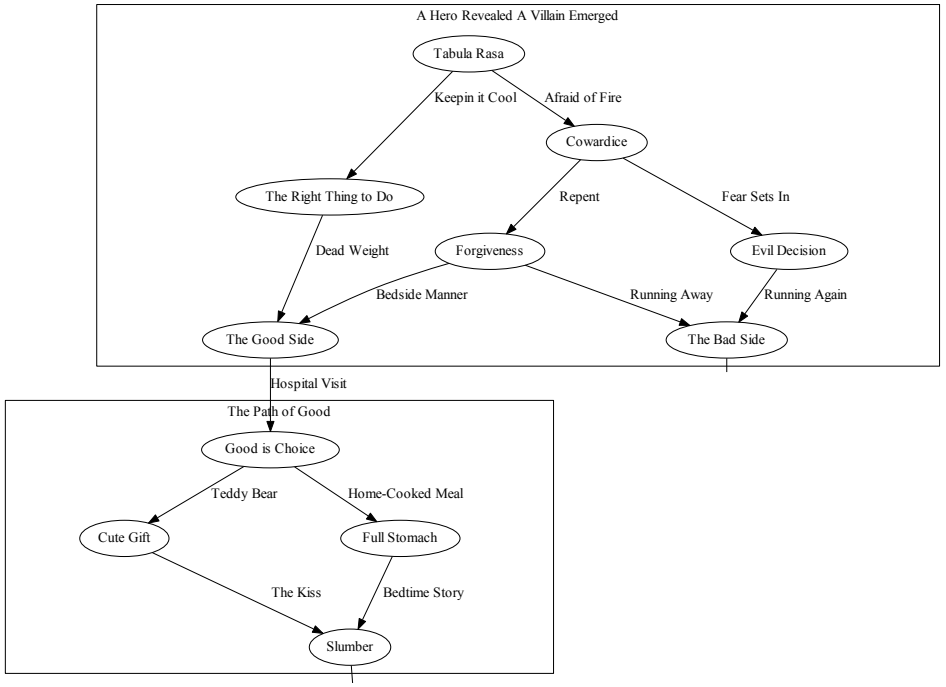
publication	query	publication type	research category	note
[31]	51n	conference paper	proposal of solution	
[30]	66n	Master's thesis	proposal of solution	

Language 50 *(P)NFG* *genre-specific / tool / practice*

Picket et al. address a lack of tool support for expressing *computer game narratives*, and in particular resolving the inherent logical consistency and playability issues [210]. They present a textual language and an environment for programming and analyzing structured narratives and assuring good narrative properties. This language, Programmable NFG, is based on Narrative Flow Graphs (NFGs). Since NFGs are a class of 1-safe Petri Nets (Language 23), the authors can

```
STORY Sealed Fate

  INITIAL Tabula Rasa

  SECTION A Hero Revealed A Villain Emerged {
    Tabula Rase GOES The Right Thing to Do WHEN Keepin'
        it Cool,
    Tabula Rasa GOES Cowardice WHEN Afraid of Fire,
    The Right Thing to Do GOES The Good Side WHEN Dead
        Weight,
    Cowardice GOES Forgiveness WHEN Repent,
    Cowardice GOES Evil Decision WHEN Fear Sets In,
    Forgiveness GOES The Good Side WHEN Bedside Manner,
    Forgiveness GOES The Bad Side WHEN Running Away,
    Evil Decision GOES The Bad Side WHEN Running Again
  }

  SECTION The Path of Good {
    Good is Choice GOES Cute Gift WHEN Teddy Bear,
    Good is Choice GOES Full Stomach WHEN Home-Cooked
        Meal,
    Cute Gift Goes Slumber WHEN The Kiss,
    Full Stomach GOES Slumber WHEN Bedtime Story
  }

  ...
```

(a) Story Description



(b) (Partial) Story Graph

Fig. 25. SAGA excerpt of *"Sealed Fate"* (adapted from Beyak and Carette [31])

```
object cloak { }

room closet {
  state {lit, locket}
}

room you {
  counter {lives 0 3 }
}
```

(a) Declaring an object, room with two states and a lives counter

```
room lighthousefront {
(you,go,north) {
  "You are now on the mountain pass.";
  move you from lighthousefront to
       mountainpass;
}
(you,go,east) {
  "You are now behind the lighthouse.";
  move you from lighthousefront to
  lighthouseback;
  ...
}
```

(b) Room-specific Actions

Fig. 26. (P)NFG code snippets (adapted from Pickett and Verbrugge [210])

```
"Behold... THE PHONE BOOTH GAME!"

        words (objects)
phone           0 1
telephone       1
"rotary phone" 2

        pre action
        "look phone"  o?phone  m=\
"The phone is a robust contraption with a
     rotary dial."
```

(a) `.misc` file, containing location-independent code

```
#1      Telephone Booth
        You are in a telephone booth.
        exit    2

#2      Outside Telephone Booth
        You are not in a telephone booth.
        enter   1
```

(b) `.wrld` file, containing location-dependent code

Fig. 27. Text game featuring a phone booth in Wander (adapted from Aycock [20])

leverage widely available techniques for formal analysis. (P)NFG's interactive narrative interpreter (and runtime) analyzes NFGs using the symbolic model checker NuSMV[1]. (P)NFG has specific statements for `object`, `state`, `room` and more general ones such as `if`, `for` and `thread`. Figure 26 shows examples. As a demonstration, they model several IF games and annalyze narrative properties.

[1]http://nusmv.fbk.eu (visited August 17th 2019)

publication	query	publication type	research category	note
[271]	–w	conference paper	proposal of solution	NFG
[210]	–w	conference paper	proposal of solution	(P)NFG
[272]	–w	conference paper	validation research	(P)NFG

Language 51 *Wander* *genre-specific / tool / practice*

In "Retrogame Archeology", Aycock shares an excerpt of Wander, an early example of a DSL for textual adventures and 'number games' from 1974 that ran on mainframe [20]. Figure 27 shows a game featuring a phone booth.

```
Location(HM,room)
Location(afrikaaner,room2)
Location(casion-guard,room2)
Fire-Alarm(floor7,on)
```

(a) Current World State

```
Action:
  Disguise-As(HM,casino-guard)
Pre-Conditions:
  Inventory(HM,casino-vest)
Add-List:
  Disguised(HM,casino-guard)
Delete-List:
  Disguised(HM,none)
  Disguised(HM,afrikaaner)
  Disguised(HM,casino-staff)
```

(b) Disguise Action

```
Panel Template:
  HM-Disguising
Actors Layer:
  Disguised(HM,casino-guard)
Environment Layer:
  Location(HM,room)
Atmosphere  Layer:
  Fire-Alarm(floor7, on)
```

(c) Disguising Panel Template Instantiation

Fig. 28. Example template instantiation from an action and a current state (adapted from Pizzi et al. [212])

publication	query	publication type	research category	note
[20]	186n	book chapter	historical account	

Language 52 *Storyboards and STRIPS*

genre-specific / tool / practice

Pizzi et al. propose an authoring tool to allow game designers to formalize, visualize, modify and validate game level solutions in the form of automatically generated storyboards [211, 212]. First, AI programmers represent game worlds as a set of propositions, characterizing its *planning domain*, based on the input provided by game designers. Game states are conjunctions of propositions and transitions between states, or *planning operators*. These are represented using a STanford Research Institute Problem Solver (STRIPS)-like formalism. Operators are categorized according to different styles which can be used in the solution generation, which is the second step. The tooling supports two modes. In the off-line mode a level designer select the style and the heuristics planner generates a complete storyboard, if one exists, depending on the constraints. In the on-line mode the level designer can simulate the level (plan) step-by-step and explore alternatives. The results are visualized in a solution tree. They validate the approach on a design of a game called Hitman. Figure 28 shows (a) an example world state; (b) a disguise action; and (c) a template instantiation, which is used to generate an image in the storyboard image sequence.

publication	query	publication type	research category	note
[211]	~~language~~	conference paper	proposal of solution	
[212]	~~language~~	journal article	validation research	

Language 53 *Versu and Praxis*

genre-specific / engine / practice

Versu is a text-based interactive drama and storytelling system that simulates autonomous agents. Praxis is a DSL for describing social practices as reactive plans that provide affordances to the agents who participate in them. Prompter is an environment for authoring Praxis that speeds up the content creation process. Figure 29 shows an example of two agents greeting each other. Applying exclusion logic, the '!' operator expresses that variables can only have

```
process.greet.X(agent).Y(agent)
  action "Greet"
    preconditions
      // They must be co-located
      X.in!L and Y.in!L
    postconditions
      text "[X] says 'Hi' to [Y obj]"
end
```

Fig. 29. The social convention of greeting in Praxis (adapted from Evans and Short [83])

one value. In the example, agents X and Y must both be at location L. This is a more concise notation than in STRIPS (see Language 52) or PDDL (see Language 31).

publication	query	publication type	research category	note
[83]	–n	journal article	experience report	

Language 54 ***Tracery*** *generic / engine / practice*

Compton et al. present Tracery, a language and tool for authoring stories and art using generative grammars. Its users have created a wide variety of creative generators, e.g., visual patterns, poetry, Twitter bots and games [56]. Specifications are JavaScript Object Notation (JSON) objects consisting of rewrite rules that express how strings of characters can be produced. Features include recursion and storing results. Figure 30 shows an example of a "Hero and Pet" story. Tracery is implemented in JavaScript. Other versions support platforms such as Python and Ruby. Users (also non-programmers) can build generators using an online visual interactive authoring environment[1]

[1] tracery.io (visited March 29th 2019)

publication	query	publication type	research category	note
[56]	–w	conference paper	validation research	

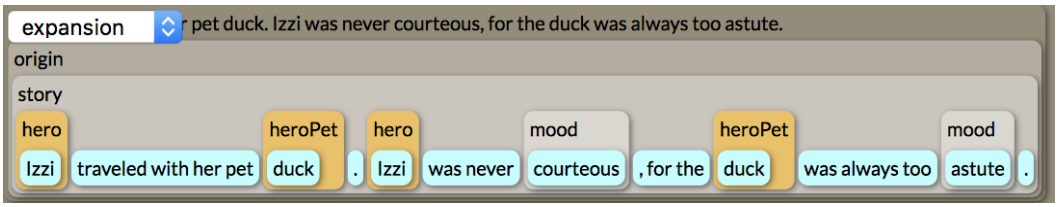
B.8 Analytics and Metrics

Language 55 ***Gameplay Metrics*** *generic / framework / practice*

Canossa and Drachen propose adopting the 'personas' framework for improving player experiences. They add *gameplay metics*, patterns of player behaviors, to the model for analyzing how different player categories use game mechanics, e.g., by measuring the amount of jumping, solving and shooting. Designers can use gameplay metrics to compare player behaviors to their gameplay hypotheses, and use the values to estimate player engagement. The use of data in games has evolved rapidly. In their book, Game Analytics, El-Nasr et al. provide an overview for a wide audience [184].

```
{
  "name":   ["Yuuma","Darcy","Mia","Chiaki","Izzi","Azra","Lina"],
  "animal": ["eagle","owl","lizard","zebra","duck","kitten"],
  "mood":   ["impassioned","wistful","astute","courteous"],
  "story":  ["#hero# traveled with her pet #heroPet#.
              #hero# was never #mood#, for the #heroPet# was always too #
                  mood#."],
  "origin": ["#[hero:#name#][heroPet:#animal#]story#"]
}
```

(a) Tracery grammar of *"Hero and Pet"* in JavaScript Object Notation



(b) Example story output string and production

Fig. 30. Tracery example of *"Hero and Pet"* stories (adapted from a tutorial on tracery.io)

publication	query	publication type	research category	note
[48]	142w	conference paper	proposal of solution	gameplay metrics
[184]	–	textbook	multiple categories	game analytics

Language 56 *Launchpad* *genre-specific / engine / practice*

Smith et al. describe Launchpad, a level generator for 2D platform games that can generate a wide variety of game levels. These levels are varied by adjusting rhythm parameters and level geometry [245]. The expressive range of a level generator is analyzed using two metrics that measure quantities of level structure associated with its qualities. The first, *linearity*, measures the aesthetic 'profile' of generated levels by fitting a straight line to the level (under an optimal angle), and calculating to what extent the geometry fits that line. The second, *leniency*, measures how forgiving a level is to player mistakes by aggregating a score of level elements. Gaps, enemies and falls: `-1`. Springs and stompers: `-0.5`. Moving platforms: `+0.5` and jumps with no associated gaps `+1.0`.

publication	query	publication type	research category	note
[242]	~~language~~	conference paper	proposal of solution	rhythms
[245]	~~language~~	journal article	validation research	Launchpad

```
obj.move(forward, 1)
obj.move(forward, 1, duration=3)
obj.move(forward, 1, speed=4)
obj.move(forward, speed=2)
# change of coordinate system
obj.move(forward, 1, AsSeenBy=camera)
# different interpolation function
obj.move(forward, 1, style=abruptly)
```

(a) Object Movements

```
ArmsOut = DoTogether(
    Bunny.Body.LeftArm.Turn(Left, 1/8),
    Bunny.Body.RightArm.Turn(Right, 1/8))
ArmsIn = DoTogether(
    Bunny.Body.LeftArm.Turn(Right, 1/8),
    Bunny.Body.RightArm.Turn(Left, 1/8))
BangTheDrumSIowly = DoInOrder(
    ArmsOut,
    Armsln,
    Bunny.PlaySound('bang'))
BangTheDrumSIowly.Loop()
```

(b) Bunny Drum Script

Fig. 31. Alice script example (adapted from [58])

Language 57 *Playspecs*

generic / tool / practice

Osborn et al. introduce PlaySpecs, regular expressions for specifying and analyzing desirable properties of game play traces, sequences of player actions. PlaySpecs are validated with the PuzzleScript engine, which is itself described as Language 91, and Prom Week, a social simulation puzzle game. The TypeScript sources of PlaySpecs are released under the MIT license[1].

[1]https://github.com/JoeOsborn/playspecs-js

publication	query	publication type	research category	note
[204]	930w	conference paper	validation research	
[200]	–w	PhD thesis	validation research	

Language 58 *MAD and SAnR*

generic / engine / practice

Van Rozen and Heijn study Ludoscope (Language 36) and address quality issues of grammar-based level generation. They propose two techniques for improving grammars to generate better game levels. The first, Metric of Added Detail (MAD), leverages the intuition that grammar rules gradually add detail, and uses a detail hierarchy that indicates for calculating the score of rule applications. The second, Specification Analysis Reporting (SAnR) proposes a language for specifying level properties, and analyzes level generation histories, showing how properties evolve over time. LudoScope Lite (LL), a prototype that implements the techniques and demonstrates their feasibility[1].

[1]https://github.com/visknut/LudoscopeLite (visited April 25th 2019)

publication	query	publication type	research category	note
[269]	–n	workshop paper	proposal of solution	LL


```
See - purple - move - away
See - apple - move - towards
```

(a) Visual Kodu Rule (b) Textual Kodu Rules

Fig. 32. Visual and Textual Kodu Examples (adapted from MacLaurin [145])

B.9 Education

Language 59 ***Alice*** *generic / tool / educative*

Alice is intended for authoring interactive 3D animations, and teaching programming constructs to undergraduates with no prior programming knowledge. Figure 31 shows a textual Alice example that uses Logo-style coordination and movement. Whereas the earlier versions were textual and based on Python, later version of Alice support mediated transfer from block based notation to script. Alice is available online for free[1].

[1]http://www.alice.org (visited March 27th 2019)

publication	query	publication type	research category	note
[57]	~~gd~~	PhD thesis	proposal of solution	
[58]	g	conference paper	experience report	
[273]	108w	conference paper	proposal of solution	Cheshire

Language 60 ***Gamestar Mechanic*** *genre-specific / tool / educative*

Salen presents the overview of the pedagogy and development process of Gamestar Mechanic, an RPG style online game for teaching the fundamentals of game design to children aged 7 to 14. Games presents the results of a study into teaching middle school children to think like game designers by repairing broken games and developing games from scratch given specifications [102]. The game is available commercially[1].

[1]http://gamestarmechanic.com/ (visited April 10th 2019)

publication	query	publication type	research category	note
[223]	2w	journal article	proposal of solution	
[102]	19w	journal article	validation research	

Language 61 *Kodu*

application-specific / tool / educative

Kodu is a graphical programming language for helping young children learn through independent playful exploration. Kodu expresses rules that control robots in a realtime 3D gaming environment. Figure 32a shows a visual rule, which states that when a monster sees a red apple, it moves towards it. Figure 32b shows two textual rules. Kodu GameLab and educational materials are available from Microsoft Research[1].

[1]https://www.kodugamelab.com (visited March 27th 2019)

publication	query	publication type	research category	note
[246]	~~gd~~	manual	practice	
[144]	466w	invited talk	proposal of solution	
[145]	466w	journal article	proposal of solution	reprint

Language 62 *Squeak – Croquet*

generic / engine / practice

Squeak is a dialect of Smalltalk, an object-oriented, class-based, and reflective programming language[1]. Its *Morphic framework* facilitates visual and interactive programming and debugging of applications for domains such as education, gaming and research. Masuch and Rueger report experiences on using Squeak for teaching game design [160]. They investigate requirements for a collaborative learning environment that uses OpenCroquet, an audio-visual 3D environment that has built-in features supporting collaboration. Because the OpenCroquet project web site no longer exists, we share a blog with several related frameworks[2].

[1]http://squeak.org (visited January 9th 2019)
[2]http://planetcroquet.squeak.org (visited March 27th 2019)

publication	query	publication type	research category	note
[160]	142w	conference paper	experience report	

Language 63 *Scratch*

generic / engine / educative

Scratch is an visual environment for creating, designing and remixing interactive stories, games, animations, and simulations, which is intended for children between 6 and 12 years old. Scratch is a block based language implemented on Squeak (Language 62) whose its syntactic constructs fit together as puzzle pieces, for learning creative thinking and understanding logic and programming concepts [216]. A web-based editor of the current version and a large collection of projects contributed by its users are available online[1].

[1]https://scratch.mit.edu/ (visited March 27th 2019)

publication	query	publication type	research category	note
[216]	~~gd~~	journal article	experience report	
[52]	56w	short paper	evaluation research	

Language 64 *Starlogo TNG* *generic / tool / educative*

StarLogo The Next Generation (TNG) is a language, tool and 3D simulation environment for novices for creating and understanding complex systems such as games. The language is a block-based extension of Logo, a dialect of Lisp and a successor of StarLogo. Its elements are represented as colored blocks that fit together like puzzle pieces that do not permit syntax mistakes. As such, it lends itself teaching introductory game development [28, 276]. A distribution is available for Windows and MAC OS[1]. An open source version, OpenStarLogo, is available under the MIT license[2]. A successor called StarLogo Nova can be used online[3].

[1] http://web.mit.edu/mitstep/projects/starlogo-tng.html (visited January 8th 2019)
[2] http://web.mit.edu/mitstep/openstarlogo/index.html (visited January 8th 2019)
[3] https://www.slnova.org (visited January 8th 2019)

publication	query	publication type	research category	note
[28]	170w	journal article	proposal of solution	
[276]	12w	extended abstract	experience report	

Language 65 *AgentSheets and AgentCubes* *generic / tool / educative*

Repenning proposes Agentsheets, a tool for building domain-specific visual environments [215]. Later, AgentSheets becomes a tool for creating agent-based games and simulation, also used for teaching game design. In *conversational programming*, when a programmer edits a game or simulation, an agent executes the program and provides syntactic and semantic feedback [214]. Ioannidou et al. propose AgentCubes, a 3D game-authoring environment for teaching middle school children modeling, animation and programming. Both are commercial products[1].

[1] http://www.agentsheets.com (visited April 17th 2019)

publication	query	publication type	research category	note
[215]	~~game~~	journal article	proposal of solution	AgentSheets
[125]	151w	conference paper	proposal of solution	AgentCubes
[214]	488w	conference paper	demo paper	AgentSheets
[6]	75w	conference paper	proposal of solution	AgentWeb
[8]	34n	conference paper	evaluation research	AgentSheets
[7]	158n	PhD Thesis	evaluation research	AgentWeb

B.10 Gamification

Language 66 *Gamification Language* *generic / tool / practice*

Herzig et al. aim to enrich information systems with game design elements that increase the engagement and motivation of its users [116]. The Gamification Language (GAML) is a textual and declarative DSL that helps domain-experts define these elements, and helps IT experts more easily incorporate them.

Matallaoui et al. propose a model-driven architecture for designing and generating building blocks of serious games, and apply it in the creation of an achievement system [161]. An Xtext grammar is available under the MIT license[1].

[1]https://github.com/AmirIKM/GamiAsService/ (visited April 10th 2019)

publication	query	publication type	research category	note
[116]	5n	conference paper	proposal of solution	
[161]	7n	conference paper	validation research	

Language 67 *GLiSMo* *genre-specific / tool / educative*

Thillainathan et al. aim to enable educators without prior programming skills to create didactically sound serious games [256]. They propose Serious Game Logic and Structure Modeling Language (GLiSMo), a visual modeling language that applies model-driven development techniques for generating games from visual models.

publication	query	publication type	research category	note
[256]	14n	conference paper	proposal of solution	

Language 68 *UAREI* *generic / tool / practice*

Ašeriškis et al. present User-Action-Rule-Entities-Interface (UAREI), a visual language for representing game mechanics for software gamification. They use UAREI to simulate and evaluate the effects of gamified systems on different types of players. UAREI models are directed graphs consisting of five node types (one for each word in the acronym) with intentionally limited expressiveness. An operational semantics is not defined. Case studies include OilTrader and the Trogon Project Management System.

publication	query	publication type	research category	note
[19]	120n	journal article	solution proposal	
[18]	127n	workshop paper	solution proposal	
[17]	241n	PhD thesis	validation research	

B.11 General Game Playing

Language 69 *Multigame* *genre-specific / tool / research*

Romein et al. present Multigame, a procedural DSL for expressing the rules of board games intended to research automatic parallelism and parallel game tree search in particular [217]. Multigame helps programmers focus on choosing parameters that influence behavior instead of resolving issues in communication, synchronization, work- and data distribution and deadlocks. The manual describes Chess and Checkers [219]. Figure 33 shows two simpler examples.

```
knight_move =
  find own knight,
  pickup, orthogonal, step,
  either rotate 45 or rotate -45, step,
  not points at own piece, putdown.
```

(a) A Knight's move in Chess

```
dimensions (3,3)
symmetry all directions
pieces { mark 'X' 'O' }
main =
  irreversible, # each move is a conversion
  try new_mark else draw.
new_mark =
  find empty field,
  replace by mark,
  try [ test three_in_a_row, win ].
three_in_a_row =
  find own piece, # start from any position
  any direction,
  repeat 2 times [ step, points at own piece ].
```

(b) Tic Tac Toe

Fig. 33. Multigame Examples (adapted from Romein et al. [217] (a) and [219] (b))

publication	query	publication type	research category	note
[217]	~~gd~~	conference paper	validation research	
[219]	~~gd~~	manual	validation research	
[218]	~~gd~~	PhD thesis	validation research	

Language **70** ***Game Description Language*** *generic / engine / research*

General game playing studies how generic AI algorithms and techniques can help computer systems play more than one game successfully. The Game Description Language (GDL) provides a formal description of a game's rules that systems can use as a testbed for intelligent agents and algorithms. Thielscher translates GDL into action language semantics [255] and introduces GDL-II, an extension of GDL for incomplete information games [253]. Figure 34 shows a simple example that explicitly defines turn-taking, next states and sequence. Stanford hosts a web site, which refers to the annual general game playing competition and also includes additional examples[1].

[1] http://ggp.stanford.edu (visited March 26th 2019)

publication	query	publication type	research category	note
[143]	~~gd~~	technical report	report	GDL
[253]	~~gd~~	conference paper	validation research	GDL-II
[254]	~~gd~~	conference paper	validation research	GDL
[255]	507w	book chapter	validation research	GDL
[222]	~~gd~~	journal article	validation research	GDL
[221]	217n	Master's thesis	validation research	GDL-II

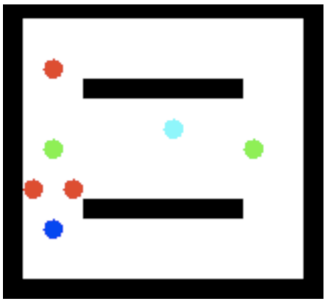
```
role(candidate).
role(random).
init(closed(1)).
init(closed(2)).
init(closed(3)).
init(step(1)).

legal(random,hide_car(?d))
  <= true(step(1)), true(closed(?d)).
legal(random,open_door(?d))
  <= true(step(2)),
     true(closed(?d)),
     not true(car(?d)),
     not true(chosen(?d)).
legal(random,noop) <= true(step(3)).

legal(candidate,choose(?d))
  <= true(step(1)), true(closed(?d)).
legal(candidate,noop)   <= true(step(2)).
legal(candidate,noop)   <= true(step(3)).
legal(candidate,switch) <= true(step(2)).
sees(candidate,?d)
  <= does(random,open_door(?d)).

next(car(?d))
  <= does(random,hide_car(?d)).
next(car(?d)) <= true(car(?d)).
next(closed(?d))
  <= true(closed(?d)),
     not does(random,open_door(?d)).
next(chosen(?d))
  <= does(candidate,choose(?d)).
next(chosen(?d))
  <= true(chosen(?d)),
     not does(candidate,switch).
next(chosen(?d))
  <= does(candidate,switch),
     true(closed(?d)),
     not true(chosen(?d)).
next(step(2)) <= true(step(1)).
next(step(3)) <= true(step(2)).
next(step(4)) <= true(step(3)).
terminal      <= true(step(4)).
goal(candidate, 100)
  <= true(chosen(?d)), true(car(?d)).
goal(candidate, 0)
  <= true(chosen(?d)), not true(car(?d)).
```

There are two players, a candidate and a host. Initially, three doors are closed. The host may first hide a car behind a door, and open a closed door at step 2 if it does not conceal a car and is not chosen. The candidate may first choose a closed door, optionally switch doors at step 2, and sees it when the host opens a door. When a car is hidden behind a door it remains there. Doors remain closed when not opened. When the candidate choses a door, it remains chosen unless they switch. Steps are sequential and the candidate wins only when choosing correctly.

Fig. 34. GDL description of the Monty Hall game (adapted from Thielscher [255])

```
t_max= 28
score_max= 6
0 Red things, random short
4 Green things, clockwise
9 Blue things, still

//When Red and Green things collide, Red survives and
     Green dies, and the score is -1-1 = -2.
collision: Red, Green → none, death, -1, -1
collision: Red, Blue → death, death, 1, 1
collision: Red, Agent → death, death, -1, 0
collision: Green, Blue → none, death, -1, -1
collision: Green, Agent → teleport, none, -1, 1
collision: Blue, Agent → death, none, 1, 1
```

(a) Race against green: score 6 within 28 time steps



(b) A game's start state: things and the agent (cyan) are randomly placed on the fixed grid

Fig. 35. Rule set of a Pac-man-like game (adapted from Togelius and Schmidhuber [257])

Language **71** ***Rules of Pac-man-like Games*** *application-specific / tool / research*

Togelius and Schmidhuber propose automatic game design as a means to generalize AI techniques. They demonstrate playable rule sets can be generated and evolved for the restricted

domain of Pac-man-like games [257]. Games consist of a fixed grid of cells populated by an agent (cyan) and things (red, blue and green) with random start positions. Variable movement logics allow for the agent and the things to move and collide, i.e. end up on the same cell. The rule space consists of parameters for limiting the amount of time steps $t_{max} \in \{0..100\}$ and scoring $score_{max} \in \{1..50\}$, the movement logic, and effects of collisions on things (none, death or teleportation to random cell) and scoring (limited to $-1, 0, +1$). Figure 35 shows an evolved rule set of a game where the objective is to compete with green things to catch blue things.

publication	query	publication type	research category	note
[257]	83w	conference paper	proposal of solution	

Language 72 *Ludi*

genre-specific / engine / practice

Browne and Maire examine how to synthesize and evaluate high quality combinatorial games using evolutionary game design, an approach that combines evolutionary search with quality measurements in self-play simulations. They describe Ludi, a game system that synthesizes board games and evaluates their qualities [39]. Ludi's Game Description Language includes game facets (called *ludemes*) for player (name), board (shapes and size), pieces with definitions of how they can move and end conditions. Figure 36 shows two examples. Yavalath is a novel commercially published game generated by Ludi where making four-in-a-row is winning, but making three-in-a-row before is losing. Browne proposes generating context-free grammars by analyzing the class hierarchies of game systems, in particular Ludii, to obtain so-called *class grammars* for varying constructor parameters and evolving games [42]. Ludii is being developed in the context of the Digital Ludeme Project[1], which studies how historical games developed by means of modern AI techniques.

[1] http://ludeme.eu (visited March 25th 2019)

publication	query	publication type	research category	note
[40]	–w	PhD thesis	evaluation research	Ludi
[39]	99w	journal article	evaluation research	Ludi
[41]	803w	book	evaluation research	Ludi
[42]	80n	conference paper	proposal of solution	Ludii

Language 73 *Strategy Game Description Language*

genre-specific / tool / practice

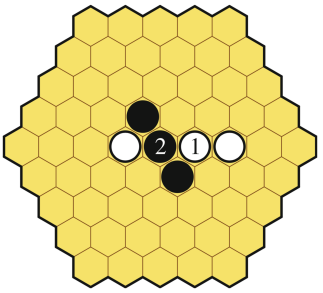
Mahlmann et al. propose the Strategy Game Description Language (SGDL). Combined with evolutionary algorithms and appropriate fitness functions, SGDL serves as a means to describe and generate complete new strategy games, and variations of old ones [152, 153]. SGDL visually models behaviors as trees of conditions and consequences. These are expressions and statements whose nodes are actions (triangles) comparators and functions (ovals), operators (diamonds) and constants (circles). Figure 37 shows a simple example of a 'Go North' action. In its left hand condition, the *_Map* function takes attributes *x* and *y - 1* as input, and its right hand consequence *y = y - 1* happens if the output equals *null*. Other examples include complex variations of Rock Paper Scissors and Dune II [151].

```
(game Tic-Tac-Toe
  (players White Black)
  (board
    (tiling square i-nbors)
    (size 3 3)
  )
  (end
    (All win (in-a-row 3))
  )
)
```

(a) Tic Tac Toe

```
(game Yavalath
  (players White Black)
  (board (tiling hex)
         (shape hex) (size 5))
  (end
    (All win (in-a-row 4))
    (All lose
      (and (in-a-row 3)
           (not (in-a-row 4))))
  )
)
```

(b) Yavalath



(c) Yavalath: white forces a win (adapted from Browne [41])

Fig. 36. Ludi GDL – Source code examples adapted from Browne and Maire [39]

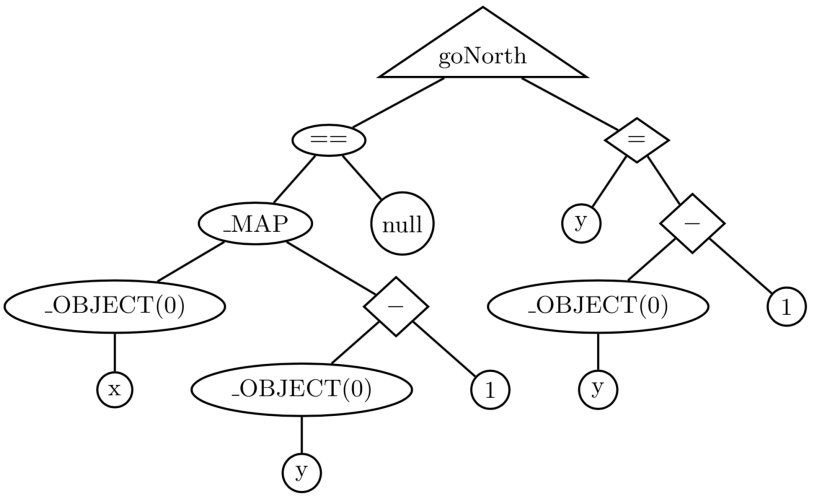


Fig. 37. Strategy Game Description Language – ”Go North” action (adapted from Mahlmann et al. [153])

publication	query	publication type	research category	note
[153]	646w	conference paper	validation research	
[152]	–w	conference paper	validation research	
[151]	289n	PhD thesis	validation research	

Language **74** ***Card Game Description Language*** *genre-specific / tool / practice*

Font et al. present initial findings on generating and analyzing both novel and existing card games [87]. They present a Card Game Description Language for expressing a wide variety of card games by formalizing the rules. They evolve playable card games using grammar-guided genetic programming. They assess playability and balance by measuring the performance of

```
THE ANT AND THE GRASSHOPPER
Stages and rules

Stage 0
COMPUTER COMMAND <Unconditional> GIVE Player:
      0 Amount: 89 tokens
COMPUTER COMMAND <Unconditional> DEAL Table:
     0 Amount: 1 cards
COMPUTER COMMAND <Unconditional> GIVE Player:
      2 Amount: 39 tokens
COMPUTER COMMAND <Unconditional> GIVE Player:
      <all> Amount: 87 tokens

Stage 1
if SHOW >= T0 then PLAY IT
COMPUTER COMMAND <Unconditional> DEAL Player:
      <all> Amount: 6 cards
COMPUTER COMMAND <Unconditional> GIVE Player:
      <all> Amount: 58 tokens
PLAY ONLY ONCE if SHOW SAME RANK T1 then PLAY
      IT

Stage 2
if SHOW < T1 then PLAY IT
PLAY ONLY ONCE if SHOW >= T0 then PLAY IT
PLAY ONLY ONCE if SHOW > T0 then PLAY IT

Stage 3
COMPUTER COMMAND <Unconditional> GIVE Player:
      0 Amount: 77 tokens

Stage 4
COMPUTER COMMAND <Unconditional> GIVE Player:
      0 Amount: 44 tokens
MANDATORY if PLAY 994, > T0 then BET

Stage 5
PLAY ONLY ONCE if DRAW then BET
if SHOW >= T0 then PLAY IT
COMPUTER COMMAND <Unconditional> GIVE Player:
      <all> Amount: 63 tokens
PLAY ONLY ONCE if DRAW then BET

Ranking
Card(s)            Value
Four of a kind     190
6 + 8 + Jack       212

Winning conditions
5 points for each token.
3 points for finishing the game.
```

Fig. 38. Card Game Description Language – "The Ant and the Grasshopper" (adapted from Font et al. [87])

several agents. In addition, they filter games with too many stages and rules [88]. Figure 38 shows an example. Other examples include poker variant Texas hold âĂŹem, Blackjack and UNO.

publication	query	publication type	research category	note
[87]	47w	conference paper	proposal of solution	
[88]	590w	conference paper	validation research	

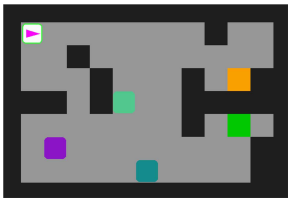
Language 75 ***Video Game Description Language*** *generic / engine / research*

Ebner et al. propose a Video Game Description Language (VGDL) as a means for general video game playing that expresses a wide range of classic 2D game types in a high-level, concise and human readable manner [81], e.g., approximations of Pong, Boulder-Dash, Tank Wars, Super Mario, Lunar Lander and Pac-Man.
Schaul proposes PyVGDL, a Python implementation of VGDL and a game simulation environment for conducting research [226, 227]. Figure 39 shows an example description (a). This description maps each game object to an ASCII character (LevelMapping) used in level descriptions (b). Next, it specifies their behaviors (SpriteSet) by using predefined functions (c). Finally, it defines the effects of possible collisions (InteractionSet) and win conditions (TerminationSet).

```
BasicGame
  LevelMapping
    G > goal
    + > key
    A > nokey
    1 > monster
  SpriteSet
    goal   > Immovable color=GREEN
    key    > Immovable color=ORANGE
    sword > Flicker limit=5 singleton=True
    movable      >
      avatar     > ShootAvatar stype=sword
        nokey    >
        withkey > color=ORANGE
      monster    > RandomNPC cooldown=4
  InteractionSet
    movable wall    > stepBack
    nokey goal      > stepBack
    goal withkey    > killSprite
    monster sowd    > killSprite scoreChange=1
    avatar monster > killSprite
    key avatar      > killSprite scoreChange=5
    nokey key       > transformTo stype=withkey
  TerminationSet
    SpriteCounter stype=goal    win=True
    SpriteCounter stype=avatar win=False
```

(a) VGDL description for a Legend of Zelda-like game

```
wwwwwwwwwwwww
wA         w   w
w   w          w
w     w    w +ww
www  w1   wwwww
w          w G w
w  1           ww
w         1    ww
wwwwwwwwwwwww
```



(b) Side-by-side level description (left) and rendering (right) of a Legend of Zelda-like game level, where the hero Link (A) confined by dungeon walls (w) must find a key (+) and the exit goal (G) while killing or avoiding monsters (1).

```
def killIfFromAbove(s, p, game):
  """Kills the sprite, if the other one is
       higher and moving down."""
  if (s.lastrect.top > p.lastrect.top and
      p.rect.top > p.lastrect.top):
    killSprite(s, p, game)
```

(c) Extension for Super Mario that restricts the *killSprite* procedure to downward movement

Fig. 39. Video Game Description Language (adapted from in Schaul [227])

PyVGDL is used in The General Video Game AI Competition[1] for benchmarking algorithms for planning, level generation and learning. PyVGDL is available under the 3-clause BSD license[2].

[1]http://www.gvgai.net (visited April 5th 2019) – also maintains a list of related publications
[2]https://github.com/schaul/py-vgdl (visited April 5th 2019)

publication	query	publication type	research category	note
[81]	31w	book chapter	proposal of solution	VGDL
[226]	–w	conference paper	proposal of solution	PyVGDL
[26]	56n	extended abstract	proposal of solution	VGDL
[227]	236w	journal article	proposal of solution	PyVGDL

Language **76** ***Recycle*** *genre-specific / tool / practice*

Bell and Goadrich describe Recycle, a card game description language and its implementation Cardstock, which can automatically playtest card games with algorithms that represent intelligent players. As a demonstration, they playtest variants of the games Agram, Pairs and War.

publication	query	publication type	research category	note
[29]	–w	journal article	proposal of solution	

B.12 Script and Programming

Language 77 *Python* *generic / engine / practice*

Python is an interpreted general-purpose programming language originally developed by van Rossum. Its language features include modules, exceptions, dynamic typing, data types and classes. Examples of languages built on top of Python include versions of Alice (Language 59) and PyVGDL (Language 75). The Python Package Index[1] hosts many reusable modules for various purposes, including game development. The current version (v3) of its portable and embeddable C implementation is released under the Python Software Foundation License[2].

[1]https://pypi.org (visited July 14th 2019)
[2]https://www.python.org (visited July 14th 2019)

publication	query	publication type	research category	note
[65]	–w	article	experience report	
[270]	10w	book	practice	
[128]	227w	short paper / tutorial	practice	
[209]	377n	book	practice	

Language 78 *Lua* *generic / engine / practice*

Lua is an interpreted general-purpose programming language developed by Ierusalimschy et al. [123, 124]. Originally intended for the petrochemical industry, Lua is now also used for scripting in Games. Its APIs enable embedding in C and its functional and dynamic features support constructing embedded DSLs.
Wasty et al. describe ContextLua, a context-oriented programming extension to Lua that is suitable for implementing dynamic behavioral variations in computer games [277]. Layers modify the behaviour of function calls as shown in Figure 40. The `proceed` method calls the next appropriate method in the current layer composition. The `with` and `without` statements are used to activate and deactivate layers respectively.
The sources of Lua[1] and ContextLua[2] are available online under the MIT license.

[1]https://www.lua.org (visited March 21st 2019)
[2]https://www.hpi.uni-potsdam.de/hirschfeld/trac/Cop/wiki/ContextLua (visited May 1st 2019)

publication	query	publication type	research category	note
[123]	551w	journal paper	proposal of solution	Lua
[124]	834w	conference paper	experience report	Lua
[277]	~~gd~~	workshop paper	proposal of solution	ContextLua
[133]	54n	workshop paper	proposal of solution	Lua

Language 79 *Vision on Game Programming* *generic / framework / practice*

In an invited talk on “The Next Mainstream Game Programming Language”, Sweeney (Epic Games) shares a perspective on language constructs for game development with focus on performance, modularity, reliability and concurrency [248]. He argues for productivity, modular

```
function Monster:getSensingDistance()
  return 10
end

function Monster:Night_getSensingDistance()
  return proceed() - 5
end
```

(a) Monster behaviour variation

```
with(Night, function()
  print(monster:getSensingDistance())
end)

with({Night, Sneak}, function()
  print(monster:getSensingDistance())
end)
```

(b) Night Layer Activation

Fig. 40. ContextLua code snippets (adapted from Wasty et al. [277])

libraries, debugging facilities, and reflects briefly on perceived strengths (unions, maybe) and weaknesses of Haskell.

publication	query	publication type	research category	note
[248]	3w	abstract and slide deck	opinion paper	

Language 80 *DisCo*

generic / tool / practice

Nummenmaa et al. propose simulating gameplay on a logical event level in the early states of the game development process [196]. As a design tool, simulating long-term dynamics of abstract and simplified game prototypes can reveal problems early on. They use DisCo[1], a software package for creating and executing formal specifications, which has been extended to for the analysis and simulation of games. The DisCo language has an action-oriented execution model based on temporal logic. A simulation model of a game called Tower Bloxx demonstrates the approach.

[1] http://disco.cs.tut.fi (visited August 15th 2019)

publication	query	publication type	research category	note
[194]	298w	Master's thesis	proposal of solution	
[196]	26w	conference paper	vision paper	simulate prototypes
[195]	662w	workshop paper	validation research	analyze changes

Language 81 *Design by Contract*

generic / tool / practice

Paige et al. present qualitative and empirical results showing that light-weight formal methods are effective for developing a networked, multiplayer game. Their results, obtained in a pilot study on applying the *Design-by-Contract* approach, show that contracts (pre- and post-conditions) indeed help in diagnosing defects.

publication	query	publication type	research category	note
[207]	186w	journal article	evaluation research	

```
move is choose
pieces are Rock and Paper and Scissors
board starts [[Rock, Paper, Scissors]]
turns synchronize
Beat means player(Rock) && opponent(Scissors)
  or player(Scissors) && opponent(Paper)
  or player(Paper) && opponent(Rock)
goal is Beat # success!
score increments
3x1 grid
```

(a) Rock Paper Scissors

```
turn is player place piece
3x3 grid
pieces are X and O
turns alternate
players are X and O
goal is &Three_in_a_row
Three_in_a_row means
     (x-1,y)   && (x,y) && (x+1,y)
  or (x,y-1)   && (x,y) && (x,y+1)
  or (x-1,y-1) && (x,y) && (x+1,y+1)
  or (x-1,y+1) && (x,y) && (x+1,y-1)
board starts empty
```

(b) Tic Tac Toe

Fig. 41. EGGG example specifications (adapted from Orwant [198])

Language 82 *GameMaker* *generic / engine / practice*

GameMaker is a commercial graphical game creation tool with a drag and drop interface by YOYO Games[1], which is described by Overmars [205, 206]. The Game Maker Language (GML) is a C-like language for scripting.

[1] https://www.yoyogames.com/gamemaker (visited November 19th 2018)

publication	query	publication type	research category	note
[205]	14w	journal article	experience report	
[206]	358w	journal article	experience report	

Language 83 *Extensible Graphical Game Generator* *genre-specific / engine / practice*

Orwant describes the Extensible Graphical Game Generator (EGGG), a system for game programming aimed at productivity and reuse. EGGG offers a textual formalism, and leverages an ontology that codifies similarities between traditional games such as board- and card games. Examples include, Rock Paper Scissors, Tic Tac Toe, Poker, Crossword, Deducto, Tetris and Chess [198]. Figure 41 shows the two simplest examples.

publication	query	publication type	research category	note
[199]	–w	journal article	proposal of solution	
[198]	–w	PhD thesis	proposal of solution	

Language 84 *Mogemoge* *genre-specific / tool / practice*

Nishimori and Kuno address the lack support in game script languages for interactions among multiple concurrent activities in a state-dependent manner. They propose a novel event handling framework called *join token* as a supplementary mechanism to conventional object orientation, in which the states of game characters can be expressed as tokens and interactions as handlers. The language Mogemoge implements join tokens, and is used for creating two simple 2D games,

Balloon (defending against bombs) and Descender (climbing down a wall). Its Java sources are available online[1]. Copyright is retained by Nishimori.

[1]http://www.nisnis.jp/mogemoge/ (visited January 10th 2019)

publication	query	publication type	research category	note
[191]	76w	conference paper	proposal of solution	
[193]	–w	conference paper	proposal of solution	
[192]	–w	journal article	proposal of solution	

Language 85 *Scalable Game Language* *generic / engine / practice*

White et al. propose the Scalable Game Language (SGL), a declarative language that extends SQL for improving the quality of games, notably scalability. They describe two patterns: the *state-effect-pattern*, which is similar to the well-known game loop, and the *restricted iteration pattern*, which prevents out-of-bounds exceptions.

publication	query	publication type	research category	note
[281]	–w	conference paper	proposal of solution	
[283]	–w	conference paper	proposal of solution	
[282]	544w	journal article	philosophical paper	
[284]	457w	journal article	philosophical paper	reprint

Language 86 *Network Scripting Language* *generic / tool / practice*

Russell et al. present a novel DSL called Network Scripting Language (NSL) for programming bandwidth-efficient online games. Developers can use NSL to create the game logic of deterministic, concurrent and distributed games. The system automatically maintains consistency between the clients and the sever that run the scripts. NSL has a Java-like syntax. In NSL, objects are lightweight processes that execute a game loop. Scripts contain specialized statements for sending and receiving messages and handling synchronization. PointWorld is a simulation that demonstrates the approach.

publication	query	publication type	research category	note
[220]	33n	workshop paper	proposal of solution	

Language 87 *Haskell – 4Blocks DSL* *application-specific / engine / practice*

Calleja and Pace propose scripting game-specific AI with embedded DSLs in Haskell. They demonstrate their approach with the 4Blocks DSL for Tetris.

publication	query	publication type	research category	note
[46]	~~gd~~	workshop paper	proposal of solution	
[47]	52n	workshop paper	proposal of solution	

Language 88 *Casanova* *generic / engine / practice*

Maggiore et al. describe Casanova [146, 147, 148, 149]., a language-extension to F# for engineering games aimed at consistency and performance. Rules inside entity type declarations determine how entities change during a tick of the game loop. Additionally, imperative processes are supported through coroutines integrated with the rules. Game scripts consist of the main script and pairs of event detection- and event response scripts. Abbadi et al. [2, 3, 4] and di Giacomo et al. [68, 69, 70, 71] continue work on Casanova, in the context of optimized compilation, meta-programming and high performance encapsulation. Casanova 2 is available for Unity or stand-alone under the MIT license on GitHub[1]. The distribution includes an asteroid game and several tutorials.

[1]https://github.com/vs-team/casanova-mk2 (visited November 19th 2018)

publication	query	publication type	research category	note
[146]	47n	workshop paper	proposal of solution	Casanova
[147]	34w	conference paper	validation research	Casanova
[148]	95w	conference paper	proposal of solution	Casanova
[149]	308w	conference paper	validation research	Casanova
[68]	177n	Master's thesis	validation research	Casanova
[4]	60n	conference paper	proposal of solution	Casanova II
[3]	26n	conference paper	proposal of solution	Casanova II
[69]	53n	conference paper	proposal of solution	Metacasanova
[2]	195n	PhD thesis	validation research	Casanova II
[71]	101n	conference paper	validation research	Metacasanova
[70]	86n	journal article	validation research	Casanova II

Language 89 *Haskell – Sound Specification DSL* *generic / tool / practice*

Bäärnhielm et al. describe a sound specification DSL intended for designing immersive and interactive experiences for a Nordic technology-supported Live Action Role Playing (LARP) game. They demonstrate features of the Haskell-based DSL, by expressing *sound scenes* of a Nordic LARP called The Monitor Celestra. In this game, which takes place on a space ship, participants receive roles such as crew, passengers and refugees. Within a framework of plots, storylines and clues, supported by sound, they act out a story where choices determine the outcome.

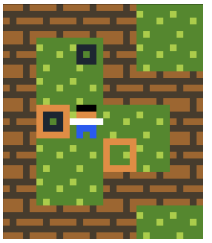
publication	query	publication type	research category	note
[22]	196n	journal article	experience report	

Language 90 *MUDDLE* *application-specific / tool / practice*

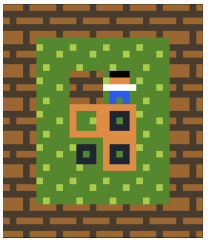
Bartle gives a historical account of the creation of MUDDLE, a language for the first Mutli-User Dungeon (MUD) game, which gave the genre its name, also known as Massive Multiuser Online (MMO) games.

```
title Simple Block Pushing Game
author Stephen Lavelle
homepage www.puzzlescript.net
========
OBJECTS
========
Background
lightgreen green
11111
01111
11101
11111
10111

Target
darkblue
.....
.000.
.0.0.
.000.
.....

Wall
brown darkbrown
00010
11111
01000
11111
00010

Player
black orange
     white blue
.000.
.111.
22222
.333.
.3.3.

Crate
orange
00000
0...0
0...0
0...0
00000

=======
LEGEND
=======
. = Background
# = Wall
P = Player
* = Crate
@ = Crate and
     Target
O = Target

=======
SOUNDS
=======
Crate MOVE
     36772507

================
COLLISIONLAYERS
================
Background
Target
Player, Wall,
     Crate
======
RULES
======
[>  Player |
     Crate] -> [>
       Player |>
     Crate]

==============
WINCONDITIONS
==============
All Target on
     Crate

=======
LEVELS
=======
####..
#.O#..
#..###
#@P..#
#..*.#
#..###
####..

######
#....#
#.#P.#
#.*@.#
#.O@.#
#....#
######
```

(a) Source code with dynamic dynamic syntax highlighting of sprites



(b) First level



(c) Second level

Fig. 42. PuzzleScript tutorial: “Simple Block Pushing Game” (from puzzlescript.net)

publication	query	publication type	research category	note
[27]	303n	book chapter	historical account	

Language 91 *PuzzleScript* *genre-specific / engine / practice*

PuzzleScript is an online textual puzzle game design language and interpreter[1] created by Stephen Lavelle using JavaScript and HTML5/CSS. PuzzleScript game levels are tile maps populated by objects (named sprites of 5x5 pixels) that can move and collide, and whose game logic is defined as a set of rewrite rules. Figure 42 shows an example where the objective is to push crates into place. When the player collides with a crate, both directionally move if possible. The source are released under the MIT license[2].

Lim and Harell present an approach for automated evaluation and generation of PuzzleScript videogames and propose two heuristics [140]. The first, level state heuristics, determines how close the state of given level is to completion during gameplay. The second, ruleset heuristics, evaluates rules defining a videogame’s mechanics and assesses them for playability. Osborn et al. apply Playspecs (Language 57).

[1] https://www.puzzlescript.net (visited November 24th 2018)

[2] https://github.com/increpare/PuzzleScript (visited November 24th 2018)

publication	query	publication type	research category	note
[140]	110w	conference paper	validation research	

B.13 Model-Driven Engineering

Language 92 *UML -Metamodeling* *generic / engine / practice*

Montero Reyno and Carsí Cubel address the increasing complexity of game development by applying model-driven engineering and UML to the development of 2D platform games [176]. They aim to enhance productivity in terms of quality, time and cost [175]. A prototype tool uses Platform Independent Models (PIMs) for defining the structure and behaviour of the game, and Platform Specific Model (PSM) for mapping game actions to hardware control devices for player interaction [176] and UML metamodels for social context, structure diagram and rule set [175]. The tool generates C++ prototype games for a middleware called HAAF Game Engine. These are then iteratively play tested and manually completed and fine-tuned.

publication	query	publication type	research category	note
[176]	58w	conference paper	proposal of solution	
[175]	27w	conference paper	proposal of solution	
[177]	72w	journal article	proposal of solution	

Language 93 *UML – Class and State Diagrams* *generic / tool / practice*

Tang and Hanneghan investigate how to define a Domain-Specific Modeling Language for serious game design. They perform an analysis and propose a modeling framework that uses UML class diagrams and state diagrams for modeling user interactions and in-game components. They extend state diagrams with UI modeling elements [250].
In later work, they examine the state of the art in model-driven game development from a game-based learning perspective [251]. We compare this related work in Section 8.
Tang et al. propose a Game Technology Model for modeling serious games [252].

publication	query	publication type	research category	note
[250]	1w	conference paper	proposal of solution	
[251]	13n	journal article	survey	
[252]	–w	journal article	proposal of solution	

Language 94 *Statecharts* *generic / tool / practice*

Statecharts are visual diagrams for modeling behavior. Several several variants of the notation exist [61]. We identify two used in model-driven game development.
Kienzle et al. propose visual modeling game AI of NPCs in a RHAPSODY Statechart variant to ease the difficulty of programming consistent, modular and reusable game AI. They demonstrate the approach in an AI competition of EA Games called Tank Wars.

Brusk and Lager propose applying State Chart XML (SCXML) to the design and implementation of games, in particular games featuring natural language dialogue [43]. Brusk investigates how statecharts can be used for describing social interaction and dialogue behavior for believable characters in game worlds [44]. Various tools and libraries for Statecharts have since become available. The latest recommendation for v1.0 of SCXML as w3c standard dates from September 1st 2015[1].

[1]https://www.w3.org/TR/scxml/ (visited March 27th 2019)

publication	query	publication type	research category	note
[67]	~~gd l~~	conference paper	proposal of solution	Rhapsody Sc.
[131]	−w	conference paper	proposal of solution	Rhapsody Sc.
[43]	180w	conference paper	proposal of solution	SCXML
[44]	569w	conference paper	proposal of solution	SCXML

Language 95 ***Feature Models*** *generic / tool / practice*

Feature Models (FMs) are a visual notation for describing the variability of product features, e.g., in software product lines for the automotive or aerospace industries. Sarinho et al. propose an approach that entails using FMs for representing and manipulating the variability of game features, and an environment that integrates and adapts features of available game engines, e.g., for configuring game logic, rules and goals.

publication	query	publication type	research category	note
[224]	6n	conference paper	proposal of solution	
[225]	~~language~~	workshop paper	proposal of solution	

Language 96 ***SharpLudus*** *genre-specific / engine / practice*

Furtado et al. study how game development can be improved using visual domain-specific modeling languages, software product lines, software factories, generators and semantic validators aimed at software reuse and productivity [95, 97]. SharpLudus is a software factory intended to empower game designers in creating 2d adventure video games [97], but over the years targets also included RPG games, mobile touch-based games and 2d arcade games [95]. For instance, ArcadEx is a factory for 2d arcade games for the PC based on Microsoft XNA and the FlatRedBall engine [95]. DSLs are provided for describing games, mapping input of Xbox 360 buttons into XNA Keyboard keys, and modeling variability using feature models. Game descriptions are visual models of introduction screens and rooms with transitions between them (arrows), sound, entities, input handling, triggers, events and actions of NPCs. The project web site[1] contains videos and demos of Ultimate Berzerk, Stellar Quest and Tank Brigade, and links a to a distribution[2].

[1]http://cin.ufpe.br/~sharpludus/ (visited march 27th 2019)
[2]https://archive.codeplex.com/?p=sharpludus (visited March 27th 2019)

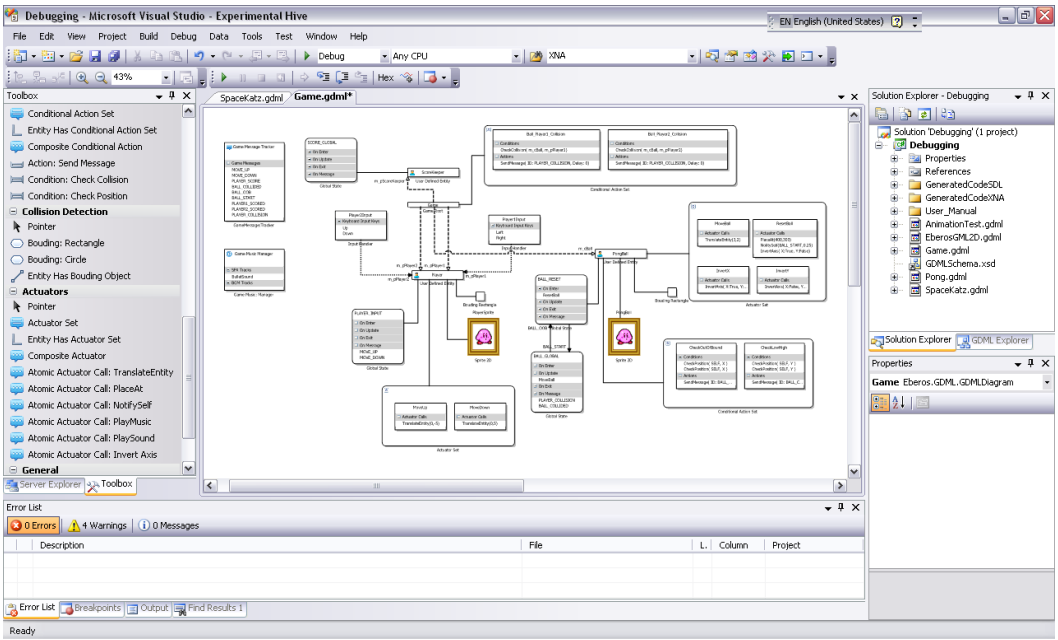


Fig. 43. Eberos GML2D showing a model of Pong (appears in Hernandez and Ortega [115])

publication	query	publication type	research category	note
[97]	1n	workshop paper	proposal of solution	SharpLudus
[99]	3n	Master's thesis	validation research	SharpLudus
[96]	87n	conference paper	tutorial	MS DSL tools
[101]	25n	journal article	validation research	SharpLudus
[98]	16n	workshop paper	experience report	SharpLudus
[95]	25w	journal article	proposal of solution	ArcadEx
[100]	93n	PhD thesis	validation research	all the above

Language 97 *Eberos GML2D* *genre-specific / tool / practice*

Hernandez and Ortega wish to learn how the game industry can profit from model-driven development approaches [115]. Eberos Game Modeling Language 2D (GML2D) is a graphical DSL that aims for expressiveness, simplicity, platform independence and library independence. Figure 43 shows the UI and a Pong model.

publication	query	publication type	research category	note
[115]	2n	workshop paper	proposal of solution	

Language 98 FlexibleRules

genre-specific / engine / practice

Frapolli et al. present FlexibleRules, a framework for implementing all aspects of digital board games aimed at customization, adaptability and end-user programming [89, 90, 91]. Its toolkit offers a *logic editor* for visually defining a directed graph of game entities (nodes), properties and relationships (edges). The *code editor* for a Lisp-like DSL enables programming games as the behavior specification of those entities. Aside from statements for control flow and messaging for communicating between entities, it includes rules that are defined as *laws* and *side effects*, similar to point-cuts and advice in aspect oriented programming. Flexible Rules is available under GPL v3[1]. Examples include Tic Tac Toe, Go and Snakes and Ladders.

[1]http://flexiblerules.fulviofrapolli.net (visited April 10th 2019)

publication	query	publication type	research category	note
[91]	21n	conference paper	validation research	
[89]	57n	journal article	proposal of solution	
[90]	78n	conference paper	validation research	

Language 99 PhyDSL

genre-specific / engine / practice

Guana and Stroulia propose PhyDSL, a textual DSL for rapidly prototyping mobile 2D physics-based games, and a model-driven environment that generates code for Android devices. PhyDSL has features for defining actors, environment and layout, activities and scoring rules. PhyDSL-2 is implemented in Xtext and available on GitHub[1].

[1]https://guana.github.io/phydsl (visited September 1st 2019)

publication	query	publication type	research category	note
[107]	24n	conference paper	proposal of solution	PhyDSL
[108]	29n	workshop paper	experience report	PhyDSL-2
[106]	209n	PhD thesis	validation research	PhyDSL-2

Language 100 Pong Designer

genre-specific / engine / practice

Mayer and Kuncak aim to empower end-users and to simplify modifying running programs [167]. They explore *game programming by demonstration* and present Pong Designer, an environment for developing 2D physics games through direct manipulation of object behaviors. Internally, a game's rules are expressed in an embedded DSL implemented in Scala. These rules are updated whenever a user performs a new demonstration. Sources are available on GitHub under the Apache 2.0 license[1]. Examples include Pong, Brick Breaker, Pacman and Tilting maze.

[1]https://github.com/epfl-lara/pongdesigner (visited July 12th 2019)

publication	query	publication type	research category	note
[167]	285n	conference paper	proposal of solution	
[166]	390n	PhD thesis	validation research	

Language 101 ***Board Game DSL*** *genre-specific / tool / practice*

Altunbay et al. describe a model-driven software development approach aimed at addressing increased complexity in video games, which is illustrated by a DSL for the board game domain based on UML meta-modeling [9].

publication	query	publication type	research category	note
[9]	383w	workshop paper	proposal of solution	

Language 102 ***RougeGame Language*** *genre-specific / tool / practice*

Féher and Lengyel illustrate the strength of model transformations based on graph rewriting-based in a case study on Rouge-like games, a genre of 2D dungeon crawlers. In particular, they study which cells are visible from a specific location on a 2D level map. They define the RougeGame language, a DSL defined as a meta-model expressing maps and visibility parameters. A transformation pipeline calculates the cell visibility based on rewrite rules.

publication	query	publication type	research category	note
[84]	189n	conference paper	proposal of solution	

Language 103 ***Reactive AI Language*** *genre-specific / engine / practice*

Zhu describes the Reactive AI Language (RAIL), a DSL for modeling behaviors in adventure games, and a model-driven game development approach that uses meta-modeling and EMF. The approach is validated in a case study called Orc's gold, a 2D action adventure game.

publication	query	publication type	research category	note
[291]	165n	PhD Thesis	validation research	

B.14 Metaprogramming

Language 104 ***Whimsy*** *application-specific / engine / educative*

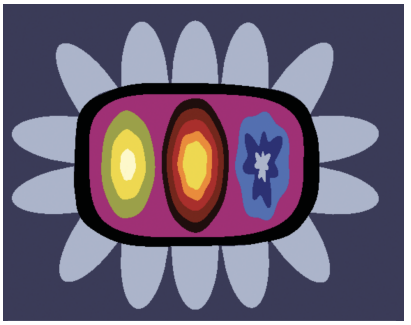
West discusses potential uses for DSLs in games and demonstrates Whimsy, a DSL for creating whimsical flowery shapes inspired by the works of Rodney Alan Greenblat [278]. Figure 44 shows how SuperEgg, Inner and Petal primitives can be used for generating an image similar to a painting. Whimsy is an external DSL implemented in C++ that requires the Windows SDK and DirectX 9. Its sources are available under the MIT license from GDCVault[1].

[1] https://twvideo01.ubm-us.net/o1/vault/GD_Mag_Archives/aug07.zip (visited May 9th 2019)

publication	query	publication type	research category	note
[278]	~~gd~~	magazine article	philosophical paper	practice

```
superegg 0.15,0.10,3.5 at .3,.7 size 1.2 black distort .01
petals 14 0.05 size 1.8 petalblue
inner .88,.01 tvpurple
superegg .1,.2,2 at .20,.7 size .4 distort .01 tvlime
inner .65,.01 tvyellow
inner .45,.01 tvlightyellow
superegg .1,.2,2 at .3,.7 size .5 distort .01 tvblack
inner .85 tvbrown
inner .80 tvred
inner .75 distort .03 tvorange
inner .70 tvyellow
superegg .1,.2,2 at .4,.7 size .4 distort .05 tvblue
inner .6 distort .2 tvdarkblue
inner .4 petalblue
```

(a) Source code



(b) Generated image

Fig. 44. Whimsy example replicating the style of a painting (adapted from West [278])

Language 105 ***Level Editors in DiaMeta*** *genre-specific / engine / educative*

Maier and Volk report teaching experiences on applying DiaMeta, an EMF-based language workbench for creating visual domain-specific languages, e.g., for level editors for classic games such as PacMan and the platform game Pingus [154]. Insights include that meta-modeling has a steep learning curve and that the proposed approach speeds-up game prototyping.

publication	query	publication type	research category	note
[154]	4w	conference paper	experience report	

Language 106 ***Text Adventures in Racket*** *genre-specific / tool / educative*

Flatt demonstrates in a tutorial-like manner how to create languages in Racket. He describes an illustrative text-adventure DSL for interactive fiction [85, 86]. The Racket metaprogramming language is distributed under the GNU LFPL[1].

[1]https://racket-lang.org (visited May 9th 2019)

publication	query	publication type	research category	note
[85]	181n	journal article	philosophical paper	
[86]	181n	journal article	philosophical paper	reprint

Language 107 ***Ficticious*** *genre-specific / tool / practice*

Palmer reports experiences on developing a set of micro-languages (DSLs) called Ficticious for describing narrative worlds in Interactive Fiction [208], including rich text markup, virtual world design and character interaction. The approach demonstrates how Ginger, a language with support for literate programming through so-called G-expressions, can be used to separate concerns

```
object Hook extends: FixedItem
  set name "hook"
  set aliases ("hook" "peg")
  set adjective ("small" "brass")
  set location 'CloakRoom
  action examine
    :story It's just a small brass hook,
    if (isIn cloak self)
      :story with a cloak hanging on it.
    else
      :story screwed to the wall.
```

(a) People, Places and Things

```
panel GamePageLakeShore extends: GamePage
  property Image panel1
  property Image panel2
  init
  setText 10 445 580 250
  setImage panel1 "imgs/charonGone.png"
  setImage panel2 "imgs/charonWaits.png"
draw
  if (eq? 'Boatman.location 'LakeShore)
    drawImage panel2 7 7
  else:
    drawImage panel1 7 7
```

(b) Page Layout

```
:story
  The sign reads "No␣Loitering" but
  ironically a cowboy whittles a small
  piece of wood *right beside* the sign.
```

(c) Rich Text and Grammar

```
conversation on OldMine
  oldMine "Ask␣about␣the␣old␣mine."
    :dialog
      Sam: I keep seeing an old donkey
      at the mine.
  donkey "Ask␣about␣the␣donkey." => oldMine
    :dialog
      Sam: The donkey comes and goes.
```

(d) Dialogue

Fig. 45. Ficticious microlanguages code snippets (adapted from Palmer [208])

in DSLs. Figure 45 shows code snippets for describing (a) people places and things; (b) page layout; (c) rich text and grammar; and (d) dialogue.

publication	query	publication type	research category	note
[208]	62n	conference paper	experience report	

Language 108 ***Dialog Script in Xtext*** *genre-specific / tool / educative*

In a textbook chapter on engineering DSLs for games, Walter proposes DSLs for bridging the gap between game design and implementation [274]. He describes a textual language called Dialog Script, as an introductory example for creating interactive branching narratives [274], which closely resembles an earlier version [275]. Dialog Script is implemented in Xtext and its prototype is available on GitHub[1] under version 2.0 of the Apache license.

[1] https://github.com/RobertWalter83/DialogScriptDSL (visited May 9th 2019)

publication	query	publication type	research category	note
[275]	4n	conference paper	proposal of solution	
[274]	94n	textbook chapter	proposal of solution	

REFERENCES

[1] E. Aarseth et al. "A Multi-Dimensional Typology of Games". In: *Proceedings of the 2003 DiGRA International Conference: Level Up, DiGRA 2003, Utrecht, The Netherlands, November 4–6, 2003*. Utrecht University, 2003.

[2] M. Abbadi. "Casanova 2: A Domain-Specific Language for General Game Development". PhD thesis. Tilburg University, Sept. 2017.

[3] M. Abbadi et al. "Casanova: A Simple, High-Performance Language for Game Development". In: *Serious Games – Proceedings of the 1st Joint International Conference on Serious Games, JCSG 2015, Huddersfield, UK, June 3–4, 2015*. Springer, 2015.

[4] M. Abbadi et al. "High Performance Encapsulation in Casanova 2". In: *Proceedings of the 7th Computer Science and Electronic Engineering Conference, CEEC, Colchester, UK, September 24–25, 2015*. IEEE, 2015.

[5] E. Adams and J. Dormans. *Game Mechanics: Advanced Game Design*. 1st ed. Thousand Oaks, CA, USA: New Riders Publishing, 2012.

[6] N. Ahmadi. "Beyond Upload and Download: Enabling Game Design 2.0". In: *End-User Development – Proceedings of the 3rd International Symposium, IS-EUD 2011, Torre Canne, Italy, June 7–10, 2011*. Vol. 6654. LNCS. Springer, 2011.

[7] N. Ahmadi. "Broadening Educational Game Design using the World Wide Web". PhD thesis. Università della Svizzera Italiana – Faculty of Informatics, 2012.

[8] N. Ahmadi et al. "Engineering an Open-Web Educational Game Design Environment". In: *Proceedings of the 19th Asia-Pacific Software Engineering Conference, APSEC 2012, Hong Kong, China, December 4–7, 2012*. IEEE, 2012.

[9] D. Altunbay et al. "Model-driven Approach for Board Game Development". In: *Proceedings of the 1st Turkish Symposium of Model-Driven Software Development, TMODELS 2009, Ankara, Turkey, May 20, 2009*. Bilkent University, 2009.

[10] V. Alves and L. Roque. "A Pattern Language for Sound Design in Games". In: *Proceedings of the 5th Audio Mostly Conference: A Conference on Interaction with Sound, AM 2010, Piteå, Sweden, September 15–17, 2010*. ACM, 2010.

[11] V. Alves and L. Roque. "A Deck for Sound Design in Games: Enhancements based on a Design Exercise". In: *Proceedings of the 8th International Conference on Advances in Computer Entertainment Technology, ACE 2011, Lisbon, Portugal, November 8–11, 2011*. ACM, 2011.

[12] V. Alves and L. Roque. "An Inspection on a Deck for Sound Design in Games". In: *Proceedings of the 6th Audio Mostly Conference: A Conference on Interaction with Sound, AM 2011, Coimbra, Portugal, September 7–9, 2011*. ACM, 2011.

[13] V. Alves and L. Roque. "Design Patterns in Games; The Case for Sound Design". In: *Workshop Proceedings of the 8th International Conference on the Foundations of Digital Games, as part of the 2nd Workshop on Design Patterns in Games, DPG 2013, Chania, Crete, Greece, May, 14–17, 2013*. Society for the Advancement of the Science of Digital Games, 2013.

[14] E. F. Anderson. "On the Definition of Non-Player Character Behaviour for Real-Time Simulated Virtual Environments". PhD thesis. Bournemouth University, Apr. 2008.

[15] E. F. Anderson. "Scripted Smarts in an Intelligent Virtual Environment: Behaviour Definition Using a Simple Entity Annotation Language". In: *Proceedings of the 2008 Conference on Future Play: Research, Play, Share, Future Play 2008, Toronto, Ontario, Canada, November 3–5, 2008*. ACM, 2008.

[16] M. Araújo and L. Roque. "Modeling Games with Petri Nets". In: *Proceedings of the 3rd annual DiGRA conference Breaking New Ground: Innovation in Games, Play, Practice and Theory, DiGRA 2009, London, UK, September 1–4, 2009*. Digital Games Research Association, 2009.

[17] D. Ašeriškis. “Modeling and Evaluation of Software System Gamification Elements”. PhD thesis. Kaunas University of Technology, 2017.

[18] D. Ašeriškis and R. Damaševičius. “Player Type Simulation in Gamified Applications”. In: *Proceedings of the IVUS International Conference on Information Technology, Kaunas, Lithuania, April 28, 2017*. Vol. 1856. CEUR-WS, 2017.

[19] D. Ašeriškis et al. “UAREI: A Model for Formal Description and Visual Representation /Software Gamification”. In: *DYNA* 84.200 (Mar. 2017).

[20] J. Aycock. “Endgame”. In: *Retrogame Archeology: Exploring Old Computer Games*. Springer, 2016.

[21] A. Azadegan and C. Harteveld. “Work for or Against Players: On the Use of Collaboration Engineering for Collaborative Games”. In: *Proceedings of Workshops Colocated with the 9th International Conference on the Foundations of Digital Games – as part of the 3rd Workshop on Design Patterns in Games, DPG 2014, Liberty of the Seas, Caribbean, April 3–7, 2014*. Society for the Advancement of the Science of Digital Games, 2014.

[22] H. Bäärnhielm et al. “A Haskell Sound Specification DSL: Ludic Support and Deep Immersion in Nordic Technology-Supported LARP”. In: *The Monad Reader* 23 (2014).

[23] D. Balas et al. “Hierarchical Petri Nets for Story Plots Featuring Virtual Humans”. In: *Proceedings of the 4th Conference on Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE 2008, Stanford, California, USA, October 22–24, 2008*. AAAI, 2008.

[24] A. Baldwin et al. “Towards Pattern-Based Mixed-initiative Dungeon Generation”. In: *Proceedings of the 12th International Conference on the Foundations of Digital Games, FDG 2017 as part of the 8th International Workshop on Procedural Content Generation, PCG 2017, Hyannis, Massachusetts, USA, August 14–17, 2017*. ACM, 2017.

[25] A. Baldwin et al. “Mixed-Initiative Procedural Generation of Dungeons using Game Design Patterns”. In: *2017 IEEE Conference on Computational Intelligence and Games, CIG 2017, New York, NY, USA, August 22–25, 2017*. 2017.

[26] G. A. B. Barros and J. Togelius. “Exploring a Large Space of Small Games”. In: *Proceedings of the 2014 IEEE Conference on Computational Intelligence and Games, CIG 2014, Dortmund, Germany, August 26–29, 2014*. 2014.

[27] R. A. Bartle. *MMOs from the Inside Out: The History, Design, Fun, and Art of Massively-Multiplayer Online Role-Playing Games*. Apress, 2016.

[28] A. Begel and E. Klopfer. “Starlogo TNG: An Introduction to Game Development”. In: *Journal of E-Learning* 53 (2005).

[29] C. Bell and M. Goadrich. “Automated Playtesting with RECYCLEd CARDSTOCK”. In: *Game & Puzzle Design* 2.1 (2016).

[30] L. Beyak. “SAGA: A Story Scripting Tool for Video Game Development”. MA thesis. McMaster University – Department of Computing and Software, 2011.

[31] L. Beyak and J. Carette. “SAGA: A DSL for Story Management”. In: *Proceedings IFIP Working Conference on Domain-Specific Languages, DSL 2011, Bordeaux, France, September 6–8, 2011*. Vol. 66. EPTCS. arXiv, 2011.

[32] S. Björk and J. Holopainen. “Games and Design Patterns”. In: *The Game Design Reader: A Rules of Play Anthology*. MIT Press, 2006.

[33] S. Björk et al. “Game Design Patterns”. In: *Proceedings of the 2003 DiGRA International Conference: Level Up, DIGRA 2003, Utrecht, The Netherlands, November 4–6, 2003*. Digital Games Research Association, 2003.

[34] N. Bojin. “Language Games/Game Languages: Examining Game Design Epistemologies Through a ‘Wittgensteinian’ Lens”. In: *Journal for Computer Game Culture* 2.1 (2008).

[35] N. Bojin. “Ludemes and the Linguistic Turn”. In: *Proceedings of the International Academic Conference on the Future of Game Design and Technology, Future Play 2010, Vancouver, British Columbia, Canada, May 6–7, 2010*. ACM, 2010.

[36] Y. C. Borghini. “An Assessment and Learning Analytics Engine for Games-Based Learning”. PhD thesis. University of the West of Scotland, Dec. 2015.

[37] C. Brom and A. Abonyi. “Petri Nets for Game Plot”. In: *Proceedings of Artificial Intelligence and the Simulation of Behaviour as part of the Workshop on Narrative AI and Games*. AISB, 2006.

[38] C. Brom et al. “Story Manager in ’Europe 2045’ Uses Petri Nets”. In: *Virtual Storytelling. Using Virtual Reality Technologies for Storytelling – Proceedings of the 4th International Conference, ICVS 2007, Saint-Malo, France, December 5–7, 2007*. Vol. 4871. LNCS. Springer, 2007.

[39] C. Browne and F. Maire. “Evolutionary Game Design”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 2.1 (Mar. 2010).

[40] C. Browne. “Automatic Generation and Evaluation of Recombination Games”. PhD thesis. Queensland University of Technology, Feb. 2008.

[41] C. Browne. *Evolutionary Game Design*. SpringerBriefs in Computer Science. Springer, 2011.

[42] C. Browne. “A Class Grammar for General Games”. In: *Computers and Games – Proceedings of the 9th International Conference on Computers and Games, CG 2016, Leiden, The Netherlands, June 29–July 1, 2016*. Vol. 10068. LNCS. Springer, 2016.

[43] J. Brusk and T. Lager. “Developing Natural Language Enabled Games in (Extended) SCXML”. In: *Proceedings of the International Symposium on Intelligence Techniques in Computer Games and Simulations, GAME-ON-ASIA 2007, Shiga, Japan, March 1–3, 2007*. EUROSIS, 2007.

[44] J. Brusk. “Dialogue Management for Social Game Characters Using Statecharts”. In: *Proceedings of the 2008 International Conference on Advances in Computer Entertainment Technology, ACE 2008, Yokohama, Japan, December 3–5, 2008*. ACM, 2008.

[45] D. Burgos et al. “Building Adaptive Game-based Learning Resources: The Integration of IMS Learning Design and <e-Adventure>”. In: *Simulation & Gaming* 39.3 (July 2008).

[46] A. Calleja and G. J. Pace. “A Domain-Specific Embedded Language Approach for the Scripting of Game Artificial Intelligence”. In: *Proceedings of the 2nd National Workshop in Information and Communication Technology, WICT 2009, Valletta, Malta, November 17, 2009*. University of Malta, 2009.

[47] A. Calleja and G. J. Pace. “Scripting Game AI: An Alternative Approach using Embedded Languages”. In: *Proceedings of the 3rd National Workshop in Information and Communication Technology, WICT 2010, Valletta, Malta, November 16, 2010*. University of Malta, 2010.

[48] A. Canossa and A. Drachen. “Patterns of Play: Play-Personas in User-Centred Game Development”. In: *Proceedings of the 2009 DiGRA International Conference: Breaking New Ground: Innovation in Games, Play, Practice and Theory, DiGRA 2009, West London, UK, September 1–4, 2009*. Brunel University, 2009.

[49] M. Carbonaro et al. “Interactive Story Authoring: A Viable form of Creative Expression for the Classroom”. In: *Computers & Education* 51.2 (Sept. 2008).

[50] A. J. Champandard. *Behavior Trees for Next-Gen Game AI*. AIGameDev.com. Dec. 2007.

[51] A. J. Champandard. *Understanding the Second-Generation of Behavior Trees – AltDevConf*. AIGameDev.com. Feb. 2012.

[52] C. Chang et al. “Relationships between Engagement and Learning Style for using VPL on Game Design”. In: *Proceedings of the 4th IEEE International Conference on Digital Game and Intelligent Toy Enhanced Learning, DIGITEL 2012, Takamatsu, Japan, March 27–30, 2012*. IEEE, 2012.

[53] Y. Chaudy et al. “EngAGe: A Link between Educational Games Developers and Educators”. In: *Proceedings of the 6th International Conference on Games and Virtual Worlds for Serious Applications, VS-GAMES 2014, Valletta, Malta, September 9–12, 2014*. IEEE, 2014.

[54] D. Church. “Formal Abstract Design Tools”. In: *Gamasutra* (July 1999).

[55] D. Church. “Formal Abstract Design Tools”. In: *Game Developer* (Aug. 1999).

[56] K. Compton et al. “Tracery: An Author-Focused Generative Text Tool”. In: *Interactive Storytelling – Proceedings of the 8th International Conference on Interactive Digital Storytelling, ICIDS 2015, Copenhagen, Denmark, November 30–December 4, 2015*. Vol. 9445. LNCS. Springer, 2015.

[57] M. J. Conway. “Alice: Easy-to-Learn 3D Scripting for Novices”. PhD thesis. University of Virginia, Dec. 1997.

[58] M. Conway et al. “Alice: Lessons Learned from Building a 3D System for Novices”. In: *Proceedings of the CHI 2000 Conference on Human factors in computing systems, The Hague, The Netherlands, April 1–6, 2000*. ACM, 2000.

[59] M. Cook et al. “Mechanic Miner: Reflection-Driven Game Mechanic Discovery and Level Design”. In: *Applications of Evolutionary Computation – Proceedings of the 16th European Conference, EvoApplications 2013, Vienna, Austria, April 3–5, 2013*. Vol. 7835. LNCS. Springer, 2013.

[60] M. Cook et al. “Nobody’s A Critic: On The Evaluation Of Creative Code Generators – A Case Study In Video Game Design”. In: *Proceedings of the Fourth International Conference on Computational Creativity, ICCC 2013, Sidney, Australia, June 12–14, 2013*. computational-creativity.net, 2013.

[61] M. L. Crane and J. Dingel. “UML vs. Classical vs. Rhapsody Statecharts: Not all Models are Created Equal”. In: *Software and System Modeling* 6.4 (2007).

[62] M. Cutumisu. “Using Behaviour Patterns to Generate Scripts for Computer Role-Playing Games”. PhD thesis. University of Alberta, 2009.

[63] M. Cutumisu et al. “ScriptEase: A Generative/Adaptive Programming Paradigm for Game Scripting”. In: *Science of Computer Programming* 67.1 (June 2007).

[64] M. Cutumisu et al. “Evaluating Pattern Catalogs: The Computer Games Experience”. In: *Proceedings of the 28th International Conference on Software Engineering, ICSE 2006, Shanghai, China, May 20–28, 2006*. ACM, 2006.

[65] B. Dawson. “GDC 2002: Game Scripting in Python”. In: *Gamasutra* (Aug. 2002).

[66] O. de Troyer et al. “Creating Story-Based Serious Games Using a Controlled Natural Language Domain Specific Modeling Language”. In: *Serious Games and Edutainment Applications: Volume II*. Springer, 2017.

[67] A. Denault et al. “Model-Based Design of Game AI”. In: *Proceedings of the 2nd International North American Conference on Intelligent Games and Simulation, GAME-ON-NA 2006, Monterey, USA, September 19–20, 2006*. EUROSIS, 2006.

[68] F. di Giacomo. “Design of an Optimized Compiler for Casanova Language”. MA thesis. Università Ca’Foscari Venezia – Corso di Laurea magistrale in Informatica, 2014.

[69] F. di Giacomo et al. “Building Game Scripting DSLs with the Metacasanova Metacompiler”. In: *Intelligent Technologies for Interactive Entertainment – Proceedings of the 8th International Conference, Revised Selected Papers, INTETAIN 2016, Utrecht, The Netherlands, June 28–30, 2016*. Vol. 178. LNICST. Springer, 2016.

[70] F. di Giacomo et al. “High Performance Encapsulation and Networking in Casanova 2”. In: *Entertainment Computing* 20 (May 2017).

[71] F. di Giacomo et al. “Metacasanova: An Optimized Meta-Compiler for Domain-Specific Languages”. In: *Proceedings of the 10th ACM SIGPLAN International Conference on Software Language Engineering, SLE 2017, Vancouver, BC, Canada, October 23–24, 2017*. ACM, 2017.

[72] J. Dormans. “Machinations: Elemental Feedback Patterns for Game Design”. In: *Proceedings of the 5th International North American Conference on Intelligent Games and Simulation, GAME-ON-NA 2009, Atlanta, USA, August 26–28, 2009*. EUROSIS, 2009.

[73] J. Dormans. “Adventures in Level Design: Generating Missions and Spaces for Action Adventure Games”. In: *Proceedings of the 1st Workshop on Procedural Content Generation in Games, PCG 2010, Monterey, California, USA, June 18, 2010*. ACM, 2010.

[74] J. Dormans. “Level Design as Model Transformation: A Strategy for Automated Content Generation”. In: *Proceedings of the 2nd Workshop on Procedural Content Generation in Games, PCG 2011, Bordeaux, France, June 28, 2011*. ACM, 2011.

[75] J. Dormans. “Simulating Mechanics to Study Emergence in Games”. In: *Workshops at the 7th Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE 2011, as part of the workshop on Artificial Intelligence in the Game Design Process, Stanford University, October 10–14, 2011*. Vol. WS-11-19. AAAI Workshops. AAAI, 2011.

[76] J. Dormans. “Engineering Emergence: Applied Theory for Game Design”. PhD thesis. University of Amsterdam, 2012.

[77] J. Dormans. “Generating Emergent Physics for Action-Adventure Games”. In: *Proceedings of the 3rd Workshop on Procedural Content Generation in Games, PCG 2012, Raleigh, NC, USA, May 29–June 01, 2012*. ACM, 2012.

[78] J. Dormans and S. Bakkes. “Generating Missions and Spaces for Adaptable Play Experiences”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.3 (Sept. 2011).

[79] J. Dormans and S. Leijnen. “Combinatorial and Exploratory Creativity in Procedural Content Generation”. In: *Workshop Proceedings of the 8th International Conference on the Foundations of Digital Games, as part of the 4th Workshop on Procedural Content Generation in Games, PCG 2013, Chania, Crete, Greece, May, 14–17, 2013*. Society for the Advancement of the Science of Digital Games, 2013.

[80] C. Dowd. “The Scrabble of Language towards Persuasion: Changing Behaviors in Journalism”. In: *Persuasive Technology – Proceedings of the 8th International Conference, PERSUASIVE 2013, Sydney, NSW, Australia, April 3–5, 2013*. Vol. 7822. LNCS. Springer, 2013.

[81] M. Ebner et al. “Towards a Video Game Description Language”. In: *Artificial and Computational Intelligence in Games*. Vol. 6. Dagstuhl Follow-Ups. Schloss Dagstuhl - Leibniz-Zentrum für Informatik, 2013.

[82] C. Elverdam and E. Aarseth. “Game Classification and Game Design: Construction Through Critical Analysis”. In: *Games and Culture* 2.1 (Jan. 2007).

[83] R. Evans and E. Short. “Versu–A Simulationist Storytelling System”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 6.2 (June 2014).

[84] P. Féher and L. Lengyel. “The Power of Graph Transformation – Implementing a Shadow Casting Algorithm”. In: *Proceedings of the 10th Jubilee International Symposium on Intelligent Systems and Informatics, SISY 2012, Subotica, Serbia, October 25, 2012*. IEEE, 2012.

[85] M. Flatt. “Creating Languages in Racket”. In: *Queue* 9.11 (Nov. 2011).

[86] M. Flatt. “Creating Languages in Racket”. In: *Communications of the ACM* 55.1 (Jan. 2012).

[87] J. M. Font et al. “A Card Game Description Language”. In: *Applications of Evolutionary Computation – Proceedings of the 16th European Conference, EvoApplications 2013, Vienna, Austria, April 3–5, 2013*. Vol. 7835. LNCS. Springer, 2013.

[88] J. M. Font et al. “Towards the Automatic Generation of Card Games through Grammar-Guided Genetic Programming”. In: *Proceedings of the 8th International Conference on the*

Foundations of Digital Games, FDG 2013, Chania, Crete, Greece, May 14–17, 2013. Society for the Advancement of the Science of Digital Games, 2013.

[89] F. Frapolli et al. "Decoupling Aspects in Board Game Modeling". In: *International Journal of Gaming and Computer-Mediated Simulations* 2.2 (Apr. 2010).

[90] F. Frapolli et al. "Exploiting Traditional Gameplay Characteristics to Enhance Digital Board Games". In: *Proceedings of the 2nd International IEEE Consumer Electronics Society's Games Innovations Conference, GiC 2010, Hong Kong, China, December 21–23, 2010*. IEEE, 2010.

[91] F. Frapolli et al. "FLEXIBLE RULES: A Player Oriented Board Game Development Framework". In: *Proceedings of the 3rd International Conference on Advances in Computer-Human Interactions, ACHI 2010, Sint Maarten, Netherlands, Antilles, February 10–16, 2010*. IEEE, 2010.

[92] D. Fu and R. T. Houlette. "Putting AI in Entertainment: An AI Authoring Tool for Simulation and Games". In: *IEEE Intelligent Systems* 17.4 (July 2002).

[93] D. Fu et al. "A Visual Environment for Rapid Behavior Definition". In: *Prococeedings of the 12th Conference on Behavior Representation in Modeling and Simulation, BRIMS 2003, Scottsdale, Arizona, USA, May 12–15, 2003*. SISO, 2003.

[94] D. Fu et al. "An AI Modeling Tool for Designers and Developers". In: *IEEE Aerospace Conference Proceedings, Big Sky, MT, USA, March 3–10, 2007*. IEEE, 2007.

[95] A. W. B. Furtado et al. "Improving Digital Game Development with Software Product Lines". In: *IEEE Software* 28.5 (Sept. 2011).

[96] A. W. B. Furtado and A. L. M. Santos. "Tutorial: Applying Domain-Specific Modeling to Game Development with the Microsoft DSL Tools". In: *Proceedings of the 5th Brazilian Symposium on Computer Games and Digital Entertainment, SBGames 2006, Recife, Brazil, November 8–10, 2006*. UFPE, 2006.

[97] A. W. B. Furtado and A. L. M. Santos. "Using Domain-Specific Modeling Towards Computer Games Development Industrialization". In: *Proceedings of the 6th Workshop on Domain-Specific Modeling, DSM 2006, Portland, Oregon, USA, October 22, 2006*. Computer Science and Information System Reports, Technical Reports, TR-37. University of Jyväskylä, Finland, 2006.

[98] A. W. B. Furtado et al. "SharpLudus Revisited: From Ad Hoc and Monolithic Digital Game DSLs to Effectively Customized DSM Approaches". In: *SPLASH '11 Workshops: Proceedings of the Compilation of the Co-located Workshops on DSM'11, TMC'11, AGERE! 2011, AOOPES'11, NEAT'11, & VMIL'11 – as part of the 11th workshop on Domain-Specific Modeling, DSM 2011, Portland, Oregon, USA, October 23–24, 2011*. ACM, 2011.

[99] A. W. B. Furtado. "SharpLudus: Improving Game Development Experience Through Software Factories and Domain-Specific Languages". MA thesis. Universidade Federal de Pernambuco (UFPE) – Mestrado em Ciência da Computação – Centro de Informática (CIN), 2006.

[100] A. W. B. Furtado. "Domain-Specific Game Development". PhD thesis. Universidade Federal de Pernambuco, 2012.

[101] A. W. B. Furtado et al. "A Computer Games Software Factory and Edutainment Platform for Microsoft.NET". In: *IET Software* 1.6 (Dec. 2007).

[102] I. A. Games. "Gamestar Mechanic: Learning a Designer Mindset through Communicational Competence with the Language of Games". In: *Learning, Media and Technology* 35.1 (Mar. 2010).

[103] S. Gaudl. "Building Robust Real-Time Game AI: Simplifying & Automating Integral Process Steps in Multi-Platform Design". PhD thesis. University of Bath, May 2016.

[104] S. E. Gaudl et al. "Behaviour Oriented Design for Real-Time-Strategy Games". In: *Proceedings of the 8th International Conference on the Foundations of Digital Games, FDG 2013, Chania,*

Crete, Greece, May 14–17, 2013. Society for the Advancement of the Science of Digital Games, 2013.

[105] A. Grow et al. "A Methodology for Requirements Analysis of AI Architecture Authoring Tools". In: *Proceedings of the 9th International Conference on the Foundations of Digital Games, FDG 2014, Liberty of the Seas, Caribbean, April 3–7, 2014*. Society for the Advancement of the Science of Digital Games, 2014.

[106] V. Guana. "End-to-end Fine-grained Traceability Analysis in Model Transformations and Transformation Chains". PhD thesis. Department of Computing Science – University of Alberta, 2017.

[107] V. Guana and E. Stroulia. "PhyDSL: A Code-generation Environment for 2D Physics-based Games". In: *IEEE Games, Entertainment, and Media Conference, GEM 2014, Toronto ON Canada, October 22–24, 2014*. IEEE, 2014.

[108] V. Guana et al. "Building a Game Engine: A Tale of Modern Model-Driven Engineering". In: *Proceedings of the 4th International Workshop on Games and Software Engineering, GAS 2015, Florence, Italy, May 18, 2015*. IEEE, 2015.

[109] H. Guo. "Concepts and Modelling Techniques for Pervasive and Social Games". PhD thesis. Norwegian University of Science et al., June 2015.

[110] H. Guo et al. "PerGO: An Ontology Towards Model Driven Pervasive Game Development". In: *On the Move to Meaningful Internet Systems: OTM 2014 Workshops, as part of Ontologies, DataBases, and Applications of Semantics, ODBASE 2014 Posters, Amantea, Italy, October 27–31, 2014*. Vol. 8842. LNCS. Springer, 2014.

[111] H. Guo et al. "A Workflow for Model Driven Game Development". In: *Proceedings of the IEEE 19th International Enterprise Distributed Object Computing Conference, EDOC 2015, Adelaide, SA, Australia, September 21–25, 2015*. IEEE, 2015.

[112] H. Guo et al. "RealCoins: A Case Study of Enhanced Model Driven Development for Pervasive Games". In: *International Journal of Multimedia and Ubiquitous Engineering* 10.5 (2015).

[113] T. Hastjarjanto. "Strategies for Real-Time Video Games". MA thesis. Utrecht University, Mar. 2013.

[114] T. Hastjarjanto et al. "A DSL for Describing the Artificial Intelligence in Real-time Video Games". In: *Proceedings of the 3rd International Workshop on Games and Software Engineering: Engineering Computer Games to Enable Positive, Progressive Change, GAS 2013, San Francisco, CA, USA, May 18–26, 2013*. IEEE, 2013.

[115] F. E. Hernandez and F. R. Ortega. "Eberos GML2D: A Graphical Domain-Specific Language for Modeling 2D Video Games". In: *Proceedings of the 10th Workshop on Domain-Specific Modeling, DSM 2010, Reno/Tahoe, Nevada, USA, October 17–18, 2010*. Aalto-Print, 2010.

[116] P. Herzig et al. "GaML: A Modeling Language for Gamification". In: *Proceedings of the IEEE/ACM 6th International Conference on Utility and Cloud Computing, Dresden, Germany, December 9–12, 2013*. IEEE, 2013.

[118] J. Holopainen and S. Björk. "Game Design Patterns – Lecture Notes". In: *Game Developers Conference, GDC 2003*. 2003.

[119] J. Holopainen and S. Björk. "Gameplay Design Patterns for Motivation". In: *Games: Virtual Worlds and Reality – Proceedings of the 39th Conference of the International Simulation And Gaming Association, ISAGA 2008, Kaunas, Lithuania, July 7–11, 2008*. Kaunas University of Technology, 2008.

[120] J. Holopainen et al. "Teaching Gameplay Design Patterns". In: *Organizing and Learning through Gaming and Simulation – Proceedings of the 38th Conference of the International*

Simulation And Gaming Association, ISAGA 2007, Nijmegen, The Netherlands, July 9–13, 2007. Eburon, 2007.

[121] K. Hullett and J. Whitehead. "Design Patterns in FPS Levels". In: *Proceedings of the Fifth International Conference on the Foundations of Digital Games, FDG 2010, Monterey, California, USA, June 19–21, 2010*. ACM, 2010.

[122] R. Hunicke et al. "MDA: A Formal Approach to Game Design and Game Research". In: *Proceedings of the AAAI workshop on Challenges in Game Artificial Intelligence*. AAAI, 2004.

[123] R. Ierusalimschy et al. "The Implementation of Lua 5.0". In: *Journal of Universal Computer Science* 11.7 (2005).

[124] R. Ierusalimschy et al. "The Evolution of Lua". In: *Proceedings of the 3rd ACM SIGPLAN Conference on History of Programming Languages, HOPL III, San Diego, California, June 9–10, 2007*. ACM, 2007.

[125] A. Ioannidou et al. "Using Scalable Game Design to Promote 3D Fluency: Assessing the AgentCubes incremental 3D End-user Development Framework". In: *Proceedings of the 2008 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2008, Herrsching am Ammersee, Germany, September 15–19, 2008*. IEEE, 2008.

[126] D. Isla. "GDC 2005 Proceeding: Handling Complexity in the Halo 2 AI". In: *Gamasutra* (Mar. 2005).

[127] D. Isla. "Managing Complexity in the Halo 2 AI System". In: *Proceedings of the Game Developers Conference, GDC 2005*. Gdcvault.com, 2005.

[128] R. Jones. "Rapid Game Development in Python". In: *OpenSource Developers' Conference*. 2005.

[129] D. Karavolos et al. "Mixed-Initiative Design of Game Levels: Integrating Mission and Space into Level Generation". In: *Proceedings of the 10th International Conference on the Foundations of Digital Games, FDG 2015, Pacific Grove, CA, USA, June 22–25, 2015*. Society for the Advancement of the Science of Digital Games, 2015.

[130] J. Kessing et al. "Designing Semantic Game Worlds". In: *Proceedings of the 3rd workshop on Procedural Content Generation in Games, PCG 2012, Raleigh, NC, USA, May 29–June 1, 2012*. ACM, 2012.

[131] J. Kienzle et al. "Model-Based Design of Computer-Controlled Game Character Behavior". In: *Model Driven Engineering Languages and Systems – Proceedings of the 10th International Conference, MoDELS 2007, Nashville, USA, September 30–October 5, 2007*. Vol. 4735. LNCS. Springer, 2007.

[132] P. Klint and R. van Rozen. "Micro-Machinations: a DSL for Game Economies". In: *Software Language Engineering – Proceedings of the 6th International Conference on Software Language engineering, SLE 2013, Indianapolis, IN, USA, October 26–28, 2013*. Vol. 8225. LNCS. Springer, 2013.

[133] P. Klint et al. "Game Developers Need Lua AiR: Static Analysis of Lua Using Interface Models". In: *Entertainment Computing – Proceedings of the 11th International Conference on Entertainment Computing, ICEC 2012, as part of the 2nd Workshop on Game Development and Model-Driven Software Development, GD&MDSD 2012, Bremen, Germany, September 26–29, 2012*. Vol. 7522. LNCS. Springer, 2012.

[134] R. Koster. "A Grammar of Gameplay". In: *Game Developers Conference, GDC 2005*. 2005.

[135] R. Koster. *Theory of Fun for Game Design*. 1st ed. Paraglyph Press, 2005.

[136] R. Koster. "The Limits of Formalism". In: *Presentation delivered at the BIRS Workshop on Computational Modeling in Games*. Raph Koster's Website, 2016.

[137] B. Kreimeier. "The Case for Game Design Patterns". In: *Gamasutra* (Mar. 2002).

[138] P. Lemay. “Developing a Pattern Language for Flow Experiences in Video Games”. In: *Proceedings of the 2007 DiGRA International Conference: Situated Play, DiGRA 2007, Tokyo, Japan, September 24–28, 2007*. The University of Tokyo, 2007.

[139] A. Liapis et al. “Sentient Sketchbook: Computer-Aided Game Level Authoring”. In: *Proceedings of the 8th International Conference on the Foundations of Digital Games, FDG 2013, Chania, Crete, Greece, May 14–17, 2013*. Society for the Advancement of the Science of Digital Games, 2013.

[140] C. U. Lim and D. F. Harrell. “An Approach to General Videogame Evaluation and Automatic Generation using a Description Language”. In: *Proceedings of the 2014 IEEE Conference on Computational Intelligence and Games, CIG 2014, Dortmund, Germany, August 26–29, 2014*. IEEE, 2014.

[141] C. Lim et al. “Evolving Behaviour Trees for the Commercial Game DEFCON”. In: *Applications of Evolutionary Computation, EvoApplicatons 2010: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC, Proceedings, Part I – as part of EvoGAMES, Istanbul, Turkey, April 7–9, 2010*. Vol. 6024. LNCS. Springer, 2010.

[142] A. Lindenmayer. “Mathematical Models for Cellular Interactions in Development”. In: *Journal of Theoretical Biology* 18.3 (1968).

[143] N. Love et al. “General Game Playing: Game Description Language Specification”. In: (Mar. 2008).

[144] M. B. MacLaurin. “The Design of Kodu: A Tiny Visual Programming Language for Children on the Xbox 360”. In: *Proceedings of the 38th Annual ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2011, Austin, Texas, USA, January 26–28, 2011*. ACM, 2011.

[145] M. B. MacLaurin. “The Design of Kodu: A Tiny Visual Programming Language for Children on the Xbox 360”. In: *SIGPLAN Notices* 46.1 (Jan. 2011).

[146] G. Maggiore et al. “Monadic Scripting in F# for Computer Games”. In: *Proceedings of the 5th International Workshop on Harnessing Theories for Tool Support in Software, TTSS 2011, Oslo Norway, September 13, 2011*. UIO, 2011.

[147] G. Maggiore et al. “A Formal Specification for Casanova, a Language for Computer Games”. In: *Proceedings of the 4th ACM SIGCHI Symposium on Engineering Interactive Computing Systems, EICS 2012, Copenhagen, Denmark, June 25–26, 2012*. ACM, 2012.

[148] G. Maggiore et al. “Designing Casanova: A Language for Games”. In: *Advances in Computer Games – Proceedings of the 13th International Conference, Revised Selected Papers, ACG 2011, Tilburg, The Netherlands, November 20–22, 2011*. Vol. 7168. LNCS. Springer, 2012.

[149] G. Maggiore et al. “Writing Real-Time .Net Games in Casanova”. In: *Entertainment Computing – Proceedings of the 11th International Conference on Entertainment Computing, ICEC 2012, Bremen, Germany, September 26-29, 2012*. Vol. 7522. LNCS. Springer, 2012.

[150] G. Maggiore. “Casanova: A Language for Game Development”. PhD thesis. Università Ca’ Foscari di Venezia, Dec. 2012.

[151] T. Mahlmann. “Modelling and Generating Strategy Games Mechanics”. PhD thesis. IT University of Copenhagen, Mar. 2013.

[152] T. Mahlmann et al. “Towards Procedural Strategy Game Generation: Evolving Complementary Unit Types”. In: *Applications of Evolutionary Computation – Proceedings of EvoApplications 2011: EvoCOMPLEX, EvoGAMES, EvoIASP, EvoINTELLIGENCE, EvoNUM, and EvoSTOC, Torino, Italy, April 27–29, 2011*. Vol. 6624. LNCS. Springer, 2011.

[153] T. Mahlmann et al. “Modelling and Evaluation of Complex Scenarios with the Strategy Game Description Language”. In: *Proceedings of the 2011 IEEE Conference on Computational Intelligence and Games, CIG 2011, Seoul, South Korea, August 31–September 3, 2011*. 2011.

[154] 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, Future Play 2008, Toronto, Ontario, Canada, November 3–5, 2008*. ACM, 2008.

[155] E. J. Marchiori et al. "A Narrative Metaphor to Facilitate Educational Game Authoring". In: *Computers & Education* 58.1 (2012).

[156] B. Marne et al. "A Design Pattern Library for Mutual Understanding and Cooperation in Serious Game Design". In: *Intelligent Tutoring Systems – Proceedings of the 11th International Conference, ITS 2012, Chania, Crete, Greece, June 14–18, 2012*. Vol. 7315. LNCS. Springer, 2012.

[157] C. Martens. "Ceptre: A Language for Modeling Generative Interactive Systems". In: *Proceedings of the 11th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2015, University of California, Santa Cruz, USA, November 14–18, 2015*. AAAI, 2015.

[158] C. Martens et al. "Generative Story Worlds as Linear Logic Programs". In: *Proceedings of the 7th Intelligent Narrative Technologies Workshop, Wisconsin, USA, June 17–18, 2014*. AAAI, 2014.

[159] C. Martens et al. "A Resourceful Reframing of Behavior Trees". In: *CoRR* abs/1803.09099 (2018).

[160] M. Masuch and M. Rueger. "Challenges in Collaborative Game Design: Developing Learning Environments for Creating Games". In: *Proceedings of the 3rd International Conference on Creating, Connecting and Collaborating through Computing, C5 2005, Kyoto, Japan, Japan, January 28–29, 2005*. IEEE, 2005.

[161] A. Matallaoui et al. "Model-Driven Serious Game Development Integration of the Gamification Modeling Language GaML with Unity". In: *Proceedings of the 48th Annual Hawaii International Conference on System Sciences, HICSSS 2015, Kauai, HI, USA, January 5–8, 2015*. IEEE, 2015.

[162] M. Mateas and A. Stern. "A Behavior Language for Story-Based Believable Agents". In: *IEEE Intelligent Systems* 17.4 (July 2002).

[163] M. Mateas and W.-F. Noah. "Defining Operational Logics". In: *Proceedings of the 2009 DiGRA International Conference: Breaking New Ground: Innovation in Games, Play, Practice and Theory, DiGRA 2009, London, UK, September 1–4, 2009*. Brunel University and Digital Games Research Association, 2009.

[164] M. Mateas and A. Stern. "Build It to Understand It: Ludology Meets Narratology in Game Design Space". In: *Proceedings of the 2005 DiGRA International Conference: Changing Views: Worlds in Play, DiGRA 2005, Vancouver, Canada, June 16–20, 2005*. Digital Games Research Association, 2005.

[165] M. Mateas and A. Stern. "Structuring Content Within the Façade Interactive Drama Architecture". In: *Proceedings of the 1st AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2005, Marina del Rey, California, USA, June 1–3, 2005*. AAAI, 2005.

[166] M. Mayer. "Interactive Programming by Example". PhD thesis. École Polytechnique Fédérale de Lausanne, Apr. 2017.

[167] M. Mayer and V. Kuncak. "Game Programming by Demonstration". In: *Proceedings of the 2013 ACM International Symposium on New Ideas, New Paradigms, and Reflections on Programming & Software, Onward! 2013, Indianapolis, Indiana, USA, October 29–31, 2013*. ACM, 2013.

[168] M. McNaughton et al. "Pattern-based AI Scripting using ScriptEase". In: *Proceedings of the 16th Canadian Society for Computational Studies of Intelligence Conference on Advances in Artificial Intelligence, AI 2003, Halifax, Canada, June 11–13, 2003*. Springer, 2003.

[169] M. McNaughton et al. "ScriptEase: Generating Scripting Code for Computer Role-Playing Games". In: *Proceedings of the 19th International Conference on Automated Software Engineering, ASE 2004, Linz, Austria, September 20–24, 2004*. IEEE, 2004.

[170] M. McNaughton et al. "ScriptEase: Generative Design Patterns for Computer Role-Playing Games". In: *Proceedings of the 19th International Conference on Automated Software Engineering, ASE 2004, Linz, Austria, September 20–24, 2004*. IEEE, 2004.

[171] F. Mehm. "Authoring of Adaptive Single-Player Educational Games". PhD thesis. Technische Universität Darmstadt, Jan. 2013.

[172] F. Mehm et al. "Authoring Environment for Story-Based Digital Educational Games". In: *Proceedings of the 1st International Open Workshop on Intelligent Personalization and Adaptation in Digital Educational Games*. 2009.

[173] F. Mehm et al. "Bat Cave: A Testing and Evaluation Platform for Digital Educational Games". In: *Proceedings of the 4th European Conference on Games Based Learning, ECGBL 2010, Copenhagen, Denmark, October 21–22, 2010*. Academic Conferences International Limited, 2010.

[174] F. Mehm et al. "Authoring Processes and Tools". In: *Serious Games: Foundations, Concepts and Practice*. Springer, 2016.

[175] E. Montero Reyno and J. Á. Carsí Cubel. "A Platform-Independent Model for Videogame Gameplay Specification". In: *Proceedings of the 2009 DiGRA International Conference: Breaking New Ground: Innovation in Games, Play, Practice and Theory, DiGRA 2009, West London, UK, September 1–4, 2009*. Brunel University, 2009.

[176] E. Montero Reyno and J. Á. Carsí Cubel. "Model Driven Game Development: 2D Platform Game Prototyping". In: *Proceedings of the 9th International Conference on Intelligent Games and Simulation, GAME-ON 2008, Valencia, Spain, November 17–19, 2008*. EUROSIS, 2008.

[177] E. Montero Reyno and J. Á. Carsí Cubel. "Automatic Prototyping in Model-driven Game Development". In: *Computers in Entertainment – Special Issue on Media Arts and Games* 7.2 (June 2009).

[178] P. Moreno-Ger et al. "Language-Driven Development of Videogames: The <e-Game> Experience". In: *Entertainment Computing – Proceedings of the 5th International Conference on Entertainment Computing, ICEC 2006, Cambridge, UK, September 20–22, 2006*. Vol. 4161. LNCS. Springer, 2006.

[179] P. Moreno-Ger et al. "A Documental Approach to Adventure Game Development". In: *Science of Computer Programming – Special Issue on Aspects of Game Programming* 67.1 (June 2007).

[180] P. Moreno-Ger et al. "An eLearning Specification Meets a Game: Authoring and Integration with IMS Learning Design and <e-Adventure>". In: *Organizing and Learning through Gaming and Simulation – Proceedings of the 38th Conference of the International Simulation And Gaming Association, ISAGA 2007, Nijmegen, The Netherlands, July 9–13, 2007*. Eburon, 2007.

[181] P. Moreno-Ger et al. "Model-Checking for Adventure Videogames". In: *Information and Software Technology* 51.3 (2009).

[182] J. B. Mossmann et al. "Project and Preliminary Evaluation of VR-MED, a Domain-Specific Language for Serious Games in Family Medicine Teaching". In: *Proceedings of the IEEE 40th Annual Computer Software and Applications Conference, COMPSAC 2016, Atlanta, Georgia, USA, June 10–14 2016*. IEEE, 2016.

[183] T. Murata. "Petri nets: Properties, analysis and applications". In: *Proceedings of the IEEE* 77.4 (Apr. 1989).
[184] M. S. El-Nasr et al. *Game Analytics*. Springer, 2016.
[185] S. Natkin and L. Vega. "Petri Net Modelling for the Analysis of the Ordering of Actions in Computer Games". In: *Proceedings of the 4th International Conference on Intelligent Games and Simulation, GAME-ON 2003, London, UK, November 19–21, 2003*. EUROSIS, 2003.
[186] S. Natkin et al. "A new Methodology for Spatiotemporal Game Design". In: *Proceedings of the 5th Game-On International Conference on Computer Games: Artificial Intelligence, Design and Education, CGAIDE 2004, Reading, UK, November 8–10, 2004*. University of Wolverhampton, 2004.
[187] K. Neil. "Game Design Tools: Can They Improve Game Design Practice?" PhD thesis. Flinders University, Dec. 2015.
[188] M. J. Nelson and M. Mateas. "Towards Automated Game Design". In: *Artificial Intelligence and Human-Oriented Computing – Proceedings of the 10th Congress of the Italian Association for Artificial Intelligence, AI*IA 2007, Rome, Italy, September 10–13, 2007*. Vol. 4633. LNCS. Springer, 2007.
[189] M. J. Nelson and M. Mateas. "An Interactive Game-Design Assistant". In: *Proceedings of the 13th International Conference on Intelligent User Interfaces, IUI 2008, Gran Canaria, Spain, January 13–16, 2008*. ACM, 2008.
[190] M. Nelson and M. Mateas. "Recombinable Game Mechanics for Automated Design Support". In: *Proceedings of the 4th Conference on Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE 2008, Stanford, California, USA, October 22–24, 2008*. AAAI, 2008.
[191] T. Nishimori and Y. Kuno. "Mogemoge: A Programming Language Based on Join Tokens". In: *Proceedings of The International Workshop on Information Science Education & Programming Languages, Korean University & University of Tsukuba*. 2006.
[192] T. Nishimori and Y. Kuno. "Join Token: A Language Mechanism for Programming Interactive Games". In: *Entertainment Computing* 3.2 (May 2012).
[193] T. Nishimori and Y. Kuno. "Join Token-Based Event Handling: A Comprehensive Framework for Game Programming". In: *Software Language Engineering – Proceedings of the 4th International Conference, SLE 2011, Braga, Portugal, July 3–4, 2011, Revised Selected Papers*. Vol. 6940. LNCS. Springer, 2012.
[194] T. Nummenmaa. "Adding Probabilistic Modeling to Executable Formal DisCo Specifications with Applications in Strategy Modeling in Multiplayer Game Design". MA thesis. University of Tampere – Department of Computer Sciences, June 2008.
[195] T. Nummenmaa et al. "Exploring Games as Formal Models". In: *Proceedings of the 4th South-East European Workshop on Formal Methods, SEEFM 2009, Thessalonihi, Greece, December 4–5, 2009*. IEEE, 2009.
[196] T. Nummenmaa et al. "Simulation as a Game Design Tool". In: *Proceedings of the International Conference on Advances in Computer Entertainment Technology, ACE 2009, Athens, Greece, October 29–31, 2009*. ACM, 2009.
[197] F. R. Ortega et al. "Exploring Modeling Language for Multi-touch Systems Using Petri Nets". In: *Proceedings of the 2013 ACM International Conference on Interactive Tabletops and Surfaces, ITS 2013, St. Andrews, Scotland, UK, October 6–9, 2013*. ACM, 2013.
[198] J. Orwant. "EGGG: The Extensible Graphical Game Generator". PhD thesis. Massachusetts Institute of Technology, Feb. 2000.
[199] J. Orwant. "EGGG: Automated Programming for Game Generation". In: *IBM Systems Journal* 39.3 (2000).

[200] J. C. Osborn. “Operationalizing Operational Logics”. PhD thesis. UC Santa Cruz, June 2018.
[201] J. C. Osborn et al. “Modular Computational Critics for Games”. In: *Proceedings of the 9th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2013, Boston, USA, October 14–15, 2013*. AAAI, 2013.
[202] J. C. Osborn et al. “Automated Game Design Learning”. In: *IEEE Conference on Computational Intelligence and Games, CIG 2017, New York, NY, USA, August 22–25, 2017*. IEEE, 2017.
[203] J. C. Osborn et al. “Refining Operational Logics”. In: *Proceedings of the International Conference on the Foundations of Digital Games, FDG 2017, Hyannis, MA, USA, August 14–17, 2017*. ACM, 2017.
[204] J. Osborn et al. “Playspecs: Regular Expressions for Game Play Traces”. In: *Proceedings of the 11th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2015, University of California, Santa Cruz November 14–18, 2015*. AAAI, 2015.
[205] M. Overmars. “Teaching Computer Science Through Game Design”. In: *Computer* 37.4 (Apr. 2004).
[206] M. Overmars. “Learning Object-Oriented Design by Creating Games”. In: *IEEE Potentials* 23.5 (2004).
[207] R. F. Paige et al. “Game Development using Design-by-Contract”. In: *Journal of Object Technology* 5.7 (Sept. 2006).
[208] J. D. Palmer. “Ficticious: MicroLanguages for Interactive Fiction”. In: *Proceedings of the ACM International Conference Companion on Object Oriented Programming Systems Languages and Applications Companion, SPLASH 2010, as part of Onward! 1010, Reno/Tahoe, Nevada, USA, October 17–21, 2010*. ACM, 2010.
[209] D. Phillips. *Creating Apps in Kivy: Mobile with Python*. O’Reilly, 2014.
[210] C. J. F. Pickett et al. “(P)NFG: A Language and Runtime System for Structured Computer Narratives”. In: *Proceedings of the 1st International North American Conference on Intelligent Games and Simulation, GAME-ON-NA 2005, Montreal, Canada, August 22–23, 2005*. EUROSIS, 2005.
[211] D. Pizzi et al. “Automatic Generation of Game Level Solutions as Storyboards”. In: *Proceedings of the 4th Conference on Artificial Intelligence and Interactive Digital Entertainment Conference, AIIDE 2008, Stanford, California, USA, October 22–24, 2008*. AAAI, 2008.
[212] D. Pizzi et al. “Automatic Generation of Game Level Solutions as Storyboards”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 2.3 (Sept. 2010).
[213] T. Pløhn et al. “Extending the Pervasive Game Ontology through a Case Study”. In: *NOKOBIT* 23.1 (2015).
[214] A. Repenning. “Making Programming More Conversational”. In: *Proceedings of the 2011 IEEE Symposium on Visual Languages and Human-Centric Computing, VL/HCC 2011, Pittsburgh, Pennsylvania, USA, September 18–22, 2011*. IEEE, 2011.
[215] A. Repenning and T. Sumner. “AgentSheets: A Medium for Creating Domain-Oriented Languages”. In: *IEEE Computer* 28.3 (Mar. 1995).
[216] M. Resnick et al. “Scratch: Programming for All”. In: *Communications of the ACM* 52.11 (Nov. 2009).
[217] J. W. Romein et al. “An Application Domain Specific Language for Describing Board Games”. In: *Parallel and Distributed Processing Techniques and Applications*. CSREA, 1997.
[218] J. W. Romein. “Multigame - An Environment for Distributed Game-Tree Search”. PhD thesis. Vrije Universiteit Amsterdam, Jan. 2001.
[219] J. W. Romein et al. *The Multigame Reference Manual*. Tech. rep. Vrije Univeristeit Amsterdam, 2000.

[220] G. Russell et al. “Tackling Online Game Development Problems with a Novel Network Scripting Language”. In: *Proceedings of the 7th ACM SIGCOMM Workshop on Network and System Support for Games, NetGames 2008, Worcester, Massachusetts, October 21–22, 2008*. ACM, 2008.

[221] D. V. Sadanand. “Model Checking in General Game Playing: Automated Translation from GDL-II to MCK”. MA thesis. Auckland University of Technology – School of Engineering, Computer and Mathematical Sciences, Aug. 2017.

[222] A. Saffidine. “The Game Description Language is Turing Complete”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 6.4 (Dec. 2014).

[223] K. Salen. “Gaming Literacies: A game Design Study in Action”. In: *Journal of Educational Multimedia and Hypermedia* 16.3 (July 2007).

[224] V. T. Sarinho and A. L. Apolinário. “A Generative Programming Approach for Game Development”. In: *Proceedings of the 8th Brazilian Symposium on Games and Digital Entertainment, SBGAMES 2009, Rio de Janeiro, Brazil, October 8–10, 2009*. IEEE, 2009.

[225] V. T. Sarinho et al. “A Feature-Based Environment for Digital Games”. In: *Entertainment Computing – Proceedings of the 11th International Conference on Entertainment Computing, ICEC 2012, as part of the 2nd Workshop on Game Development and Model-Driven Software Development, GD&MDSD 2012, Bremen, Germany, September 26–29, 2012*. Vol. 7522. LNCS. Springer, 2012.

[226] T. Schaul. “A Video Game Description Language for Model-Based or Interactive Learning”. In: *Proceedings of the 2013 IEEE Conference on Computational Intelligence in Games, CIG 2013, Niagara Falls, ON, Canada, August 11–13, 2013*. IEEE, 2013.

[227] T. Schaul. “An Extensible Description Language for Video Games”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 6.4 (Dec. 2014).

[228] K. Schenk et al. “ScriptEase II: Platform Independent Story Creation Using High-Level Patterns”. In: *Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2013, Boston, Massachusetts, USA, October 14–18, 2013*. AAAI, 2013.

[229] C. Simpkins et al. “Towards Adaptive Programming: Integrating Reinforcement Learning into a Programming Language”. In: *Proceedings of the 23rd ACM SIGPLAN Conference on Object-oriented Programming Systems Languages and Applications, OOPSLA 2008, Nashville, TN, USA, October 19–23, 2008*. ACM, 2008.

[230] C. Simpkins et al. “Towards Adaptive Programming: Integrating Reinforcement Learning into a Programming Language”. In: *SIGPLAN Notices* 43.10 (Oct. 2008).

[231] R. M. Smelik et al. “A Declarative Approach to Procedural Modeling of Virtual Worlds”. In: *Computers & Graphics* 35.2 (2011).

[232] R. Smelik et al. “Integrating Procedural Generation and Manual Editing of Virtual Worlds”. In: *Proceedings of the 1st Workshop on Procedural Content Generation in Games, PCG 2010, Monterey, CA, USA, June 19–21, 2010*. ACM, 2010.

[233] A. M. Smith and M. Mateas. “Variations Forever: Flexibly Generating Rulesets from a Sculptable Design Space of Mini-Games”. In: *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games, CIG 2010, Dublin, Ireland, August 18–21, 2010*. IEEE, 2010.

[234] A. M. Smith and M. Mateas. “Answer Set Programming for Procedural Content Generation: A Design Space Approach”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.3 (Sept. 2011).

[235] A. M. Smith et al. “LUDOCORE: A Logical Game Engine for Modeling Videogames”. In: *Proceedings of the 2010 IEEE Conference on Computational Intelligence and Games, CIG 2010, Dublin, Ireland, August 18–21, 2010*. IEEE, 2010.
[236] A. M. Smith. “The Intelligent Game Designer: Game Design as a New Domain for Automated Discovery”. PhD thesis proposal. UC Santa Cruz, 2009.
[237] A. M. Smith. “Mechanizing Exploratory Game Design”. PhD thesis. University of California, Santa Cruz, 2012.
[238] A. M. Smith and M. Mateas. “Towards Knowledge-Oriented Creativity Support in Game Design”. In: *Proceedings of the 2nd International Conference on Computational Creativity, ICCC 2011, Mexico City, April 27–29, 2011*. Universidad Autónoma Metropolitana, 2011.
[239] A. M. Smith et al. “Computational Support for Play Testing Game Sketches”. In: *Proceedings of the 5th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2009, Stanford, California, USA, October 14–16, 2009*. AAAI, 2009.
[240] A. M. Smith et al. “Prototyping Games with BIPED”. In: *Proceedings of the 5th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2009, Stanford, California, USA, October 14–16, 2009*. AAAI, 2009.
[241] A. M. Smith et al. *An Inclusive Taxonomy of Player Modeling*. Tech. rep. UCSC-SOE-11-13. Santa Cruz, California., 2011.
[242] G. Smith et al. “Rhythm-Based Level Generation for 2D Platformers”. In: *Proceedings of the 4th International Conference on Foundations of Digital Games, FDG 2009, Orlando, Florida, April 26–30, 2009*. ACM, 2009.
[243] G. Smith et al. “Tanagra: A Mixed-Initiative Level Design Tool”. In: *Proceedings of the 5th International Conference on the Foundations of Digital Games, FDG 2010, Monterey, CA, USA, June 19–21, 2010*. ACM, 2010.
[244] G. Smith et al. “Tanagra: An Intelligent Level Design Assistant for 2D Platformers”. In: *Proceedings of the 6th Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2010, Stanford, California, USA, October 11-13, 2010*. AAAI, 2010.
[245] G. Smith et al. “Launchpad: A Rhythm-Based Level Generator for 2-D Platformers”. In: *IEEE Transactions on Computational Intelligence and AI in Games* 3.1 (Mar. 2011).
[246] K. T. Stolee. *Kodu Language and Grammar Specification*. Microsoft Research. 2010.
[247] A. Summerville et al. “From Mechanics to Meaning”. In: *IEEE Transactions on Games* 11.1 (Mar. 2019).
[248] T. Sweeney. “The Next Mainstream Programming Language: a Game Developer’s Perspective”. In: *Conference record of the 33rd ACM SIGPLAN-SIGACT Symposium on Principles of Programming Languages, POPL 2006, Charleston, South Carolina, USA, January 11–13, 2006*. ACM, 2006.
[249] N. Szilas. “BEcool: Towards an Author Friendly Behaviour Engine”. In: *Virtual Storytelling. Using Virtual Reality Technologies for Storytelling – Proceedings of the 4th International Conference, ICVS 2007, Saint-Malo, France, December 5-7, 2007*. Vol. 4871. LNCS. Springer, 2007.
[250] S. Tang and M. Hanneghan. “Towards a Domain Specific Modelling Language for Serious Game Design”. In: *Proceedings of the 6th International Game Design and Technology Workshop and Conference, GDTW 2008, Liverpool, UK, November 12–13, 2008*. Liverpool John Moores University, 2008.
[251] S. Tang and M. Hanneghan. “State-of-the-Art Model Driven Game Development: A Survey of Technological Solutions for Game-Based Learning”. In: *Journal of Interactive Learning Research* 22.4 (Dec. 2011).

[252] S. Tang et al. "A Platform Independent Game Technology Model for Model Driven Serious Games Development". In: *The Electronic Journal of e-Learning* 11.1 (Feb. 2013).

[253] M. Thielscher. "A General Game Description Language for Incomplete Information Games". In: *Proceedings of the Twenty-Fourth AAAI Conference on Artificial Intelligence, AAAI 2010, Atlanta, Georgia, USA, July 11–15, 2010*. AAAI, 2010.

[254] M. Thielscher. "The General Game Playing Description Language is Universal". In: *Proceedings of the 22nd International Joint Conference on Artificial Intelligence, IJCAI 2010, Barcelona, Spain, July 16–22, 2011*. AAAI, 2011.

[255] M. Thielscher. "Translating General Game Descriptions into an Action Language". In: *Logic Programming, Knowledge Representation, and Nonmonotonic Reasoning: Essays Dedicated to Michael Gelfond on the Occasion of His 65th Birthday*. Vol. 6565. LNCS. Springer, 2011.

[256] N. Thillainathan and J. M. Leimeister. "Serious Game Development for Educators - A Serious Game Logic and Structure Modeling Language". In: *EDULEARN14 Proceedings – 6th International Conference on Education and New Learning Technologies, EDULEARN 2014, Barcelona, Spain, July 7–9, 2014*. IATED, 2014.

[257] J. Togelius and J. Schmidhuber. "An Experiment in Automatic Game Design". In: *Proceedings of the 2008 IEEE Symposium On Computational Intelligence and Games, CIG 2008, Perth, WA, Australia, December 15–18, 2008*. 2008.

[258] M. Treanor et al. "Proceduralist Readings: How to find meaning in games with graphical logics". In: *Proceedings of the 6th International Conference on the Foundations of Digital Games, FDG 2011, Bordeaux, France, June 28–July 1, 2011*. ACM, 2011.

[259] M. Treanor et al. "Game-O-Matic: Generating Videogames That Represent Ideas". In: *Proceedings of the 3rd Workshop on Procedural Content Generation in Games, PCG 2012, Raleigh, NC, USA, May 29–June 01, 2012*. ACM, 2012.

[260] M. Treanor et al. "The Micro-Rhetorics of Game-o-Matic". In: *Proceedings of the 7th International Conference on the Foundations of Digital Games, FDG 2012, Raleigh, North Carolina, USA, May 29–June 01, 2012*. ACM, 2012.

[261] T. Tutenel et al. "A Semantic Scene Description Language for Procedural Layout Solving Problems". In: *Proceedings of the 6th Conference on Artificial Intelligence and Interactive Digital Entertainment, AIIDE 2010, Stanford, California, USA, October 11–13, 2010*. 2010.

[262] F. van Broeckhoven and O. de Troyer. "ATTAC-L: A Modeling Language for Educational Virtual Scenarios in the Context of Preventing Cyber Bullying". In: *Proceedings of the IEEE 2nd International Conference on Serious Games and Applications for Health, SeGAH 2013, Algarve, Portugal, May 2–3, 2013*. IEEE, 2013.

[263] F. van Broeckhoven et al. "Using a Controlled Natural Language for Specifying the Narratives of Serious Games". In: *Interactive Storytelling – Proceedings of the 8th International Conference on Interactive Digital Storytelling, ICIDS 2015, Copenhagen, Denmark, November 30–December 4, 2015*. Vol. 9445. LNCS. Springer, 2015.

[264] F. van Broeckhoven et al. "Mapping between Pedagogical Design Strategies and Serious Game Narratives". In: *Proceedings of the 7th International Conference on Games and Virtual Worlds for Serious Applications, VS-GAMES 2015, Skovde, Sweden, September 16–18, 2015*. 2015.

[265] S. van Hoecke et al. "Enabling Control of 3D Visuals, Scenarios and Non-linear Gameplay in Serious Game Development Through Model-Driven Authoring". In: *Serious Games, Interaction, and Simulation – Proceedings of the 5th International Conference, SGAMES 2015, Novedrate, Italy, September 16–18, 2015*. Vol. 161. LNICST. Springer, 2016.

[266] C. van Nimwegen et al. “A Test Case for GameDNA: Conceptualizing a Serious Game to Measure Personality Traits”. In: *Proceedings of the 16th International Conference on Computer Games, CGAMES 2011, Louisville, KY, USA, July 27–30, 2011*. IEEE, 2011.

[267] R. van Rozen. “A Pattern-Based Game Mechanics Design Assistant”. In: *Proceedings of the 10th International Conference on the Foundations of Digital Games, FDG 2015, Pacific Grove, CA, USA, June 22–25, 2015*. Society for the Advancement of the Science of Digital Games, 2015.

[268] R. van Rozen and J. Dormans. “Adapting Game Mechanics with Micro-Machinations”. In: *Proceedings of the 9th International Conference on the Foundations of Digital Games, FDG 2014, Liberty of the Seas, Caribbean, April 3–7, 2014*. Society for the Advancement of the Science of Digital Games, 2014.

[269] R. van Rozen and Q. Heijn. “Measuring Quality of Grammars for Procedural Level Generation”. In: *Proceedings of the 13th International Conference on Foundations of Digital Games, FDG 2018, as part of the 9th Workshop on Procedural Content Generation, PCG 2018, Malmö, Sweden, August 7–10, 2018*. ACM, 2018.

[270] A. Varanese. *Game Scripting Mastery*. Premier Press, 2003.

[271] C. Verbrugge. “A Structure for Modern Computer Narratives”. In: *Proceedings of the 3rd international conference on Computers and Games, CG 2002, Edmonton, Canada, July 25-27, 2002, Revised Papers*. Vol. 2883. LNCS. Springer, 2003.

[272] C. Verbrugge and P. Zhang. “Analyzing Computer Game Narratives”. In: *Entertainment Computing – Proceedings of the 9th International Conference on Entertainment Computing, ICEC 2010, Seoul, Korea, September 8–11, 2010*. Vol. 6243. LNCS. Springer, 2010.

[273] K. Villaverde et al. “Cheshire: Towards an Alice Based Game Development Tool”. In: *Proceedings of the International Conference on Computer Games, Multimedia & Allied Technology, CGAT 2009, Singapore, May 11–12, 2009*. Research Pub. Services, 2009.

[274] R. Walter. “Engineering Domain-Specific Languages for Games”. In: *Game Development Tool Essentials*. Apress, 2014.

[275] 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, ACE 2011, Lisbon, Portugal, November 8–11, 2011*. ACM, 2011.

[276] K. Wang et al. “3D Game Design with Programming Blocks in StarLogo TNG”. In: *Proceedings of the 7th International Conference on Learning Sciences, ICLS 2006, Bloomington, Indiana, June 27–July 01, 2006*. International Society of the Learning Sciences, 2006.

[277] B. H. Wasty et al. “ContextLua: Dynamic Behavioral Variations in Computer Games”. In: *Proceedings of the 2nd International Workshop on Context-Oriented Programming, COP 2010, Maribor, Slovenia, June 22, 2010*. ACM, 2010.

[278] M. West. “Domain-Specific Languages”. In: *Game Developer* 14.7 (Aug. 2007).

[279] R. Wetzel. “A Case for Design Patterns Supporting the Development of Mobile Mixed Reality Games”. In: *Workshop Proceedings of the 8th International Conference on the Foundations of Digital Games – as part of the 2nd Workshop on Design Patterns in Games, DPG 2013, Chania, Crete, Greece, May, 14–17, 2013*. Society for the Advancement of the Science of Digital Games, 2013.

[280] R. Wetzel. “Introducing Pattern Cards for Mixed Reality Game Design”. In: *Proceedings of Workshops Colocated with the 9th International Conference on the Foundations of Digital Games – as part of the 3rd Workshop on Design Patterns in Games, DPG 2014, Liberty of the Seas, Caribbean, April 3–7, 2014*. Society for the Advancement of the Science of Digital Games, 2014.

[281] W. White et al. “Scaling Games to Epic Proportions”. In: *Proceedings of the 2007 ACM SIGMOD International Conference on Management of Data, SIGMOD 2007, Beijing, China, June 11–14, 2007*. ACM, 2007.
[282] W. White et al. “Better Scripts, Better Games”. In: *Queue* 6.7 (Nov. 2008).
[283] W. White et al. “Declarative Processing for Computer Games”. In: *Proceedings of the 2008 ACM SIGGRAPH Symposium on Video Games, Sandbox 2008, Los Angeles, California, USA, August 9–10, 2008*. ACM, 2008.
[284] W. White et al. “Better Scripts, Better Games”. In: *Communications of the ACM* 52.3 (Mar. 2009).
[285] G. J. Winters and J. Zhu. “Attention Guiding Principles in 3D Adventure Games”. In: *SIGGRAPH 2013 Posters*. ACM, 2013.
[286] G. J. Winters and J. Zhu. “Guiding Players through Structural Composition Patterns in 3D Adventure Games”. In: *Proceedings of the 9th International Conference on the Foundations of Digital Games, FDG 2014, Liberty of the Seas, Caribbean, April 3–7, 2014*. Society for the Advancement of the Science of Digital Games, 2014.
[287] J. P. Zagal. *Ludoliteracy: Defining Understanding and Supporting Games Education*. ETC Press, 2010.
[288] J. P. Zagal et al. “Towards an Ontological Language for Game Analysis”. In: *Proceedings of the 2005 DiGRA International Conference: Changing Views: Worlds in Play, DiGRA 2005, Vancouver, Canada, June 16–20, 2005*. Digital Games Research Association, 2005.
[289] J. P. Zagal et al. “Towards an Ontological Language for Game Analysis”. In: *Worlds in Play, International Perspectives on Digital Games Research*. Peter Lang, 2007.
[290] J. P. Zagal et al. “Dark Patterns in the Design of Games”. In: *Proceedings of the 8th International Conference on the Foundations of Digital Games, FDG 2013, Chania, Crete, Greece, May 14–17, 2013*. Society for the Advancement of the Science of Digital Games, 2013.
[291] M. Zhu. “Model-Driven Game Development Addressing Architectural Diversity and Game Engine-Integration”. PhD thesis. Norwegian University of Science et al., Mar. 2014.
[293] A. Zook and M. O. Riedl. “Automatic Game Design via Mechanic Generation”. In: *Proceedings of the 28th AAAI Conference on Artificial Intelligence, AAAI 2014, Québec City, Canada, July 27–31, 2014*. AAAI, 2014.
[294] A. Zook and M. O. Riedl. “Generating and Adapting Game Mechanics”. In: *Proceedings of the 5th Workshop on Procedural Content Generation, PCG 2014, Liberty of the Seas, Caribbean, April 3–7, 2014*. Society for the Advancement of the Science of Digital Games, 2014.