CICERO - AN AI-ASSISTED GAME DESIGN SYSTEM

DISSERTATION

Submitted in Partial Fulfillment of

the Requirements for

the Degree of

DOCTOR OF PHILOSOPHY (Computer Science)

at the

NEW YORK UNIVERSITY
TANDON SCHOOL OF ENGINEERING

by

Tiago Lemos de Araujo Machado

January 2020

CICERO - AN AI-ASSISTED GAME DESIGN SYSTEM

DISSERTATION

Submitted in Partial Fulfillment of

the Requirements for

the Degree of

DOCTOR OF PHILOSOPHY (Computer Science)

at the

NEW YORK UNIVERSITY
TANDON SCHOOL OF ENGINEERING

by

Tiago Lemos de Araujo Machado

January 2020

Approved:

_____
Department Chair Signature

_____
Date

University ID: N13740892
Net ID: tm2491

Approved by the Guidance Committee:

Major: Computer Science

Julian Togelius
Associate Professor
New York University, Tandon School of Engineering

Date

Andy Nealen
Associate Professor
University of Southern California, School of Cinematic Arts

Date

Oded Nov
Associate Professor
New York University, Tandon School of Engineering

Date

Alessandro Canossa
Associate Professor
Royal Danish Academy of Fine Arts, School of Design

Date

Microfilm/Publishing

Microfilm or copies of this dissertation may be obtained from:

UMI Dissertation Publishing
ProQuest CSA
789 E. Eisenhower Parkway
P.O. Box 1346
Ann Arbor, MI 48106-1346

Vita

Tiago Lemos de Araujo Machado

Education

PhD in Computer Science Sep. 2015 - Sep. 2019
New York University, Tandon School of Engineering, Game Innovation Lab

M.S. in Computer Science Mar. 2011 - Sep. 2013
Universidade Federal de Pernambuco, CIn - UFPE
- *CAPES Scholarship*

B.S. in Computer Science Sep. 2005 - Dec. 2010
Universidade Federal de Pernambuco, CIn - UFPE

Research and Funding

This research was performed at the NYU Game Innovation Lab. Funding for tuition and personal expenses was granted by the CNPq Science without Borders scholarship. Funding for traveling to present the papers that originated this work and to attend conferences was provided by Prof. Julian Togelius, Prof. Andy Nealen and the Game Innovation Lab.

Professional Experience

Tiago has worked as a STEM consultant at C.E.S.A.R (Recife Center for Advanced Studies and Systems), under the program NAVE. During five years Tiago, alngside a team of engineers and designers, was responsible for bringing technology to the Brazilian High School Curriculum. He Designed games and education softwares, published research papers and participated in national and international competitions. Tiago assumed the principal technical coordination position of the program in 2014, one year before joining New York University.

Acknowledgments

There are many people I would like to thank, here is a short list (I'm sorry for the ones I may forget). Firstly, I thank God for this blessed opportunity. Then, I would like to thank my advisor, Prof. Julian Togelius, the main responsible for this unforgettable journey. Secondly, my committee: Prof. Andy Nealen, Prof. Oded Nov, and Prof. Alessandro Canossa. They had contributed with their expertise in different domains to improve the quality of this work. My collaborators: Daniel Gopstein, Angela Wang, Katalina Park, Katherine LoScalzo, and Zhongheng Li, for fruitful discussion and helping me on putting my thoughts in place. My lab colleagues at the NYU Game Innovation Lab: Ahmed Khalifa, Michael Green, Ivan Bravi, Amy Hoover, Ming Jin, Gabriella Barros, Andre Mendes, Philip Bontrager, Chris DiMauro, Aaron Isaksen, Christoffer Holmgård, Christoph Salge, Luvneesh Mugrai, Scott Lee, Ramsey Nasser, Rodrigo Canaan, Bruna Oliveira, Fernando de Messentier (who I thank again for share the template of his PhD thesis), Erica Ribeiro, Henry Zhou, Christopher Michaels, Christopher Jin, Benedikte Mikkelsen, and of course, Kaho Abe and Aditya Bhat for sharing their love about movies, and Kurosawa's works. It was a pleasure play with you all. My AI for Games Summer school team: Debosmita Bhaumik, Taejong Choi, and Catalina Jaramillo (also partners during many lunches in the Game Innovation Lab). My friends from the New York Public Libraries: Peggy Yannas, Kenneth Fields, Margery Druss, Diane Quinton, Anil Pamnani, Carolyn Hale, Danielle Haub-Richmond, Gail Luria, Megan Cogswell, Daniel Rockower, Khabari Philips, Ping Zheng, Mariana Batista, Cecilio Ferreira, Victoria Ferreira Batista, Yifan Yu, Asya Wadud, Laura Dei Cas, Tamires Gonçalves, James Wu, Natalia Campos, Dayane Rocha, and Minwha Roh. My Tango Friends: Mariana Fresno, Yichun Liu, and Thomas Laetsch. My food trip friends: Zhu Wang and Yifan Hu for all the good time and support. Katy Yau for listening during hard times. Burcu Kaniskan, for helping me in everything that she could. Vincent & Vivian Cama, and Joseph Cama for all the dinners, parties, movie nights, and games. NYU Staff: Chrystanyaa Brown, Kari Schwartz, Eve Henderson, Judy Brown, Ann Boray, Kyoko Nakamura-Tirado, Dr. Wanda Ramos, Helen Leonard, Johanna Pan-Carr, Prof. Lisa Hellerstein, Prof. Ray Lutzky (now at Cornell University), Prof. Jose Ulerio, and Officer Jones. My friends at NYU Tandon School of Engineering: Jorge Henrique Ono (also my roomate in Brooklyn), Raoni Lourenço, Deniz Vurmaz (and everyone from Digistrips), Danielli Bertolli. My former work colleagues, Luiz Araujo, Anderson Paulo, Danielle Cristina, Felipe Borgianni, Mauricio Taumaturgo, Eduardo Olivreira, Felipe Furtado e Marcela Cox. The Immersive Health Group team: Dov Hirsch, Jaye Moore, Corneel Booysen, and Jake Maymar. To my first advidsors, for endless support and inspiration: Prof. Geber Ramalho, Prof. Alex Gomes, Silvia Tigre, Prof. Carina

Frota and Prof. Marcelo Walter. My conference friends: Julian Hernandez, Levi Lelis and Alberto Alvarez. And finally, my friends from the cities of Gravatá, Vitória de Santo Antão and Recife (all in Pernambuco state, Brazil).

Liberdade, palavra que o sonho humano alimenta, não há ninguém que explique e ninguém que não entenda. — Cecília Meireles

Dedication

I dedicate this work to my family - Julia, Adeilton, Diana, and Alysson.

ABSTRACT

CICERO - AN AI-ASSISTED GAME DESIGN SYSTEM

by

Tiago Lemos de Araujo Machado

Advisor: Julian Togelius

Submitted in partial fulfillment of the Requirements for the Degree of Doctor of Philosophy (Computer Science)

January 2020

Game Design is a complex task. The amount of technology surrounding the development process is constantly increasing. It harms the productivity of professionals and creates a steep learning curve for those who want to design games. A myriad of tools is available for the task of creating games, both as a research and industrial products. However many of these tools require a lot of effort from the users who spend more time learning and configuring a tool than on their design. AI-game design assistants have been developed to address many of the challenges that circa the game development process, like memory management and content creation. However, since their rise, and despite their initial results, they also present some flaws. Most of them are straight-attached to one game only, and if designers want to use the AI-assistant they had for another game, they will have to implement everything from scratch. The other issue is the lack of formal user studies. It is still not clear how these tools can bring the best out of the AI-powered human performance. In this thesis, we present Cicero, an AI-game design assisted tool that offers assistance in the form of visualization, game-rule usage statistics, replay analysis system, game events databases, AI-debugging, and recommender systems. Cicero is built upon the General Video Game AI framework and the Video Game Description Language, which means that its methods can be applied to different games inside the availability provided by the framework it relies on. Cicero's features were evaluated formally through user studies based on quantitative analysis. Cicero's design and evaluation will be discussed in detail throughout this thesis, the conclusion of these findings and applications that expanded it to other fields outside the game realm will be presented as well.

Table of Contents

List of Figures

Chapter 1

Introduction

1.1 Motivation

Game Development is hard. The engineering applied to craft a game project is an object of study. Since the rise of the game industry, researchers and practitioners have been publishing their findings of the complexities of building, managing, and evolving a game [13]. This complexity has only increased with new technologies and methods that have arisen due to new developments in fields like graphics, physics, networking, social media, and many others. Since then, many other publications have been discussing the continually evolving complexities of a game engineering process, from different perspectives like Software Engineering [108, 133] and Product Management [109]. At the same time, the game industry has become one of the most diverse industries in the world regarding segments based on disparate platforms (mobile, consoles, web, etc.), diverse customer profiles (children, teenagers, adults, the elderly, etc.) and even different goals (social and health games for example).

This constant rising complexity (inside and out the production environments) calls for more capable tools, and consequently, there is often increments and new advancements of game development toolchains. However, there are those who argue that game development tools could be so much more, given available technology [69]. To address the increasing complexity outlined by [13, 108, 133, 69] and the growing consumer demand, we have seen the rise of mixed-initiative tools, a hybrid in which humans and artificial intelligence (AI) work together to solve specific problems. Such an approach was embraced by game researchers and industry professionals alike. Nowadays, game designers have tools to assist them in a myriad of tasks, from level generation to game balancing. For the systems that offer AI assistance, the term AI-assisted Game Design Tools was coined. Despite their initial results, these tools also present challenges. One of the most prevalent problems is

the lack of generalizability. Many of the techniques presented are directly attached to a specific game. It then becomes complicated when a developer wants to apply the same techniques to another project without having to re-implement everything again from scratch. Another issue is the lack of empirical justification for these approaches. There is a dearth of literature on the human factors of mixed-initiative systems designed for game development tasks. While we can say that many game titles available nowadays were possible to be made only because mixed-initiative approaches, it is also true that we do not understand how we can extract the full potential of this man-machine combination. The reactions of the human activity in the face of an intense Algorithm Experience, as pointed by [100] is still an open field to explore. It is even more urgent in an area where the search for increasing content and creativity outcomes needs to keep the same pace of increasing demand and diversified customer segmentation.

With these challenges in mind, this thesis presents Cicero, an AI-assisted Game Design Tool that allows designers to prototype and simulate their games by using intelligent agents together with a diversified set of computational methods. Cicero offers heat-map visualization, replay analysis, game-event queries, AI-debugging assistance, and a recommender system for game mechanics. Cicero is built on top of the General Video Game AI Framework (GVGAI) and offers more generality than systems built for stand-alone games. Users can apply all of Cicero's features to many different genres within the grid physics-based-games sphere. To highlight the immediate advantage these methods offer to game designers, we conducted formal user studies. In total, these studies sum up to 119 participants. In all of them we evaluated how the users' performance was affected by having or not the presence of AI assistance. During this thesis, we detail all of the background and ideas from the game and AI communities that inspired us to do this work. The main details of Cicero's implementations and all of these features will be explained in the upcoming chapters as well. In particular, we designed this system to evaluate how AI can empower humans, especially novice game designers to be more creativity more often. To answer this question we first applied a user study that verified how humans performance in game debugging tasks is modified by having or not the assistance of an algorithm. Then we designed a second study in which we verified how the participant's levels of workload, affect, and self-efficacy changes when they design a game with and without AI assistance.

1.2 Thesis Statement

This work focuses on how AI-game design assistant tools can be designed and proved to be beneficial to game design tasks. We based this dissertation according to hypotheses that

were driven by the Thesis Statement below:

> ***We can use tools based on artificial intelligence to design systems able to help humans in game designing tasks.***

1.3 Hypotheses

We formulated the following hypotheses to support our Thesis Statement.

- **H1 - Workload** - The presence of AI reduces the users self-perception of workload.
- **H2 - Accuracy** - The presence of AI increases the users' accuracy.
- **H3 - Computer Affect** - The presence of AI increases users levels of computer affect.
- **H4 - Self-Efficacy** - The presence of AI increases users levels of computer self-efficacy.

1.4 Outcomes of interest

In this section we discuss why the selected metrics (showed in the previous subsection) are important for this research and design of AI-assisted game design systems in general.

- **Workload** - We want the users to be as focused as possible on their game design tasks. Game editors UIs are very similar from Computer Aided Design (CAD) UI tools. These tools have many critics related to their steep learning curve and the amount of effort one needs to apply to have tasks accomplished. Therefore, the goal is to minimize workload by using AI-assistants.
- **Accuracy** - In many chapters of this thesis we will mention works that were designed in a mixed-initiative way. Such works contain contents designed by humans and algorithms. Therefore, our goal is to raise the accuracy level of the users when working with AI-assistants for game design.
- **Computer Affect** - Previous research indicates that affective states of users can be exploited to increase the effectiveness of human-computer interaction experiences [29]. Therefore, we are interested in high levels of computational affect to raise the efficiency of the users when working with AI-assisted game tools.

- **Self-efficacy** - High self-efficacy means high confidence of one's ability to perform a task. We are interested in high levels of self-efficacy to fostering the raise of confidence in users when designing games with AI-assistants.

1.5 Contributions

This work consists of computational methods that can support users on game design tasks that can be performed in different games and game genres. We used a combination of visualization, databases, intelligent agents, and data mining. These are the foundations of an AI-assisted game design tool called Cicero. To evaluate the effectiveness of our methods, we designed user studies in which our tool was formally assessed following a quantitative analysis approach.

- **Cicero:** Stands for Computationally Intelligent Collaborative EnviROnment for the game and level design. It is the central system described in this work. It is based on the General Video Game Framework (GVGAI) [105, 107]. We decided on GVGAI, because it allowed us to have the freedom of applying our methods to a set of different game genres. We recognize that GVGAI has its limitations, but at the time we started our project, it was, and it remains to be the Framework with the best level of generalization thus far.

- **Game Rule Statistics:** The game rule statistics shows how human players and agents explore the rule space of a game. Designers can optimize their game design by reducing rules never explored, and also "force" the game to lead players to fire events that increase or reduces the uses of specific game's rules.

- **Playtrace Visualization:** Cicero brings a visualization tool that prints aggregate playtraces from Human and Agents players. The visualization works like a heat-map system and as more explored is an area, darker it will be its color on the level map.

- **SeekWhence:** SeekWhence is a replay analysis system. It stores and retrieves entire gameplay sessions, whether played by agents or humans. Seekwhence allows game and agent designers to inspect an entire gameplay session or chunks of it. They can use it as a debug tool or as an instrument to evaluate and tune a level. We applied the use of the visualization system to enhance SeekWhence capabilities.

- **Kwiri:** Kwiri builds on SeekWhence. It uses SeekWhence's storing system to retrieve information about game events from previous gameplay sessions. Designers can

then use it to query for specific questions they may have about a gameplay session such as "is the hidden treasure ever found?" or "how many bats were killed by the player/agent?".

- **AI-Debugging assistant** The debugging assistant is designed upon Kwiri and Seek-Whence, and it uses both for gathering data from gameplay sessions and provide interfaces for the users to navigate through the recorded session in search for the questions, bugs, and inconsistencies a game may have. the debugging system makes intense use of the agents available in the GVGAI framework, and they help the users in not rely on human or themselves to play the games for a significant amount of time with multiple test scenarios.

- **Pitako:** Pitako is a recommender system for game mechanics. It is based on frequent itemset data mining and builds association rules by exploring the games available in the GVGAI framework. It provides levels of confidence for the rules and tries to match them when users are developing new games. Then it is up to the users to decide which recommendations they will use in their games.

- **Evaluation of Pitako:** We designed a user study to evaluate the effectiveness of Pitako. This user study was based on quantitative analysis and had the participation of 87 individuals divided into two groups for indicating how the game design task changes for people with a recommender system and people without it.

1.6 Outline

The outline of this dissertation is as follows:

Chapter 2: Background Discusses the main influences and motivations for this such as games and AI, computer-aided design, mixed-initiative methods, recommender systems, and research design for quantitative user studies.

Chapter 3: Methods Discusses the main methods used in this thesis. The chapter is divided in two sections, one discusses the methods necessary to implement the system(s), while the other discusses the materials we used for evaluate its effectiveness.

Chapter 4: Cicero Introduces the Cicero system and the main details about its design and implementation. The basic features of the system will also be explained as well as a user study about the development and evaluation of an automatic AI-based game debugging assistant system designed within Cicero.

Chapter 5: Seekwhence Describes what are replay analysis system and their role in scientific visualization. This chapter introduces the design and implementation of SeekWhence, a replay analysis system for game design, as examples of its use cases as well.

Chapter 6: Kwiri Explains the movtivation and design of Kwiri. Use cases about how to use a retrieval information system will be presented, as well as cases in which it is enhanced by the use of othet Cicero's features like visualization heat-maps and SeekWhence.

Chapter 8: AI-assisted game debugging with Cicero This chapter details how we combined Cicero main features to design an AI-debugging assistant. We explain in which scenarios it is usefull, for what sorts of games, and how humans perform in game debugging tasks with and without an AI-assistant tool.

Chapter 8: Pitako Introduces the design and implementation of our first recommender system for game mechanics.

Chapter 9: Pitako 2 Introduces the motivation, designs and implementantion of our second recommender system. A brief explanation of what went wrong with our first attempt, as well as new references and methods that inspired us to push it forward are also described.

Chapter 10: Pitako's evaluation Describes an experiment in which we tested Pitako with 87 participants. They were divided into two groups (A and B) and had to design a game with the system (group A) and without the system (group B). The experiment evaluates how participant's self-perceptions of workload, affect, and self-efficacy changes in the presence of a recommender system.

Chapter 11: Conclusion and Future Work Gives an overview of the entire dissertation and discuss the contributions of this work. Finally, it points to suggestions for improvements that can be developed as potential future works.

1.7 Publications

The work in this dissertation originates from several previously published papers, and one currently under review. Most of the work presented here originates from the following:

Tiago Machado, Ivan Bravi, Zhu Wang, Andy Nealen, Julian Togelius. Shopping For Game Mechanics. In Procedural Content Generation Workshop, 2016 [81].

Tiago Machado, Andy Nealen, Julian Togelius. CICERO: Computationally Intelligent Collaborative Environment for game and level design. In Computational Creativity and Games Workshop at the 8th International Conference on Computational Creativity, 2017 [80].

Tiago Machado, Andy Nealen, Julian Togelius. Cicero - A mixed-initiative AI-assisted game design tool. In ACM Foundations of Digital Games, 2017 [86].

Tiago Machado, Daniel Gopstein, Oded Nov, Andy Nealen, Julian Togelius. AI-Assisted Game Debugging with Cicero. In IEEE Congress on Evolutionary Computation , 2018 [83].

Tiago Machado, Daniel Gopstein, Andy Nealen, and Julian Togelius. Kwiri - What, When, Where and Who: Everything You Ever Wanted to Know About Your Game But Didn't Know How to Ask. In Proceedings of the 2nd Workshop on Knowledge Extraction from Games co-located with 33rd AAAI Conference on Artificial Intelligence (AAAI 2019) [84].

Tiago Machado, Daniel Gopstein, Andy Nealen, and Julian Togelius. Pitako - Recommending Game Design Elements in Cicero. In IEEE Conference on Games 2019 (COG2019) [82].

Tiago Machado, Daniel Gopstein, Angela Wang, Oded Nov, Andy Nealen, and Julian Togelius. Evaluation of a Recommender System for Assisting Novice Game Designers. In Artificial Intelligence and Interactive Digital Entertainment 2019 (AIIDE'19) [85]

Although not present in this dissertation, I have also contributed, to the following publications inside the fields of Artificial Intelligence and Games, and Artificial Intelligence and Education:

Michael Cerny Green, Ahmed Khalifa, Gabriella A. B. Barros, Tiago Machado, Andy Nealen, Julian Togelius. AtDELFI: automatically designing legible, full instructions for games. In ACM Foundations of Digital Games, 2018 [52].

Eric Eaton, Sven Koenig, Claudia Schulz, Francesco Maurelli, John S. Y. Lee, Joshua Eckroth, Mark Crowley, Richard G. Freedman, Rogelio Enrique Cardona-Rivera, Tiago Machado, Tom Williams. Blue sky ideas in artificial intelligence education from the EAAI 2017 new and future AI educator program. in *AI Matters*, 2018 [41].

1.8 Notes on Pronoun

The work presented in this dissertation could not be achieved without the collaboration of my co-authors. Although I'm the first author in all the work discussed here, throughout the rest of this work the pronoun "we" will be used in favor of "I", to refer to the effort of all researchers involved.

When referring to a player, this dissertation chooses to use the pronouns "they" and "their" as to respect a gender inclusive speech.

Note that you will find the words "Agent" and "Human" often in the text. Whenever you see "Agent", it refers to algorithms we use to play games automatically. When you see "Human", we are talking about a human being playing the game or using a software. Do not make confusion with "Human Player Agent", which is an "Agent" that mimics an "Human" playing a game.

Chapter 2

Background

In this chapter, we present the previous related work. Context is given for the relation between the topic of this dissertation and the described work. The main point of discussion is how our work is motivated and how it diverges from the various approaches and concepts that have been previously explored.

2.1 Games and AI

Games and AI have a long history together, and it is not a wrong assumption to say that their set of records and groundbreaking results are intertwined. Games, like chess, were (and still are) used to show how powerful an AI can be [61, 23].

However, the cited example is more related to games for AI, this thesis is more related to AI for games. There is a myriad of uses for AI in games, like game balancing [65], realistic non-player character behavior [91], procedural content generation [129], and modeling and predicting player behavior [24].

While these examples are more close to work whose focus is the gameplay experience, our work is located one step before. We are using AI to help designers when they are designing their games. Instead of focusing on the player experience, we are focusing on the game designer experience.

2.2 Game Visualization

Game telemetry and analytics tools based on it, especially visualization, is still in its infancy [16]. Nevertheless, developers can have a more accurate analysis of the (in-game) behavior of players who play their products [77, 30], they can track a myriad of events

(deaths, enemies attacked, items used, etc.) and even know when a player is about to abandon their games [87]. A good review about the subject can be found in [44].

As games tend to diversify from one to another, most of the tools are created within a specific project, for instance, *Data Cracker* is a tool designed to gather and analyze data from one of the titles in the *Dead Space* franchise [90]. *Cure Runners*, a 2D runner game, also had a telemetry and visualization system implemented just for it. In [131], the authors used the game as a case study about how to integrate game analytics tools into the development cycle. G-Player [26] contains a UI that offers the users a myriad of features they can apply to filter the amount of visualization information for analyzing player behavior in games like *Rainbow Six* and *League of Legends*. Finally, the game *Super Mario Bros 3* was used by [125] to analyze gameplay through input controllers.

2.3 Game Playing Agents

Game playing agents are algorithms implemented for playing a game. Usually their common goal is to win the game. And their success metrics are based on computational resources consumed like time and memory.

However, even with the success of AlphaGo [132, 120] and AlphaStar [7], which were proven to beat games once considered impossible to be played at human level by an intelligent machine, many other goals are object of study. One of them is the persona simulation, i.e agents that can mimic personas human players have when playing a game [59].

In this thesis, we made extensive use of game playing agents, they were part of all the tests we have done. Without them it would not be possible to have the systems that we are about to describe in later chapters in a practical time. Since we did not have to rely on human players to generate data from our system, the use of game playing agents was more than suitable for our needs.

2.4 Mixed-initiative methods

Mixed-Initiative User Interfaces have been well discussed [60, 6, 19] and many fields have shown the benefits of the paradigm. It is not a new term in the game community either. However, it has only come to be embraced recently. Among the sub-fields of game design where mixed-initiative methods appear most promising, are for helping developers overcome the complexities and, even more, the scale of game design and meet the increasing demand of the game industry [79]. Filling hundreds of square kilometers with detailed and granular

content in a manual manner is insanely expensive and time consuming for human workers. Therefore, mixed initiatives and PCG is used to fill the content at large scale.

One of the examples of Mixed-Initiative applications in games is Tanagra [126], a tool that assists humans in designing levels for 2D platform games. The system works in real time, creating many different guaranteed-playable levels for the designer to choose from.

Similarly, Ropossum [116], also generates and solves levels for the popular physics puzzle game Cut The Rope. The user is assisted in the tasks of level design and evaluation. The tool is optimized to allow real-time feedback from a given state after receiving a sequence of user inputs. It generates the possible actions for the player until it finds a solution, if available.

Sentient Sketchbook [74] offers more generality than the two works already cited in this section, and according to its authors, it also fosters human creativity [135]. It is also a tool to assist in the creation of game levels. It provides this assistance on strategy and roguelike games. The system shows level suggestions in real-time. It allows users to interact by editing their levels while generating recommendations based on previous choices.

For the game *Refraction*, Smith et al. [122] and Butler et al. [22] present independent implementations of three diverse level design automation tools. They use Procedural Content Generation (PCG) techniques and Answer Set Programming (ASP) to explore the intended design space and offer levels with playability guarantee.

Aside from level generation, Isaksen et al. [64] presented an evolutionary algorithm to find variants for the popular mobile game *Flappy Bird*. The discovered game variants showed themselves to be significantly different from the original game regarding the challenge, game feel, and theme. Still aside from level generation, we have BIPED [124], which is focused on giving insights to designers in order to help them generate games. It offers assistance in the early stages of the game design process. Using BIPED, designers can leverage simple ideas and end up with a playable prototype and a formal rule system that they can test as if they were sketching their ideas for quick evaluation [123].

Mixed-Initiative systems have been applied to table-top games as well. The work of Osborn et al. [103] presents a Game Description Language called *Gamelan* designed to work with board games and card games. Gamelan has a focus on automated game design support, the language contains features for analyzing games based on the computational critics framework. It provides feedback for game elements and events that the designers can use based on their own perspective. Continuing with systems for automated game design, we have the work of Zook and Reidl [139]. It generates and combines mechanics in a high-level concept independent of game genre concerns. It offers checkers for assuring that playability is related to the definition of the generated mechanics.

Nelson and Mateas [95] explored the human factors of Mixed-Initiative systems. They presented a series of interviews with professional game developers to understand what were the most desirable features for the next generation of AI-game design assistants. Finally, Nelson also suggests the game itself as a source of metrics. His work presents strategies about how to extract valuable information from the relations between game elements, such as a rule set before any playtest session [94].

All of the work discussed so far have contributed significant results in the realm of AI-assisted game design tools. However, each is very attached to a single game or genre. There is a lack of generality in their techniques in that they need to be reimplemented every time someone starts a new game project.

2.5 System evaluation

The tools presented in the previous session have a high impact on the game research community and game industry. Actually, some games produced during the rise of AI-driven game design assistants could not have been created without Procedural Content Generation methods, such as Galactic Arms Race [58] and Borderlands for example. There is no silver bullet in game or software development. At the same time as a new technology solves old complexities, new complexities may appear [47].

AI-driven game design assistants are no exception to this rule, making it important that such tools are rigorously evaluated. Previously, the debugging functionality of the Cicero system was evaluated using a quantitative approach [83]. That study compared how humans perform when debugging a game with AI assistance and without it, and found that the AI assistance system offered significant benefit.

More details about this study will be presented later, as well as the details about the evaluation of the recommender system. Unfortunately, these studies are not a rule when dealing with the design of mixed-initiative tools for game development. To add to the list of publications that brings the design of a tool that offer help to designers, alongside with a user evaluation, we can cite the work of Yannakakis et al [136], in which the authors use the feedback of the users to evaluate their level content generation suggestions, and the paper from Guzdial et al. [54], in which authors describe the design of an AI-assisted tool for a level generation as well, as many others cited before in this thesis, but brings a study in which one hundred participants evaluated the system and reported how the experience changed their levels of affect.

The theme of the 15th AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE 19) is “Human Centered Evaluation" what reflects the need of

investigating how to use these AI-based tools to boost human performance in game design related tasks. A theme that in the course of this work seemed to be not explored by the in depth by the AI for Games community.

2.6 Recommender systems

A current popular definition of “recommender system" is “any system that produces individualized recommendations as output or has the effect of guiding the user in a personalized way to interesting or useful objects in a large space of possible options. Such systems have an obvious appeal in an environment where the amount of online information vastly outstrips any individual capability to survey it." [21].

In order to attain the goal of providing accurate recommendations, a myriad of techniques is available[3], from ‘authoritativeness" criteria [18] to collaborative filtering, what is probably the most popular and mature recommendation technique in use nowadays [12, 115], and largely used by companies such as Amazon [76] and Netflix [50]. Collaborative filtering creates personalized recommendations by combining the knowledge of similar users in the system. These techniques present strengths and weakness and its common to hybridize them to overcome the issues. In our case, we decided to use association rule mining for recommender systems [75], more specifically the apriori algorithm [4]. Which we will explore more in the upcoming chapters.

2.7 Suggestion engines for design assistance

While recommender systems are predominantly used in the e-commerce sector, there are examples in the literature of other sorts of suggesting engines. They do not necessarily rely entirely on recommender systems techniques but are an inspiration to our work because they share the idea of searching for patterns and suggesting them to assist design tasks.

In the field of sketch-based interfaces, Igarashi et al. [63] introduced a system that assists humans in 3D modeling works. When users sketch an object, the 3D structure is compared against a database of previously sketched structures. When there is a match between the structures, the parts yet to be modeled are suggested by the system, and the user may choose to utilize one.

A paper published by O’Donovan et al. [99] describes the design of a virtual illustrator assistant. In this work, users can design layouts for business cards or birthday invitations. At the same time, the system is generating altered copies of the users’ original work. These copies change spacing, text font, add and remove figures, backgrounds, resolution, and so

on. This way, the authors claim that the users benefit from an automatically exploratory design task, which they promote as a vital part of the design process.

In the work by Nguyen et al. [96] the authors bring a recommender system for topic conversation suggestions. The idea is to provide a way of two strangers to engage in a good conversation easily. The system takes into consideration a list of subjects that the users have in common and provide topics within them. The results contributed to the understanding of how communication interventions influence people's experience and behaviors, and enhance interpersonal interactions.

CACHECA [46] is a tool designed for providing source code completion suggestions. Different from conventional approaches, CACHECA uses a statistical approach by learning from the codebase of an individual corporation. The prototype was developed as an Eclipse [43] plugin, and showed that its usage in combination with the Eclipse native code suggesting tool more than doubles Eclipse's recommendation accuracy.

DeepSurv [72] is a deep neural network for modeling interactions between a patient's covariates and treatment effectiveness in order to provide personalized treatment recommendations. The ultimate goal of the tool is personalized medicine, the capability of providing health assistance entirely suited to the needs of specific patients. The authors evaluated the effectiveness of DeepSurv by running two experiments in which they compared the risk analysis performance and the accuracy of treatments recommended. In conclusion, the results demonstrated that the use of deep learning in survival analysis allows for: (i) higher performance due to the flexibility of the model, and (ii) effective treatment recommendations based on the predicted effect of treatment options on an individual's risk.

Finally, the (already cited) Sentient Sketchbook brings a mixed-initiative method for assisting the level design of real-time strategy games by providing suggestions based on the level being designed by the user.

These works show that it is possible to use recommender system techniques to assist human tasks. However, it seems that its usage in the game domain just started to scratch the surface. From the perspective of AI game design assistants, the challenge of building tools that can recognize patterns and provide solutions that enhance the work of developers is also a motivation that we think should be better explored.

2.8 Industrial tools to assist game designers

The game development process of many game development companies makes use of AI algorithms. Most of the techniques in use for years became available in the package distribution of the game engines, platforms that encapsulate a set of features necessary

to game productions like graphics, physics simulation, user interface components, audio managers, etc. In this subsection, we list some of the most used tools nowadays and what they offer in terms of AI assistance.

- **Unreal Engine 4** - It offers AI techniques like Behavior Trees, Navigation Mesh, and Environment Query System. The behavior trees allow developers to describe actions that should be performed by Non-Player Characters (NPC) in a regular fashion. Navigation mesh is a map that shows which areas are accessible by NPCs. It allows the NPCs to walk around the level and avoid areas that could be harmful like water lakes, lava, and poisoned forests. Finally, the environmental query system allows the developer to input queries inside the behavior tree, thus the NPC can look for an item at the area it is spawned while events that trigger its behavior tree are not activated.

- **Client Bot** - The tool developed by Ubisoft Reflections offers a set of methods to test a game in the context of “The Division". The system, named as "Client Bots," has an AI that mimics human input while also reporting issues like incomplete missions and performance statistics. Client Bot still includes an automated mission play through, report generation, and a tool to assist bug reproduction.

- **No Man’s Sky’s tools** - “No Man’s Sky" called attention a few years ago for its game production process. The designers and developers came up with a set of new tools to generate the game elements in real-time. The game mechanics is about space exploration, and everything is generated when the player is playing. Some of the tools created by the developers’ team took plants’ skeleton and sketches designed by humans and applied algorithms to diversify the samples and generate the vegetation of a new planet to be explored. It is just one of the examples, but No Man’s Sky had many tools like that designed during its production.

- **Unity ML Agents** - Unity ML Agents is in an advanced stage of tests and has many distributions available. It is Unity approach for applying Machine Learning into games. The company already have tools like analytics and visualizations, but ML agents is bringing AI to the center of the game design process. The tool, at this point, offers a brain interface in which you can define the type of your agent, the observable space and other features necessary to train it. The current distribution presents samples of reinforcement learning agents, multi-agent environments, imitation learning, and many others. These can be ready to be used in prototypes by applying their scripts and brain interfaces to the unity game objects. It might have a good power of replication since

some of the samples show both 3D and 2D games, but the way they will generalize we are still about to know.

2.9 Industrial tools of general assistance

In the last section, we listed some of the AI tools available for assisting game designers. We tried to avoid repetition, so we did not include further presentations of Havok or CryEngine because their features are very similar to the ones presented in Unreal. Also, we focused on tools which impact was directed related to the game mechanics since it is one of the goals of this thesis.

However, since one of the ultimate goals of AI assistance is to reduce the effort of the game designers, we cited some tools designed by companies that had contributed to let the field closer to reach this accomplishment. One of them is Siren, a joint effort conducted by Epic Games in collaboration with CubicMotion, 3Lateral, Tencent, and Vicon.

Siren is a virtual human that aims to reduce the effort in the creation of realistic human models for games and movie industries. SEED, or the Search for Extraordinary Experiences Division is an R&D division in Electronic Arts whose goal is to leverage AI and explore new technologies that they can be able to make it work for future games. As early results, SEED has published and presented applications of AI for optimizing graphics and a neural network that controls agents able to play the game Battlefield.

2.10 System features: overview discussion

In the last two sections of this chapter we wrote about commercial tools for developing games and the AI methods they have implemented. Many of the tools we designed for this thesis are not part of the most popular game engines and frameworks nowadays. Here is a brief discussion of the features we implemented. Obviously, all the features will be explained in the upcoming chapters. The Game Rule Statistics (section **4.3**) is one of them. Although it is easy to find statistics tools in engines like Unity, they deal most with game performance technical details like frames-per-second ratio or the amount of rendering function calls. We don't see tools that gives users statistics about how the design space of the game is explored. SeekWhence (chapter 5) is also not

present in the popular game design tools. Although the game community can watch (and re-watch) game matches in channels like Twitch [104] or YouTube [137], very little they can do in terms of replaying the content in terms of game state by game state. And even if they do, they cannot import the game state to a tool and edit its content. Nintendo [2] has released a rewind feature for its Nintendo Switch Game Console. Although the rewind feature allow players to replay the game from any given point, a designer still cannot interfere on the content. A system like Kwiri (chapter 6) is not present as well. Although tools like Observer[1] can collect data about events in the game, and many analytics shows how users behave in the game. There is still no way of combining all the information and retrieve the exact game state where they took place that works for multiple games. Also, Kwiri captures these data automatically from game player agents without the necessity of relying on human work. Speaking about game player agents, the AI-debugger (chapter 7) assistant is also not part of any engine or framework distribution. Even with its limitations discussed on section **7.5** the debugger has no similar in its industry counterparts, mostly because the lack of game player agents to perform the tasks. Finally, Pitako (chapter 8 and chapter 9, the recommender system for game mechanics is not a feature that one could find in a commercial game editor. They do not keep a repository from where one could design a knowledge extraction tool, their scripting languages are a challenge to one create a categorization system for labeling behaviors of game objects, what makes impracticable the task of creating association rules that could attend a diverse array of games. It generates a bottleneck that end up creating issues when techniques like game blending is required.

[1] https://observeranalytics.com/

Chapter 3

Methods

In this chapter, we introduce the resources and techniques that served as methods for this work. We will be describing their foundations and discussing previous work that has been done on it. We will not be making an in-depth analysis of these methods, or any of their specific features since they will appear again on later chapters. The description is general and brings justifications about why we decided in favor of the methods presented here.

3.1 VGDL language and GVGAI framework

Cicero is based on the General Video Game AI framework (GVGAI) [105] and the Video Game Description Language (VGDL) [113, 42]. GVGAI is a framework for general video game playing. It has an associated competition in which contestants submit their best AI agents, which are judged based on their performance of unseen games. VGDL is the language used to describe games in this framework; the language is compact, human-readable, and capable of expressing a large range of 2D games based on graphical logic. Among the kinds of games which can be described are adaptations of many classic games developed for the Atari 2600 and the Nintendo Entertainment System. Because of the popularity of the GVGAI competition, there are about 150 VGDL games available and several dozens of effective AI agents, with varying strengths on different types of games [15].
A VGDL game is written by specifying four description sets.

- **The Sprite Set** - A sprite is an object in the game, including its graphical representation and behavior. In this set, the sprites are defined. It is the place to specify if your avatar can shoot and if a non-player character (NPC) will move randomly around the level or chase another game element. We stress here that in VGDL a sprite is not only associated with an image. The association is also related to a game element's

behavior. This is essential for this system since it is one of the main components of a mechanic rule system in VGDL. Therefore, every time the word sprite appears in the text, it is related to the concept of a sprite in VGDL.

- **The Interaction Set** - This set defines what happens when two sprites collide. It is the second part of the mechanic rule set in VGDL. Here, for example, you define when an element should be removed from the game after another one hits it. For example, when a missile collides with an alien airship and both sprites disappear.
- **The Termination Set** - This set defines the game over conditions. In other words, the conditions that dictate when a player wins or loses a game.
- **The Level Mapping Set** - This set is the only one that does not influence the game mechanics. Its purpose is to map sprites to a single character.

```
SpriteSet
    background > Immovable img=oryx/space1 hidden=True
    base    > Immovable    color=WHITE img=oryx/planet
    avatar  > FlakAvatar   stype=sam img=oryx/spaceship1
    missile > Missile
        sam  > orientation=UP    color=BLUE singleton=True img=oryx/bullet1
        bomb > orientation=DOWN  color=RED  speed=0.5 img=oryx/bullet2
    alien   > Bomber       stype=bomb   prob=0.01  cooldown=3 speed=0.8
        alienGreen > img=oryx/alien3
        alienBlue > img=oryx/alien1
    portal  > invisible=True hidden=True
        portalSlow  > SpawnPoint   stype=alienBlue  cooldown=16   total=20 img=oryx/door2
        portalFast  > SpawnPoint   stype=alienGreen  cooldown=12   total=20 img=oryx/dooropen1

InteractionSet
    avatar  EOS  > stepBack
    alien   EOS  > turnAround
    missile EOS  > killSprite
    base bomb > killBoth
    base sam > killBoth scoreChange=1
    base   alien > killSprite
    avatar alien > killSprite scoreChange=-1
    avatar bomb  > killSprite scoreChange=-1
    alien  sam   > killSprite scoreChange=2

TerminationSet
    SpriteCounter      stype=avatar               limit=0 win=False
    MultiSpriteCounter stype1=portal stype2=alien limit=0 win=True

LevelMapping
    . > background
    0 > background base
    1 > background portalSlow
    2 > background portalFast
    A > background avatar
```

Figure 3.1: A Sokoban game written in VGDL

The last set is used to associate sprites with symbols. This one does not influence the game rule description and game elements behaviors. The association is just a visual cue for

making the graphics process simpler.

There are many frameworks designed for testing algorithms for video game playing. Among them we can cite: MarioAI [128], for 2D side-scrolling games, AngryBirdsAI [111], for Puzzle-Physics Based Games, Pommerman [112], based on the classical puzzle Bomberman, FightingGameAI [78], for 2D fighting games, HearthStone [39], Hanabi [9], Starcraft [102], and MicroRTS [101] for real time strategy games. Some of these frameworks have the same name as the games they are based on and straight-attached to them. Others are more general and represents an entire game genre like MicroRTS or FightingGameAI. We decided to use GVGAI because among all of these frameworks, it is more flexible in terms of the games it can represent and the algorithms already available to play them, , what is possible because it has a description languages that other frameworks do not. With GVGAI/VGDL you can have action, shooters, puzzles, roguelikes, and strategy games. The agent library is rich as well, with tree-search based algorithms, genetic and evolutionary ones and variations of the Monte-Carlo tree search. It is still not ideal, all the games look like Atari 2600 or NES (Nintendo Entertainment System), no 3D games are available thus far, and everything is constrained to a grid-level approach. However, it is more generic than the other options cited above. One can argue that Unity and their Unity-ML-Agents package can be a better choice. In fact, Unity offers a power of expression higher than any other framework cited, but at the time we started this project, their ML-agents was not even announced. At the time this thesis is written, Unity-ML-agents is still a Beta concept, and despite it shows impressive results, it is still an open question how and when it will provide fast simulation, open source flexibility, and generalization among all the games a designer can create.

Note: we agree that the choice of the word *Sprite* in GVGAI and VGDL was not a good way to represent game elements. Therefore, whenever you see the word *Sprite* keep in mind that it represents a game element behavior (a game-playing agent or a game object that can chase another one, as examples) and not just an image.

3.2 Apriori Algorithm

Frequent itemset data mining is a family of methods designed to find association rules inside a long list of transactions [55, 51]. The way it creates rules based on a set of transaction suits the data we can extract from VGDL game descriptions. Because of it, we decided to use the apriori algorithm. The way it efficiently analyzes a database of transactions to find association rules between repeated items in different transactions is a good match for our database of game elements sets. An association rule says that the presence of an item **X** in

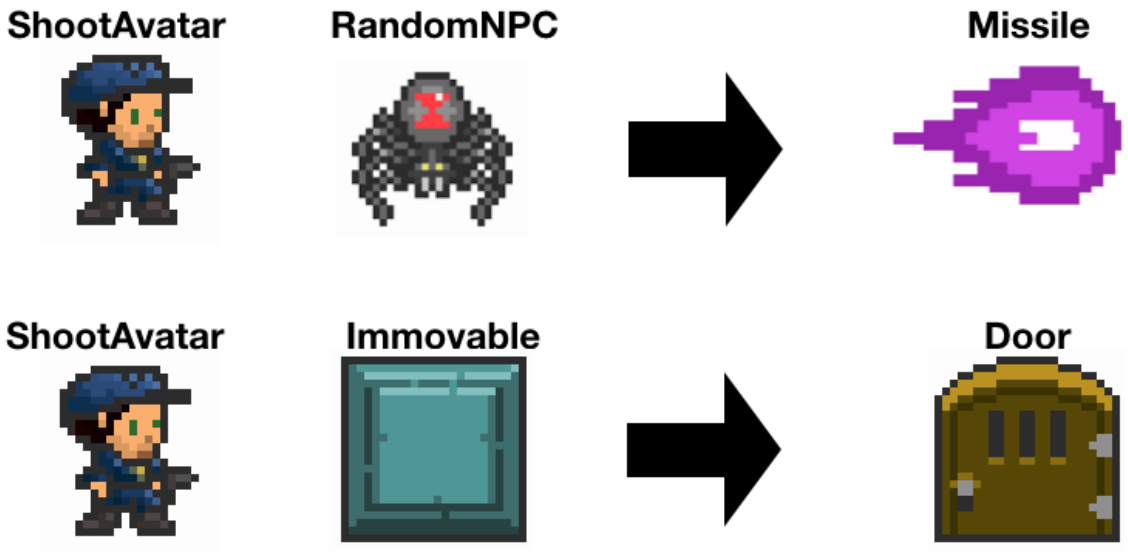


Figure 3.2: In the association rule on the top, the presence of a **ShootAvatar** and a **RandomNPC**, in a sprite set, implies in the presence of the **Missile** sprite. In the association at the bottom, the presence of a **ShootAvatar** and an **Immovable** implies in the presence of the **Door** element.

a transaction implies in the presence of **Y** in the same transaction with some probability (see Figure 3.2). A common example of this technique is the supermarket basket case. The article published by Agrawal et al. [5] exemplifies by stating that 90% of transactions that purchase bread and butter also purchase milk. The presence of bread and butter in a transaction implies the presence of milk in the same one. 90% is the confidence that such a rule will hold in a future set of new transactions. Finding all such rules is valuable for cross-marketing and attached mailing applications. Other applications include catalog design, add-on sales, store layout, and customer segmentation based on buying patterns [4]. We believe that games share behaviors in the same way that shoppers share items in their shopping carts. For example, *Mega Man X (Capcom, 1993)* and *Super Metroid (Nintendo, 1994)* have the same mechanic of shooting. On top of that, they even share the "hold the shoot button" rule to fire a more powerful shot. The idea here is that if your game has a shooting mechanic, you may want to use a powerful shot rule like the cited games. The presence of the shooting mechanic implies (with a certain probability) in the presence of the powerful shot rule. Given it is already available in previously designed games, it can be directly imported rather than re-implemented.

For the purpose of guiding the user to choose an item inside of recommendation list, it is necessary to displays confidence levels alongside each of its suggestions to help the designer understand the strength of each recommendation. Commonly used mechanics like the

'shooting example' above will appear with a high confidence level. Less common mechanics will have a low confidence level. This way, the designers can decide in which direction they will push their new games. Do they want to make a clone of an existing popular title (or maybe learn how it was done) ? Then, they can get the high confidence suggestions. Do they want to come up with something new and explore different possibilities? If so, they can go for the low confidence recommendations. Of course, they can use a mix of high and low confidence game elements and mix them with the ones they are creating by themselves.

3.3 MCTS and other agents

We have used a broad set of agents to assist us with our research and tool implementation. These agents are based on tree-search algorithms (like Breadth-First Search, Depth-First Search, and AStar) and evolutionary computation (like genetic algorithms). However, agents based on the Monte Carlo Tree Search algorithm were among the ones most used in all the cases presented in this thesis. While simple algorithms like the ones cited before are suitable for initial stages of implementation, for example, to test a heat-map visualization in a small map, agents based on MCTS, are good for scenarios that go beyond a simple automaton testing machine. In general, the MCTS works by analyzing which are the most promising moves and expands the space of exploration based on a random sample [48]. It uses an Upper Confidence Bound (UCB) formula to select the best child node at each level when navigating a tree.

The most popular implementation of MCTS is the Upper Confidence Bounds applied to Trees (UCT). The goal of MCTS is to get close to the value of the actions that an agent can take from its current state. This is achieved by building a search tree. The way that the tree is built will depend on how the nodes in it will be selected. This tree policy balances the search space between exploration and exploitation. After a node is selected, a simulation takes place. The simulation consists of doing random moves starting from the current game state until a terminal state is found. However, in practical terms, a pre-defined number of simulation steps is taken to avoid the algorithm to crash when resources are scarce due to a game complexity. After the simulation is completed, the final game state reached is evaluated and assigned a score. This score is then backpropagated from this node to its parents throughout the tree, until the root node is reached. Each node is updated during the backpropagation step, and they hold a total score, which is the sum of all backpropagated scores, and a counter that keeps track of the number of times the score was updated; this counter is equal to the number of times the node was visited [20].

MCTS has proved itself in board games, many types of card games and puzzles. The

algorithm can be considered new since its first publications are dated in 2006. However, it turned out to be the method selected for Go and “general game playing". Researchers and enthusiast had been working on an agent for playing Go for since the beginning of AI, but the results were far from something challenge and that could play even like a beginner. However, since MCTS was released, playing simulation quality increased in a notable way. Computers can now compete with human and even defeat top of the rank players as showed by AlphaGo [100]. In our case, the choice for having MCTS, was almost immediately since most of the General Game Playing Competition, and the General Video Game Playing Competition, the ones in which the framework our system is heavily used, agents based on MCTS are the ones which accumulate victories.

As mentioned above, MCTS is a tree search algorithm. It is similar to Minimax, an algorithm used for playing games such as Chess and Checkers for example. It brings, at least, two significant differences related to Minimax. First is that while Minimax explores all moves equally, MCTS focuses on exploring promising branches of the tree way deeper than others. The second difference is that while Minimax needs an evaluation function that assigns a value to a game state, MCTS evaluates a state through playing many games until the end by taking random actions. Therefore, MCTS does not need an evaluation function, works well for games where it is hard to analyze the quality of a game state, and it works for games with high branching factors.

3.4 Questionnaire based assessment

We used questionnaire based assessment for evaluating our systems and helping us when testing hypothesis.

3.4.1 NASA Task Load Index

The NASA Task Load Index is a widely used assessment tool that rates perceived levels of the workload from a subject after performing a given task. Initially, it was designed having the aviation industry as a primary target and used for usability studies and improvements on interfaces which required a high level of training to be managed by its intended audience. However, it did not take long for it to be adopted by other industries such as healthcare, entertainment, robotics, etc. Originally it consists of six dimensions which are rated in a Likert scale that goes from zero to ten. These dimensions are:

- **Mental** - How much mental and perceptual activity was required (e.g., thinking, deciding, calculating, remembering, looking, searching, etc.)? Was the task easy or

demanding, simple or complex, exacting or forgiving?

- **Physical** How much physical activity was required (e.g.. pushing, pulling, turning, controlling, activating, etc.)? Was the task easy or demanding, slow or brisk, slack or strenuous, restful or laborious?
- **Temporal** How much time pressure did you feel due to the rate or pace at which the tasks or task elements occurred? Was the pace slow and leisurely or rapid and frantic?
- **Performance** How successfully do you think you were in accomplishing the goals of the task set by the experimenter (or yourself)? How satisfied were you with your performance in accomplishing these goals?
- **Effort** How hard did you have to work (mentally and physically) to accomplish your level of performance?
- **Frustration** How insecure, discouraged, irritated, stressed, and annoyed versus secure, gratified, content, relaxed, and complacent did you feel during the task?

The NASA Task Load Index has gained multiple versions since its original implementation [57] and have been largely used by ergonomic and human factors researchers due to its easy application, reliability, and multidimensionality. Researchers have correlated the low indexes of workload measured by the NASA Task Load Index to gains of productivity and creativity as seen in the work of Nickulin et al. [98], in which the NASA TLX was applied to better understand and reduce stress levels during many phases of a production design process.

3.4.2 PANAS - The Positive and Negative Affect Scale

PANAS, or the Positive and Negative Affect Scale is a questionnaire divided into two sets of mood scales. One set measures an individual's positive emotions while the other measures the same individual's negative ones. Each set has ten dimensions that go from one (strongly disagree) to five (strongly agree). The ten positive and negative emotions are presented in the table 3.4.2. PANAS was usually intended to be applied during different time frames, but many studies have applied it as a post-test questionnaire to assess how an individual mood is at that right moment of the application. That said, it is suitable for test how people react towards the use of new technologies and new activities. Some studies focusing more on specific items of the two scales, like excitement (positive) and irritability (negative). Due to this flexibility and adaptability to many fields, as well as the NASA Task Load Index,

PANAS is widely adopted in research areas associated to Human-Computer Interaction, and also has many variations. The most common alternatives to the original PANAS are: the PANAS-C[73] which is intended for be applied with children; PANAS-SF[32], what is a just a simplified version of the original test, and the PANAS-X[134], an extended format which contains the positive and negative categories with the introduction of a third one that measures states of shyness and serenity. Researchers have reported that the high and low levels of positive and negative moods respectively are associated with increased levels of creativity as stated by Ding et al[38] and Baron et al [10].

PANAS List of Emotions	
Positive	Negative
Attentive	Hostile
Active	Irritable
Alert	Ashamed
Excited	Guilty
Enthusiastic	Distressed
Determined	Upset
Inspired	Scared
Proud	Afraid
Interested	Jittery
Strong	Nervous

3.4.3 Computer Self-Efficacy

Self-efficacy, or the belief in one's ability to perform a particular behavior [8], is considered one of the most important psychological studies in the organizational literature. It is also considered the single biggest predictor of behavioral change in individuals. While researchers in the last decades have also found it to be a good predictor of task performance (Gist, 1986), they have also noted the theoretical and methodological difficulties in measuring self-efficacy. They concluded that methodological and context themes need to be better explained [40] in order to achieve proper results. The information systems (IS) discipline started to discuss the need to design a separate computer self-efficacy (CSE) method in the late 1980s [33]. However, extensive use of the method did not happen until the development of an instrument to measure it in the mid-1990s [28]. Since then, it has been shown to influence many areas and presented impact in privacy, usage, and adoption [28] of IT products and services.

As an input, it is a key determinant of usage, and it has also been extensively used as an object of study for uncountable computer training programs [53]. However, many researchers have called attention for the fact that the CSE literature has presented variation in outcomes among different studies [28], and discussion about its purposes and goals in IS are still ongoing [138]. Based on these studies, discussion and research, two main forms of computer self-efficacy have dominated the information systems field: general computer self-efficacy (CSE) and specific computer self-efficacy (SSE). The most important difference between the two self-efficacies relates to the goals of the technology that the participants are asked to judge. While CSE judges general computer skills to accomplish a task, SSE judges computer skills in the use of specific software, such as a Computer Aided Design, used to accomplish a specific task. This research area highlights the fact that specific computer self-efficacy is a stronger predictor of outcomes [53], and thus, it is being increasingly used in information systems research [27]. For this study, our goal was to identify the participants' ability to perform a task by using a software independent of their skill level with it. Therefore, the use of the Computer Self-Efficacy Scale was more appropriate. The CSE Scale is a 10-item questionnaire that measures the subject's perception of success in performing a task by using a computer software. The scale is based on a Likert scale that goes from 1 (not at all confident) to 10 (completely confident). The items of the CSE scale are listed below.

- If there was no one around to tell me what to do as I go
- If I had never used a product like it before

- If I had only the product manuals for reference
- If I had seen someone else using it before trying it myself
- If I could call someone for help if I got stuck
- If someone else had helped me get started
- If I had a lot of time to complete the job for which the product was provided
- If someone showed me how to do it first
- If I had used similar products before this one to do the same job

Our research is based on the CSE Scale designed by [28]. From the ten items, the only one we removed was *If I had just the built-in help facility* because we did not design a built-in help for our system.

Chapter 4

The Cicero system

In this chapter, we introduce Cicero. It is the primary system of this dissertation, and all the research work done builds on it. We designed Cicero to encapsulate properties of the GVGAI framework and the VGDL language; it allowed us to expand the system and apply features to it. These features were tested to show the effectiveness of the whole system.

4.1 Cicero and its features

Cicero (Computationally Intelligent Collaborative EnviROnment for game and level design) is an AI-based game design tool which is intended to be significantly more general than previous tools while incorporating several types of AI-based design support. The system allows the design of games and levels in a wide range of genres. It can run simulations to provide data about the users' designs, fosters insights, and suggests what to do to improve their work.

Cicero allows developers to prototype games and their levels. In order to allow designers to prototype a wide range of games, we use GVGAI [105], and VGDL [42, 113]. Together, they provide a set of tools for developers to create AI agents that can play the games they have created. Our tool uses various AI agents available in GVGAI. Some are based on algorithms like Monte Carlo Tree Search (MCTS), others are the champions of previous editions of the GVGAI competition, an annual challenge that awards the AI agent which performs best in as many games as possible. The users can select these agents to play their games, or they can play the game by themselves. The main UI of our system is an editor for the VGDL language (see Figure 4.1). It allows the user to define behaviors for game elements such as NPCs and Power Ups, specify what happens when two game elements collide and determine the win and loss conditions. There is also a code area for those who like to inspect how their actions within the UI generate the code.

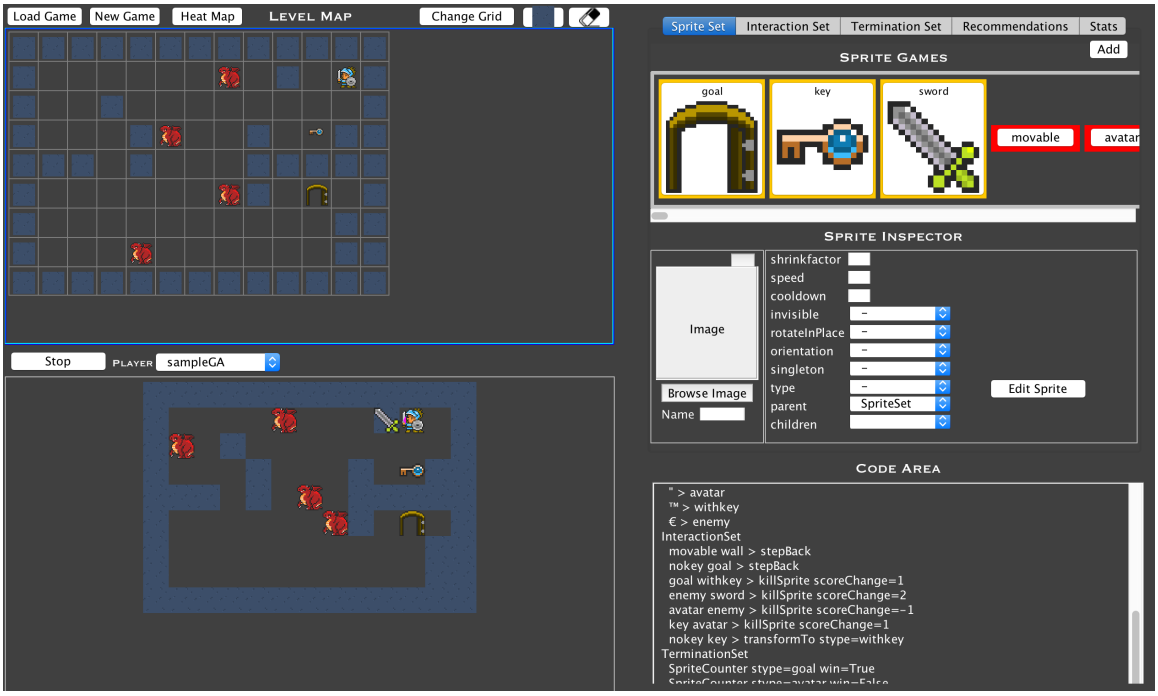


Figure 4.1: The system UI. On the left, the users can edit levels and play game simulations. On the right, they can specify the game's rules and game element behaviors. This picture shows the first version of our system entirely designed in Java.

4.2 Feature 1 - Game Rule Statistics

The game rule statistics module offers a diagnostic about how a controller explores the rules of a game. It shows a list with every rule of the game, sorted by most to the least fired by an agent. With this diagnostic, a designer can see if some rules are never used, and identify if this is due to a design flaw. The designers can use this information to optimize their game by removing unused design elements or change the parameters and rules to force the agents to explore more or less of the game design space.

4.3 Feature 1 - Example of Game Rule Statistics

As stated before, the game rule statistics show a diagnosis of a game played by a controller. We chose the "adrienctx" algorithm, a winner of the GVGAI Single-Player competition in 2014, based on Open Loop Expectimax Tree Search. Using this agent, we ran simulations of the game"Zelda", a VGDL clone from cave levels of the Nintendo Entertainment System's "The Legend of Zelda." We chose "adrienctx" because it is a controller that explores the physical space very well for most of the VGDL games available. A useful characteristic when performing a diagnostic of the game rules in use.

In the game “Zelda," we noticed that two rules are never used. One of them prohibits the player from reaching the goal without a key. The other one will trigger if an enemy kills the player.

movable	wall	stepBack	0.6842...
enemy	sword	killSprite	0.1578...
goal	withkey	killSprite	0.0526...
key	avatar	killSprite	0.0526...
nokey	key	transformTo	0.0526...
nokey	goal	stepBack	0.0
avatar	enemy	killSprite	0.0

Figure 4.2: List highlighting how an agent is exploring a game design rule space. Probably this is not a good agent, since it never kills an enemy (last item in the list from top to bottom) and spends most of its time bumping on walls (list first entry)

This allows us to inspect the level design and rules of the game more accurately, and design specific test cases. For example, to see if the rule that makes an enemy kill the player is in use, we simply need to run the simulation again without allowing the player to use his weapon, or place enemies surrounding the player’s spawn point.

The other rule, the one that prohibits the player from reaching the goal without a key, allows us to draw some conclusions about the original level design. First off, the rule does not trigger because the controller always retrieves the key. In the original level design, the key is near the player spawn point, so there is no challenge in accomplishing this objective. In order to correct it, if the designer wishes to do so, to increase the challenge and thoroughly test all the rules, they just need to change the position of the key or player’s spawn point (See Figure 4.1) in the UI level editor (See Figure 4.2 for a sample of the statistics report).

4.4 Feature 2 - Heat-map Visualization

The visualization prints the playtrace aggregation as a heat map of the game object behavior over the level. This approach is inspired on game player behavior studies, like the visualization softwares **G-Player** [25] and **Glyph** [97]. It extends every Agent class available in the GVG-AI Framework. Therefore, every agent contains a TrackControl object. This object stores every position of the agent during gameplay. It also stores a count value of how many times the same position was visited. So, in every game update, the TrackControl defines the alpha channel of every position. It normalizes by the most observed count value (the most visited one). Then the alpha channel of every position is defined by the count value of the position divided by the count value of the most visited one. The same principle is applied to all other game objects in the game (i.e enemies, items, etc.).

4.5 Feature 2 - Example of Heat-map visualization

The heat maps (See figure 4.3) provided by the visualization system are available for each level of each game and can be used with any agent. The editor is context sensitive with respect to the game definition, so every time a game is loaded or has some changes, the editor adapts and allows the users to customize what they want to see.

4.6 Cicero's Design Process

Cicero is designed based on an interaction design approach. We are continually evaluating and getting feedback from the users. The version presented in this chapter is the first one of our system. Everything was implemented in Java, primarily because we thought it would be the straightforward way to integrate with the GVGAI framework, written in the same language. Therefore you will see different graphical layouts along with this dissertation. They reflect changes in technology and changes in UI components and layout design. These new adoptions and the new features presented in the following chapters came, also, as a result of our user tests.

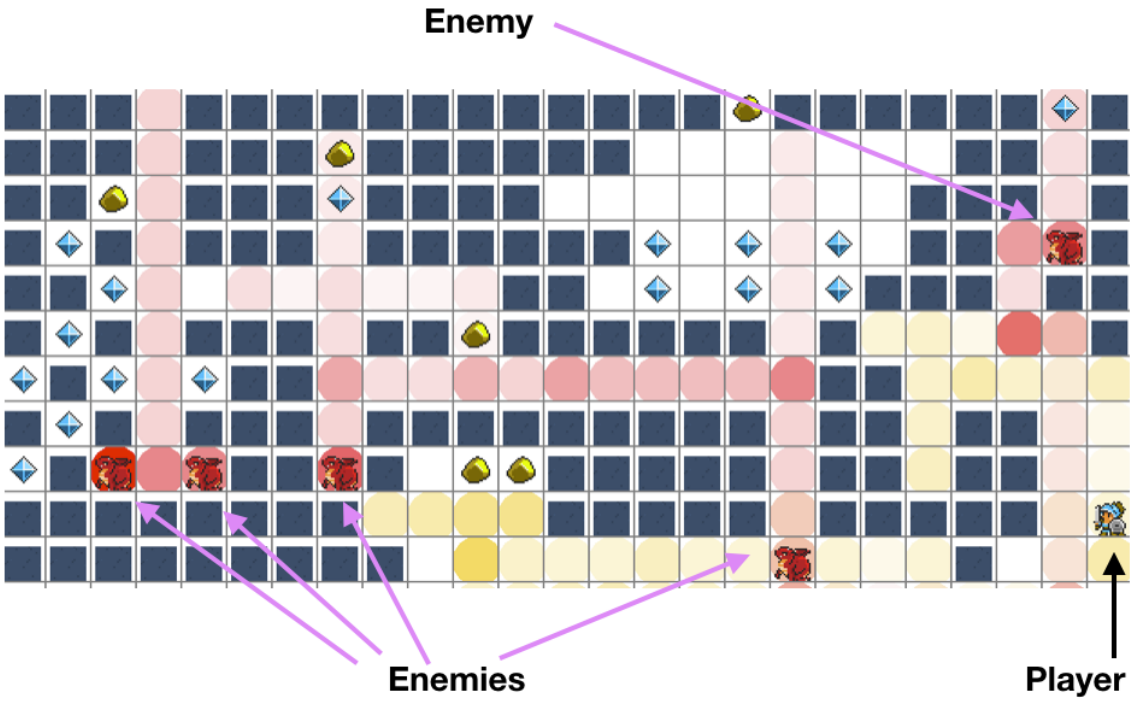


Figure 4.3: Vizualization of playtrace aggregation. Red dots show how enemies are exploring a level. Yellow ones show how a player is exploring it.

Chapter 5

SeekWhence

Early evaluations of Cicero's first version revealed that the users were in need of a tool to offer them the possibility of going back and forth in a gameplay sequence and highlight it with the Cicero's visualization system to help them in evaluating their players (human or agents) and find system failures. SeekWhence was designed to attend these requirements, it is a retrospective analysis tool enriched by a visualization system with the power to replay games played by humans or artificial agents. It does telemetry whenever a game is playing. Every frame of the gameplay session is stored and becomes available to the users in a sequential media file. The users can play it and analyze all the gameplay session frame by frame. They also can activate a visualization system to further help with the analysis task. Finally, they can import any frame of the sequence into their current project to edit it with all the information about the game and its components already available, from the beginning until that point (the frame which was chosen to be imported). In this chapter, we report our experiences regarding SeekWhence development and the benefits that game developers can have with such a retrospective analysis tool. We discuss our approach, the background which motivated this work, and our telemetry process. Finally, use cases and examples which illustrate the uses of the tool are presented as well.

5.1 Game telemetry and analytics in Cicero

We used Cicero and expanded it to have a database which would be able to gather in-game data through its agents in a variety of different games. In order to analyze these data, SeekWhence allows the designers to inspect their games step-by-step, they can interpret the data as a whole sequence of events or any event in the sequence separately. Besides that, they can import any frame of a sequence into their current project, edit, test, and play it as if it was designed at that single moment. The development of SeekWhence was inspired

by a previous evaluation of the Cicero system. When testing the application, several users suggested that it would help have a tool to help them in seeing what was happening in the game at every time-step, almost like a slow-motion retrospective analysis.

Retrospective analysis was previously used by [130] as a player-centric visualization work geared to evaluate types of visualizations in the team-based combat game *World of Tanks* [1]. It was also presented in the serious and open-ended game *RumbleBlocks* [56] where the authors used the technique to solve alignment issues.

Following the terminology defined by [16], whereas the work of Wallner et al. [130] is a retrospective analysis tool designed with the purpose of training players and the work of [56] is designed to align game design with educational outcomes, our work is a retrospective analysis tool designed to help developers in tasks such as debugging and balancing. Therefore, its purpose, target audience, its usage across different game genres, the use of agents to simulate the games, and the visualizations to enrich the navigation are the main contributions of SeekWhence.

5.2 Retrospective analysis and visualization in Cicero

This section details the implementation of SeekWhence, with explanations about the GVGAI and Cicero's UI extensions.

5.2.1 GVGAI Extensions

GVGAI (see section **3.1**) has two classes which are fundamental to this implementation: the **StateObservation** and **AbstractPlayer**. The former is responsible for keeping information about the game at every state, the latter is the class that every agent has to extend in order to execute their functions when playing. Every agent overrides a method called **act** from **AbstractPlayer**. This method receives a **StateObservation** as a parameter. It analyzes the current game state, and as a result, it takes one of the actions available to move to another state. The agent has one game tick to take a decision, and every decision leads to a new state configuration.

This means that at every game tick, the sprite level matrix, contained into the **StateObservation** is stored. This matrix contains the position of every sprite in the level and the unique sprite ID, which we can use to access sprite information at that particular game tick.

Also, at every game tick, the position of every sprite is stored in order to do a summation and come out with the normalized alpha value for the color track at that position. It is storing the visualization information of each sprite for every game tick. This computation considers

the range that goes from the first to the current game tick. In other words, it is pre-caching the necessary information in order to provide a step-by-step visualization to the user.

Then, when a gameplay session is ended by an agent winning or being defeated (or by a decision of the developer) every information tracked on the fly is then stored in different files.

5.2.2 SeekWhence File Generation

For each new game tick, a SeekWhence file is generating by combining the collections of level map matrices and the collections of stored color tracks (see Figure 5.1) of sprites and events. More specifically, there are four files to know:

Map Matrices This file stores the position of every sprite in the level in the form of a 2D matrix. It is straight related to the **LevelMapping** section of the VGDL game description (See ssection 3.1), and each index of the matrix contains the necessary information to rebuild the level at the specified game tick.

Avatar This file stores the color tracks of the positions on the level visited by the avatar (human player or AI agent).

Enemies Analogue to the last one. However focused on enemies[1].

Events Stores the color tracks of any given event. An event in a VGDL game is every interaction among two sprites described in the **InteractionSet** (See section 3.1). We have kill events, clone events, hit events, dropping resource events and so on.

The first step to assembly the SeekWhence file is to read the Map Matrices file and for every tick, we recreate, in a secondary file, the level as it was.

So, when having the level sequences all together and indexed by game ticks, we can access every frame from a stored game session step-by-step. Furthermore, based on the same tick, we can print the visualization color tracks for game elements and events.

5.2.3 Retrospective analysis - Cicero UI extension

In order to allow users to analyze the sequence of events, we extended the Cicero's UI. We implemented a component similar to a video player (Figure 5.2). The user can interact with the component by clicking on the arrow buttons (left and right) to go back and forth in the

[1] As other game elements are analogue, we decided to skip their explanations for the sake of space.

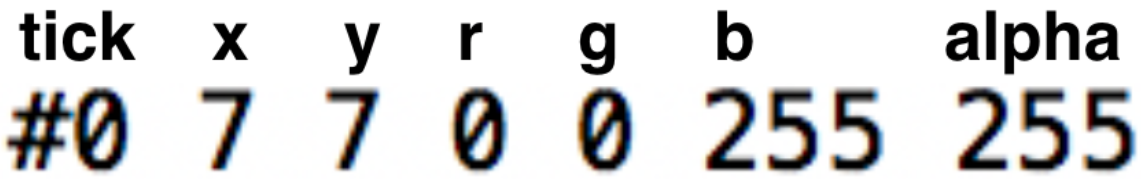


Figure 5.1: Every color track stores the game tick when it was registered in a gameplay session, its *x* and *y* position on the matrix and the its RGBA values.

sequence frame by frame. A slider bar also does this service allowing the user to skip several frames quickly. The component also has an export button, which allows the user to export any particular frame to the Cicero's main interface. An important highlight of SeekWhence is that it is not just a tool for *Replay Theater* in which the users can watch a replay of a game session and have control over what they are watching. More formally, *Replay Theater* in general, works like a video where players use to see their own and their opponents progress. It is also in use by developers to catch bugs. However, in essence, it restricts viewers to watch a non-interactive animation [16]. SeekWhence enhances the *Replay Theater* concept by allowing a designer to actually use any given frame of any gameplay video sequence, edit it, and changing whatever game element they want, like the agents for example.

By editing, we mean the process of describing the level map. By importing a frame from SeekWhence, the user gets all the information of the level (and its elements) at that state of the game. By changing it, the user will change the sequence of game states, from the one being edited to last until the game finishes. Obviously, it can reach many of the former gameplay session states, but it is not guaranteed since it depends on the kind of changes the user did and the algorithm (or player) decisions when playing again.

5.3 Use Cases

This section presents some use cases and examples where SeekWhence can be used: agent and level evaluation. We also present our findings based on an informal test conducted with three volunteers.

5.3.1 Agent evaluation

Designers can use SeekWhence to help them in analyzing their agent's implementation. In the usual way, to specify in detail the agent behavior, the user needs to use the available Java debugging tools included in the most popular IDEs among developers. However, these tools do not offer an easy way to navigate through all the data the users need to have clear

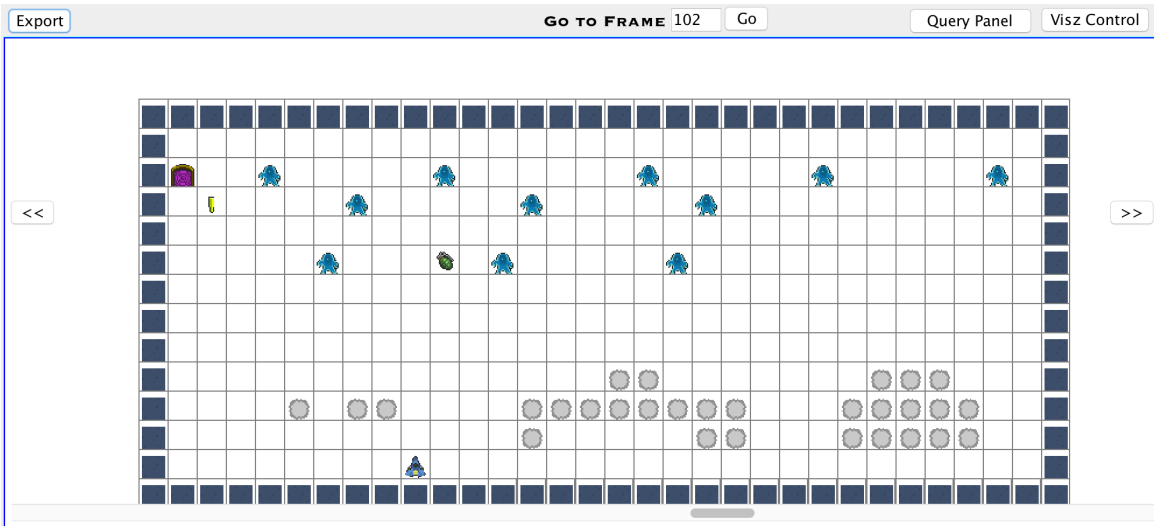


Figure 5.2: The SeekWhence UI contains buttons to allow a user to go back and forth in the stored gameplay session. It also contains a slide bar, a search for a frame specified by its tick number, button to turn the visualization on and the button to export the frame to the Cicero's main UI.

and easy comprehension about what their agents are doing. There are no graphics or other visual clues that could help. Besides that, it is responsibility of the users to write some parts of the code in order to access specific information.

SeekWhence, in this case, can present to the users a visual interface to go back and forth through a recovered gameplay session. They will not only know where is the agent position, but they will also see where it is at any given game tick. More than that, they will also see where are all the other elements in the level and how they may (or may not) influence agents decisions. This is much more convenient than checking printed lines in a console showing positions and actions of game elements.

Notably, if the users are trying to do a hybrid agent, SeekWhence offers the option of changing the game and level configurations whenever they want. So they can start a sequence with a *AStar* agent, change it to a *Monte Carlo Tree Search* and finally use one of the champion agents of the GVGAI competition. Then, by running the recorded sequence, they analyze the agents individually and come out with their conclusions about how to improve their hybridization. This kind of advantage, import the stored frames and editing the level, makes SeekWhence more than just a *Replay Theater* application, as mentioned before in subsection 4.3. In the example illustrated in this section, the designer switches among a human player and *adrienctx* (a previous GVGAI champion).

In Figure 5.3, we see a human player who is killed continuously by random enemies. The enemies are fast, and they make hard his mission of getting the key. The designer got

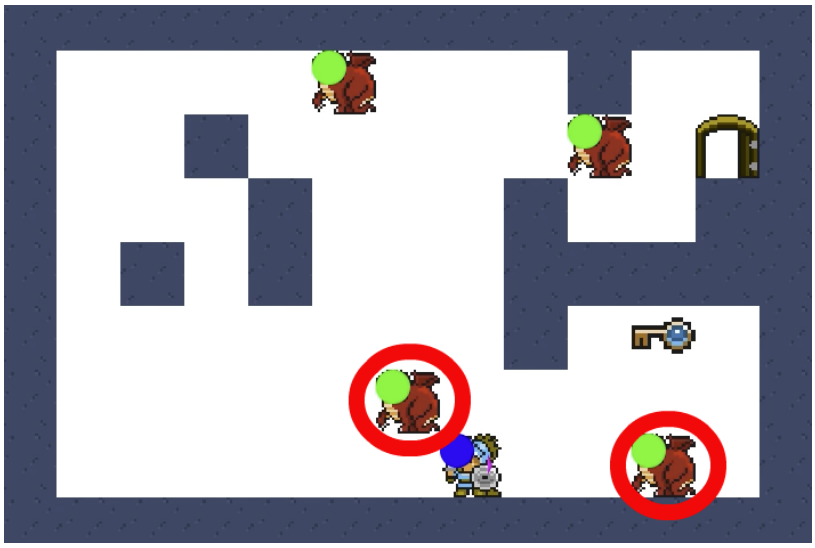


Figure 5.3: Human player playing a cave level. This is the last frame before he got killed by the enemies highlighted by the red circles.

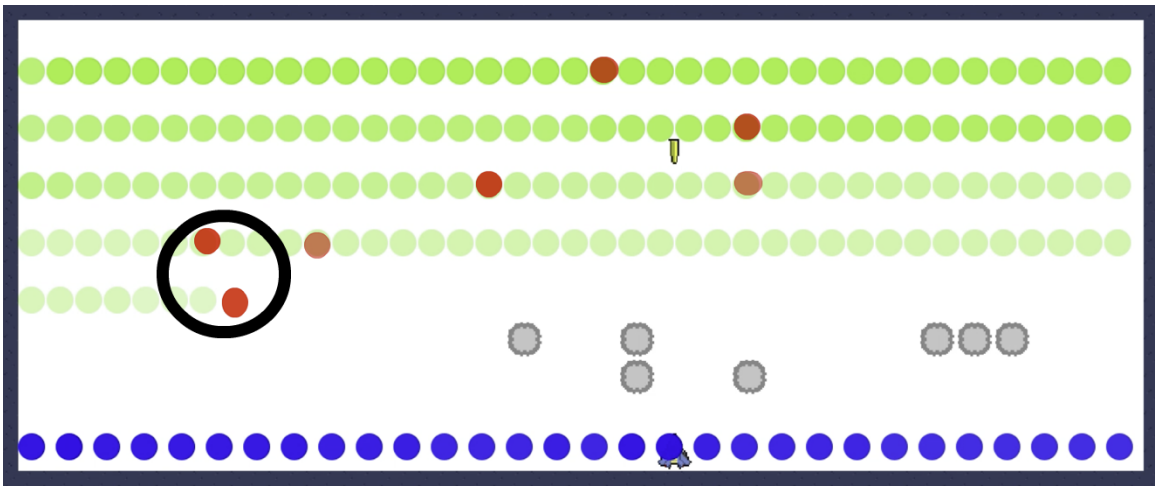
the last frame of this session played by the human (the frame before his avatar get killed by one of the enemies) and replaced him by the *adrienctx* agent. It was able to kill the monsters and get the key. However, it wastes too much time walking in circles and attacking empty spaces. At this point, the designer gives the control back to the human player to play again and complete the level.

5.3.2 Level evaluation

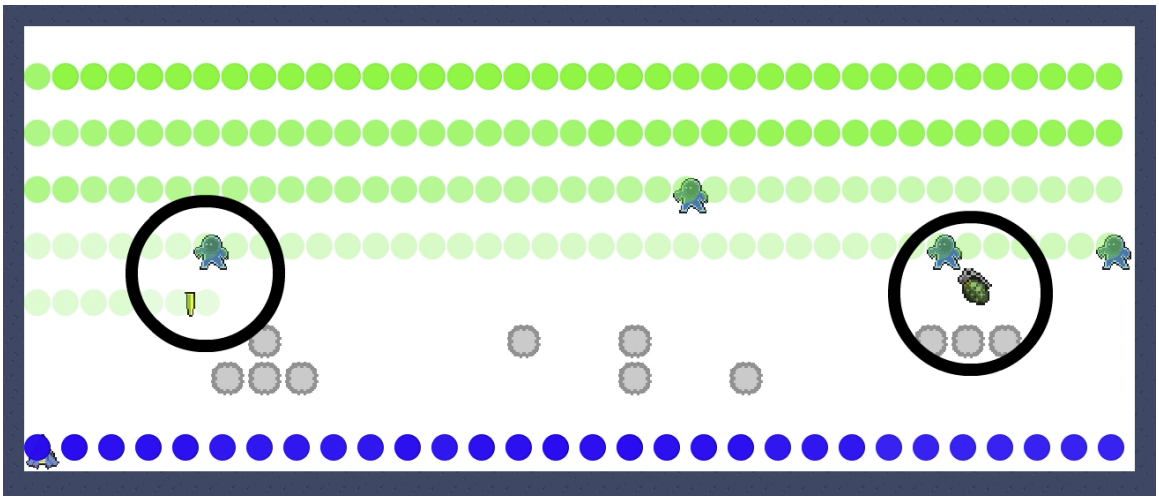
Another use of SeekWhence is level editing and evaluation. Although there are many methods and heuristics involved [36], level evaluations are done by players following formal or informal procedures [37] in which they answer questions and provide feedback about what they think. Sometimes both the designer and the player needs to use their memories in order to remember all the events that happened during the session, what can be pretty inaccurate. So by providing a way of accessing these events, SeekWhence can help users and players to express themselves better and increase the accuracy of their communication [110]. Besides that, if the players have the same opinion about a similar event, it is difficult for the designer to access the exact point and rebuild the level to edit it.

SeekWhence allows designers to have these kinds of access. They can go to the mentioned events, import the frames and all the information attached to, inspect them as many times they want to and perform the changes they find necessary.

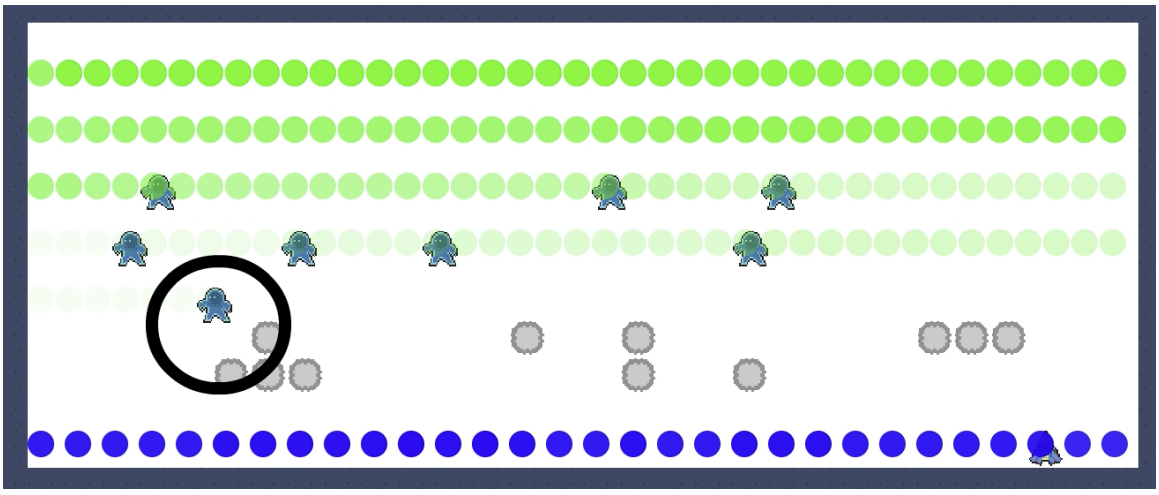
We did small changes in a VGDL clone of the classic game *Space Invaders*. We increased the enemies speed, what does the game pace extremely rapid. Also, we populated the level



(a)



(b)



(c)

Figure 5.4: From (a) to (c) we can see frames of a Space Invaders gameplay session. The black circles highlights areas where the users found flaws in the game rules.

with barriers that could not be destroyed by the enemies' bombs. To evaluate the gameplay session, the designers need to use SeekWhence; otherwise, by not see it step-by-step, they can lose essential details due to the speed of the game.

In the figure 5.4 we have the last frame of the VGDL *Space Invaders* clone 10.1a. The red dots show areas where the enemies were killed. Green dots show the enemies‘ moves and the blue ones shows the player's moves.

By running SeekWhence and moving the gameplay session backward, the users can see that a barrier is killing enemies 5.4c. The visualization system highlights this fact. In the figure 10.1a, the black left circle shows two killing spots, one of them is precisely the position of a barrier. Also, they can notice another flaw rule in figure 10.1b. While in the left black circle, they can see an enemy getting a shot from the player in the right black circle, the enemies' bombs are not destroying the barriers.

5.4 Conclusion

In this chapter, we presented SeekWhence. It is a tool that allows users to do sequential analysis in a recorded gameplay session played by agents or human players. SeekWhence works as a feature of Cicero and uses its visualization system to enhance its replay analysis purpose. The users can analyze a level step-by-step and use their conclusions, at least, in two situations: to build and debugging agents and to edit and evaluate levels. As users can go back and forth in all the sequence of events stored in a session, they can use every frame of that sequence and export it to Cicero. Then they can edit the level as they wish and with all the information until that point already available. We did informal tests, and the first reactions were positive. We used our findings to improve the experience with SeekWhence and develop a query system. The query system is intended to let the replay analysis even faster and accurate. The design and implementation of this query system are presented in the following chapter.

Chapter 6

Kwiri

Kwiri expands the concepts presented in the previous chapter with SeekWhence. It is a query system that allows users to query for game events in terms of what (was the event), when (did it happen), where (did it happen), and who (was involved). With such a tool, we explore the possibilities of applying queries to provide game-general AI-based design assistance. Throughout this chapter, we will discuss the motivation, the design of Kwiri, use cases, and a preliminary qualitative study. Our first results show that Kwiri has the potential to help designers in-game debugging tasks, and has been served as infrastructure to build another system which relies on querying for game events.

6.1 Motivation

we designed Kwiri on top of SeekWhence. We were motivated by the idea that a designer (or tester) of a game will often want to figure out when and where something happens, and this might not be evident when either playing a game or watching a replay. For example, imagine that a particular NPC (Non-Player Character) occasionally dies even when one of the player's bullets do not hit it. To find out what is going on, a designer would need to rewatch endless replays attentively. However, what if you could simply ask the game when and where an NPC died? Kwiri makes use of the fact that the games that Cicero work have formally defined game mechanics, and provides the ability to interrogate replays for particular combinations of events. Kwiri contribution relies on its generality. It is a single query system, whose interface allows designers to ask questions about any genre of game they can design with the platform. It also makes use of gameplay simulations and does not depend on human users to generate in-data game to be analyzed.

6.2 Design of Kwiri

As stated before, the query system is developed on top of Cicero and SeekWhence as well, and because of that makes use of the visualization system. In this section, we will highlight some of the implementation details about these two systems in order to facilitate further discussion about how the query system works.

6.2.1 How SeekWhence and the visualization system work

SeekWhence stores every frame of a gameplay session played by an agent or human player. In order to store the frames, we capture every game state, at every game tick. The game state contains all the information of the set of game elements available, such as their positions in the level for example. The indexation by a game tick is what makes SeekWhence runs like a video player. The level map of the game state is converted to a plain text format. Specific information related to elements present in the game state map is stored in a different file. Game ticks index both game state level and element information.

Every game element can be assigned to a specific color by the designer. It activates the visualization system [Figure 6.1], which captures all the positions of the elements in the level and apply a heatmap to show which areas were explored more.

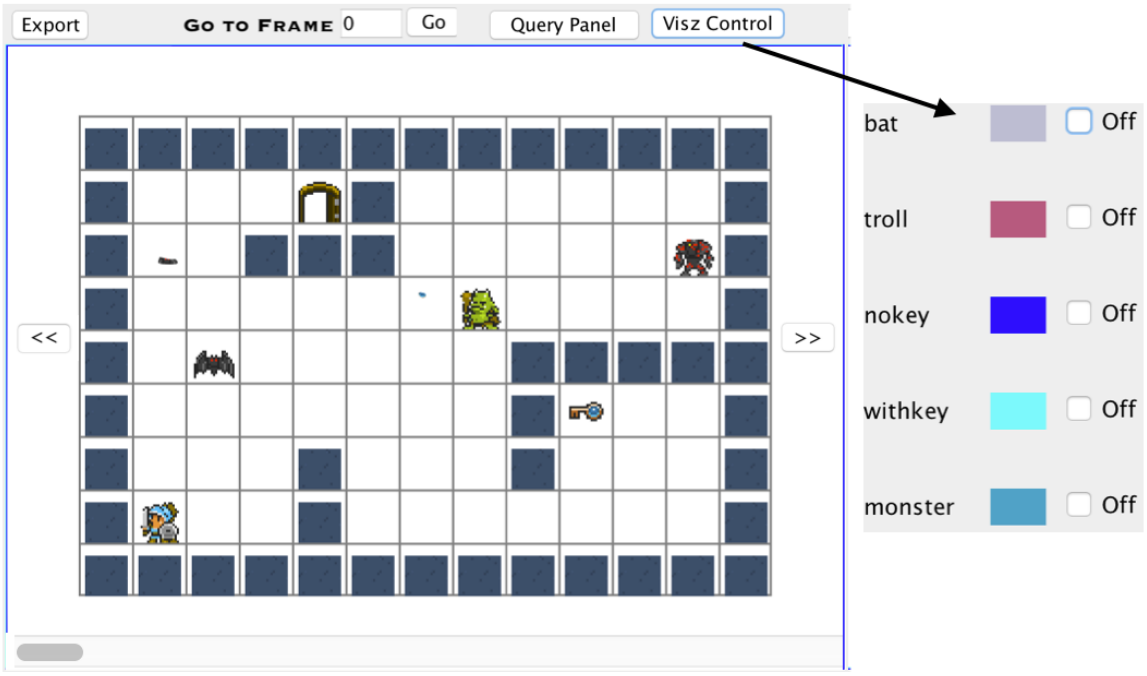


Figure 6.1: SeekWhence panel and its visualization control

6.2.2 Kwiri Implementation

Our query system is implemented on top of everything we previously discussed. It adds a database (***MySQL***). The database stores events and performs query searches related to them. An event in VGDL happens whenever two sprites collide, it is up to the designer to define what kind of event will be fired at the moment of the collision. It can be a *killSprite*, a *cloneSprite*, and more than 20 other ones. In other words, the events are the rules of the games. The use of a database implements the storage and the searching process more accessible and straightforward. The queries are responsible for searching answers for the questions What, Where, When and Who are provided to the users as a GUI so that they can insert the query's parameters [see Figure 6.2 on next page].

6.3 Example usage

We believe that Kwiri can be used to explore solutions for common and novel game problems. In the following subsections, we show some examples.

6.3.1 Quantitative User Study

Kwiri was also used in a quantitative user study [83]. The goal of the work was showing that humans with AI assistance can be more accurate in-game bug detection than humans without assistance. In one of the tasks, agents collected data, and the users of group A had to use Kwiri as a way to filter events and figure out what was causing the failures. For the same task, group B users just had SeekWhence available. The possibility of filtering the events made the users approximately 32% better than the ones without it. More details about this study will be presented at chapter 7.

6.3.2 Automatic Game Tutorials

The work of Green et al. [52] introduces a fully automatic method for generating video game tutorials. The AtDELFI system (AuTomatically DEsigning Legible, Full Instructions for games) was designed to research procedural generation of tutorials that teach players how to play video games. In the paper, the authors present models of game rules and mechanics using a graph system as well as a tutorial generation method. The concept was demonstrated by testing it on games within the General Video Game Artificial Intelligence (GVG-AI) framework. AtDELFI uses Kwiri as a way to search for the critical events that make a player win and lose a game. The graph generated by AtDELFI starts the query engine that

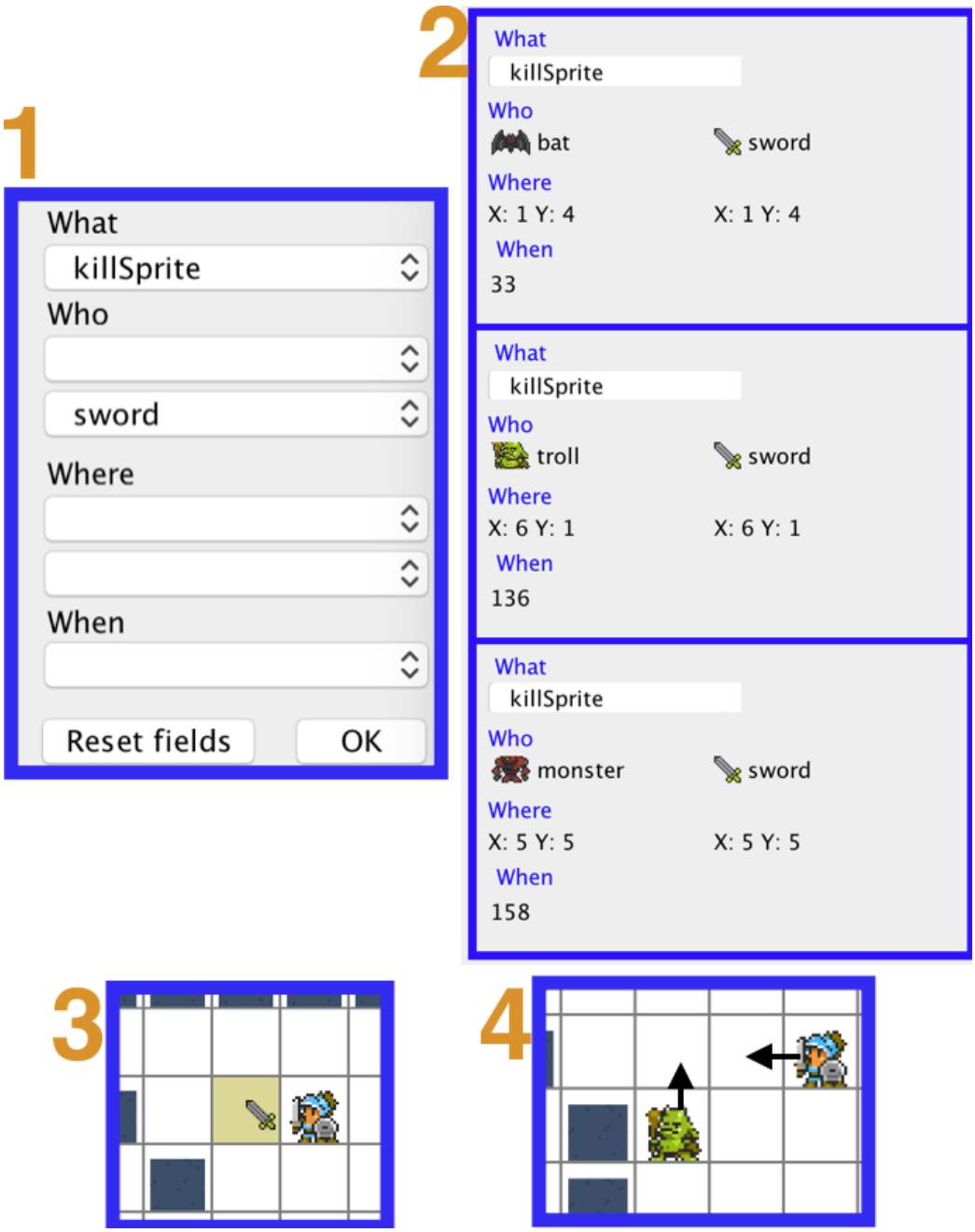


Figure 6.2: (1) The user query for "kill" events involving the use of the avatar's sword. (2) the system presents the query results. (3) After the user clicks on the middle panel, the system changes its focus to SeekWhence and shows the exact frame *when* the event happened. It also highlights the position *Where* the event happened. (4) By moving one frame backwards, the user can see the sprites positions after the event.

captures the events and returns a sequence of frames, which are used to generate videos of the tutorials.

6.4 Preliminary User study

We employed a qualitative method in order to understand, from our users, the benefits of Kwiri, as well as to solicit suggestions for future additions and improvements.

6.4.1 Study Design

We created one inconsistency in the rules for each of three different VGDL games: Zelda, Aliens, and FireStorms. Their official rules are listed below, as well as their inconsistencies.

Zelda - An action game (a clone of the cave levels of *Zelda: A Link to The Past*)

1. Player
    - Cannot go through walls;
    - Can kill enemies with a sword;
    - Changes its sprites when it gets the key;
    - To win the level needs to get the key and access the gate.
2. Enemies
    - Cannot go through walls;
    - Can occupy the same sprite of another enemy, a key and a gate;
    - Cannot kill other enemies;
    - Kill Player when colliding with it.

The inconsistency in this game is that some enemies can kill other enemies.

Aliens - A Space Invaders clone

1. Player
    - Kill enemies and barriers by shooting towards them;
    - Cannot go through walls;
    - To win the level needs to kill all the enemies.
2. Enemies
    - Cannot go through walls;
    - Kill Player by shooting towards it or when colliding with it.

3. Barriers

    - Are destroyed when hit by enemy bombs, by player bullets or when colliding with enemies.

4. Bombs/Bullets

    - Just kills enemies/ Just kills the player;
    - Are destroyed by walls.

The inconsistency here is that some barriers are not destroyed by bombs.
FireStorms - A puzzle game

1. Player

    - Cannot go through walls;
    - Cannot kill enemies;
    - Is killed when colliding with enemies or fire balls;
    - Wins the game when it reaches the closed gate.

2. Enemies

    - Cannot go through walls;
    - Can occupy the same sprite of another enemy, a purple portal, the closed gate and the fireballs;
    - Cannot kill other enemies;
    - Kill Player when colliding with it.

The inconsistency is that one of the enemies can walk through walls.

6.4.2 User Tasks

The user tasks consisted of finding the inconsistency in each one of the three games. To perform the tasks they were allowed to use Kwiri, and obviously, combine it with SeekWhence and the visualization system.

6.4.3 Participants

The study had nine participants, all of them male, eight of them were enrolled in a university program (7 Ph.D. students and one undergraduate student), and one was a digital media professional. Just two of them did not study/work with games/gaming. The other ones had mixed experience between industry and academic fields, varying from two to fifteen years. The engines and frameworks they were more familiar with are Unity, Phaser, and GameMaker. They were recruited through the department e-mailing list.

6.4.4 Procedure

The first step of the study was to ask the participants to fill out a form about their demographic data and game development experience. After that, we asked if they would be comfortable with their voices being recorded. Then we explained how SeekWhence and Kwiri work. The explanation took less than five minutes. In the end, we asked the users if they wanted to ask any question or skip it to use the tools for a quick warm up.

The second step started by introducing the users to what would be their tasks during the experiment. We informed them that we would give them three different tasks and that they should use the set of features to find a design inconsistency in each one of them. For each task, first, we handed the users a sheet of paper with all the rules of the game in the evaluation. After they read the rules, we ran the game with the agent *"adrienctx"*. We choose this agent because it is a former winner of the GVGAI competition. It can play the games well, and most of the times can beat the levels it is playing. Then, all the data were available to the user to actually start to work with them in order to try to solve the task.

The third and last step of the procedure consisted of a conversation where the users could express their opinions about the tested features and compare them with others they used before.

We checked in our users after five minutes of working on a task. We decided on this number after three pilot tests executed with a preliminary version of the system. However, this time is not a measure of success. It was only used to verify if the user is feeling tired and uninterested in continue after five minutes of effort. We gave them the option to keep trying or move to another task.

6.4.5 Source of Data

We used three sources of data in order to collect the users' activities: direct observation, audio recordings of test sessions, and a design questionnaire.

6.4.5.1 Direct observation

As participants interacted with three different games and were allowed to speak about their findings, problems, and suggestions related to the tools, they had their voices recorded, and written notes were made of their overall patterns of use and verbal comments. Attention was paid to participants' interaction between the different systems (SeekWhence, Kwiri, and visualization), how hard they had to work to find the inconsistency, what design features attracted their attention, and whether, at any stage during the study, they seemed to lose interest in the activity.

6.4.5.2 Audio recordings of test sessions

Every user had their voice recorded. This helped us to validate our written notes since we used them as tags to pay attention when listening to their recordings. Also, we used this source to clarify some actions they performed, which were not initially apparent to us.

6.4.5.3 Design questionnaire

Our design questionnaire is based on a semi-structured interview and required factual (e.g., "When I was querying for what a sprite was doing."), perceptual (e.g., "I think that the query results annoyed me with too much information.") and comparative (e.g., "It would be a plus to have these features in the tools I have used before"). We started the questionnaire by asking general questions in order to let the user feel comfortable like "Which are your thoughts about these features?". Then we moved to specific ones, many of them influenced by our notes, like: "In the second task, could you explain what you were trying to do when you asked if you could type your queries?". Finally, we asked if the users had more suggestions besides the ones they suggested during the tasks.

6.4.6 Data analysis

Our data analysis is based on text transcriptions of the design questionnaire discussed in the previous section. We used a standard procedure, Qualitative Content Analysis since it is considered suitable to analysis text materials, which varies from media products to interview data [11]. Normally it creates key common points identified among different users interviews in a technique called by codification theory, which helps researches in quick organizing and managing qualitative data. To facilitate the codification process, we used the trial version of the software Atlas.ti [49].

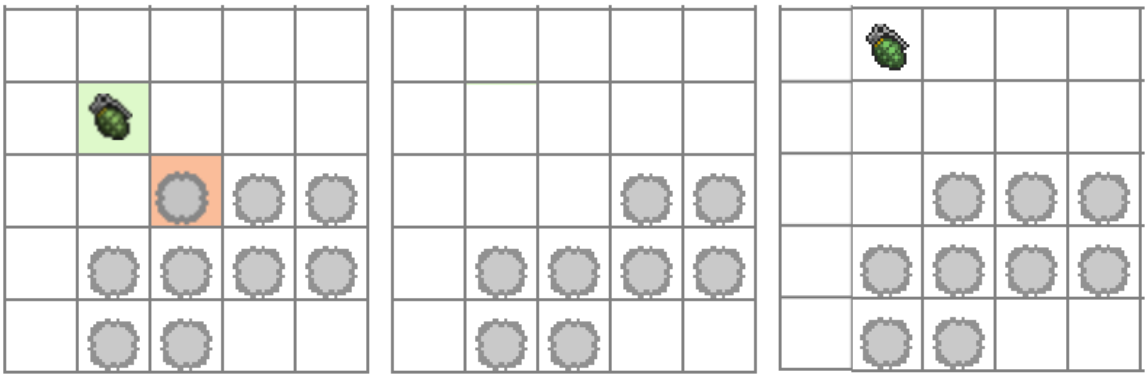


Figure 6.3: (Left) The users found a bug we were not aware of by querying for kill events involving a bomb and a barrier. (Center and Right) The users navigating forward and backward to confirm the inconsistency: the bomb was destroying a barrier which wasn't on its line of fire. Later on during the interviews, some users stated that the green and red patterns (Left figure), used to express the sprite who does the action and the one who suffers, should also be indicated on the query panel.

6.5 Results

As explained in the previous section, we did a qualitative study to know from our users which are the significant gains of our tools and which are the points of improvements in a next iteration of our design process. We will present the data first with a general discussion of our findings by observing the users. Then we will present our conclusions based on the users' point of view.

6.5.1 Task results

Most of the users were able to complete all three of the tasks with success. Just two gave up, one during the *Zelda* task, and another during the *Aliens* task.

They started by exploring their options, right after reading the game rules given to them. Some started by using SeekWhence and tried to figure out the problem just by replaying the game frame-by-frame. Then they switched to the query system to filter based on their suspicions. Other users did the opposite, starting by querying and then switched to SeekWhence. This exploratory step was skipped by two users who decided to do a warm-up during the explanation of how the system works.

One fact that grabbed our attention was that five of our users could identify one inconsistency that we were not aware of. In the game *Alien* they could observe that a bomb was destroying a barrier in a position it was not designed to. Figure 6.3.

6.5.2 Praise

In general, the users agreed that Kwiri was a valuable resource for finding game design inconsistencies. Some of them pointed to personal experiences with situations in which they needed tools like the one presented here, but instead, they ended up hand-rolling their own.

One of the users explained that he had to write his system in order to evaluate how an AI was behaving.

> *"I liked these features. Some time ago, I had some issues with an AI that I was developing for playing a game. The tool I was using didn't have this feature about navigating and use visualizations. Then I had to write my own system to do exactly what we were doing here! Because here, I really can see what the agent is doing."*

Another user, similarly, does not have the appropriate tools to inspect his game projects.

> *"It is a really cool feature, pretty deal! I would like to use it on my work now, especially this frame navigating tool. But I'm having to implement everything from scratch."*

A third user revealed that his work consisted of play his game to figure what was causing the problems.

> *"I developed a game once, but I did not have how to record it and play it again as I did here. So every time that I found a bug, the only way to debug it was by playing it again. If I had a tool like that, at least my debug process would be way easier"*. Figure 6.4

6.5.3 Issues

Some users said they had problems in understanding the queries, for example, one of them stated that at some point it pops up too much information making the process not so attractive to follow. However, another user stated the opposite. In his speech, information volume does not seem to be a problem. The way that the UI leads to them is what concerns him.

> *"I really liked these tool, but I would like to see more information, however with less clicks."*

Still, about the query system, one user said that he was not sure about the roles of a sprite in an event.

Figure 6.4: User solving the inconsistency on the third task. By navigating on the gameplay session with SeekWhence and using the visualizations he could see the enemy walking through walls.

"I would like to know who is killing who in a 'killSprite' event. The query helped me to confirm my suspicion that an enemy was killing another, but the panel in the Who area should say who is the one doing the action.". Overall, as pointed in this section, the main issues were related to the query system. Design them as a filter tool does not seem enough. At least for the tasks evaluated, despite the majority of the users were able to use it correctly, it should provide details about the sprite roles in the game and be more explicit about the event as a user stated: *"I wasn't sure about what this event - transfrom to - mean. I was doing assumptions based on what I have seen before on other tools like GameMaker. Fortunately, it is similar, but I would like to have this information before."*

6.5.4 Suggestions

We got many suggestions from our users that we could add in future iterations of the system. One user said that he would like to see small ticks on the timeline bar. The ticks would be used to let a user know that one or more event is happening on that part of the gameplay session. To assure what he was suggesting, we used the Youtube [137] yellow advertise ticks as a design metaphor. They are used to explain to a user when an add will pop up. He promptly confirmed that it was exactly what he had in mind. Figure 6.5.

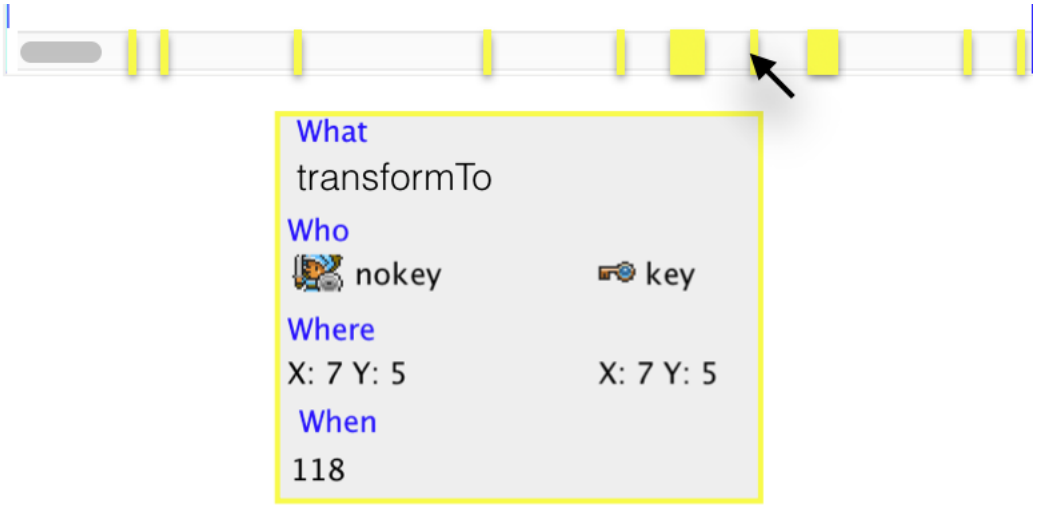


Figure 6.5: A quick prototype of a user suggestion. Yellow ticks on the timeline tells in which frames we have event(s) happening. By hovering the mouse over them we can see a preview of the event(s).

Another user suggested us to present the query options as a tree visualization where he could filter based on options available in each branch of the tree. It is an interesting suggestion, and we think it can reduce user efforts and present a better way to lead them to what they are trying to find.

Finally, another user said that the same color patterns used to represent the agents that does (green) and suffers (red) the actions should be available on the panel used to make the queries. He said that in general, the queries help when one has to identify who is involved in an event. However, it is also essential to know who started the cause and who is getting the consequences. It was also stated as an issue by another user and is something straightforward to fix in the query UI panel.

6.6 Conclusion

In this chapter, we introduced Kwiri, a system for querying game events in space and time. This tool is a new addition to Cicero. Therefore it is an expansion of it and encapsulates all the tools already presented so far. We evaluated our tool in the context of finding design inconsistencies/bugs. The users were able to solve their given tasks promptly and provided us with valuable suggestions; notably, they found bugs we were not even aware of, speaking to the usefulness of the system. To our surprise, the users were more attracted to the replay analysis system (SeekWhence) than to the query one (Kwiri) to solve their tasks. We want to design new tasks for evaluating each one of the features separately and in different kinds

of combinations. Besides finding inconsistencies, focus on agent evaluation and game balancing is a way of pushing forward the system's use. We planned a study with a broader audience of game designers and based on formal quantitative evaluations able to offer accuracy and statistical significance to our research questions — the findings of this study you can check in the following chapter.

Chapter 7

AI-assisted-game-debugging

In this chapter we discuss the design and evaluation of an AI-assisted-game-debugging in Cicero.

7.1 Pilot Study

We conducted a pilot study on two participants to evaluate our experimental design, estimate the time to complete each trial, test our desktop screen recording system, and judge the clarity of our tasks and explanations. Our first participant, a specialist in experimental design, suggested we create predefined software for each one of the tasks to save time during the experiment. The second pilot participant was a member of our lab, and the goal was to train the whole procedure and establish a balance between the time available for the user to play a game and how many times an AI should play (simulate) the same game. We also learned that in order to not make the experience exhausting to the users, it is better to present the experiment as a series of multiple tasks as we changed the parameters, rather than grouping every task together.

7.2 Subjects

After all the kinks were worked out of our experimental design, we recruited 32 students from our university to take part in the main study. They were, 10 female and 22 male. 5 were masters students, 10 were graduate students, and 17 were undergraduate students. All of them were enrolled in either the Computer Science or Digital Media departments. All subjects were recruited via their departments' email-lists. Those who completed the experiment were awarded a $10 USD Amazon Gift Card. Users took, on average, approximately 30 minutes

to complete the experiment.

7.3 Research setting

The research was conducted in an isolated, sound-proofed room. We experimented with a Mac Book Pro laptop (2.9 GHz, Intel Core i5, 8 GB RAM) with the screen mirrored on a Mac 27-inch monitor.

7.3.1 Users Tasks

We prepared two tasks involving features of Cicero to evaluate how accurate and quickly users could solve real problems.

In all the tasks we were testing the hypothesis: humans perform better in-game debugging tasks with AI assistance (game playing agents) rather than without it. The independent variables of this study were the games used for each one of the tasks and the players (humans or AI). The dependent variable was the accuracy in detecting inconsistencies. The AI agent player used was the *Adrienctx* agent, it is implemented based on an open loop approach [106] and it is a previous winner of the GVGAI competition [105]. Therefore, able to perform well in different game challenges, and as stated in the subsection "Agent-Based Test" it avoids, at some extent, disturbing characteristics common to non-human players like jitteriness and no long term planning.

7.3.1.1 Invincible Barriers: Human without AI assistance VS. Human with AI assistance

The first task was to find inconsistencies in the rules of our clone of the space-shooter game *Space Invaders*. There was only one inconsistency, barriers which could never be destroyed.

We divided the participants into two groups, which varied the order in which they received each task. Group A was asked first to find inconsistencies without any kind of AI assistance, but then they were allowed to watch the AI play the game to help identify the location of the indestructible barriers. Group B was assigned the same tasks in the opposite order. First, they completed the task using AI assistance and then by themselves. In both tasks, with and without AI assistance, after the subject (or their AI assistant) was done with their respective game session, we retrieved the last frame, and we asked the user to put a red mark on the barriers they thought were never destroyed. For all the cases, users were allowed to use SeekWhence.

Humans were given particular advantages that the AI alone did not have. For the AI, we ran the game just once. For the humans, we allowed them to play as many times as they wanted for up to 80 seconds (more than 3x the time needed for the AI to finish its simulation). For both players (AI and humans) we changed the positions of the barriers in the level. The total number of indestructible barriers was 9 out of 39. During this task, the users played two versions of *Space Invaders*, a normal one and an extreme one. In the extreme version, we changed the parameters of the game to make enemies drop bombs constantly and to make the human player immortal. Thus, we made it easier for humans to find the inconsistency in the game in two separate ways. For both game types, normal and extreme, humans have more time to play, and in the extreme version, they could not die. Figure 7.1.

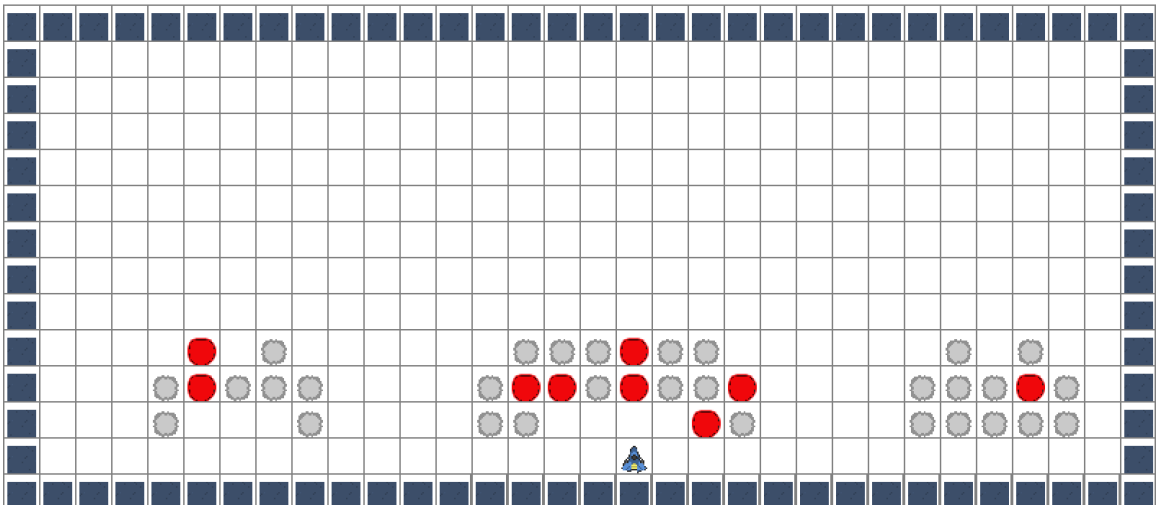


Figure 7.1: Example of a Space Invaders level. The barriers in red are the inconsistencies the users should find.

7.3.1.2 Query VS. Visualization

This task analyzes how accurate the users are in finding answers related to specific questions about the inconsistencies. For this case, they were not required to play, just analyze the result of an AI play-through. We allowed the AI to play the game for ten seconds, enough time to collect the necessary data for the task. We divided the participants into two groups. Group A started by using the query system and Group B by using the visualization system. For both cases, after we ran the game, we opened a form with true/false questions like “is any of the barriers destroying bombs?" or “Player shots can be destroyed by barriers?". For this task, we required users use only the system (query or visualization) being evaluated at the time, along with SeekWhence.

7.3.1.3 F-Walls: Human without AI assistance VS. Human with AI assistance

For this task, we altered a clone of the *Zelda* cave levels. We removed the enemies, and we planted fake walls (hence the name of this task: F-walls). The goal here was to identify how many fake walls subjects could find. We divided the participants into two groups, one started by playing by themselves while the other one started by watching an AI playing. After they or the AI finished, they were asked to put red marks on the level to indicate the location of the fake walls. The users had 80 seconds to play while the AI simulated the game five times taking, on average, less than 40 seconds in total. So that users did not rely on their memories, which would negatively affect their performance, they were allowed to use the visualization system or SeekWhence. The query system was not an option since it does not capture events without explicit interactions between two sprites.

7.3.2 Procedure

As soon as a participant came into the room, they were given consent forms explaining the research and the purposes of the experiment. After they signed the forms, we did a quick setup of the system by asking them to test the mouse and the keyboard. After that, we briefly explained the tasks and how to play the games. We made it explicit that in all the tasks, their goal was not to win the game, but to find inconsistencies in the game's implementation. We also explained to the subjects that they were not obliged to participate in the study, and they had the option to drop out at any time. For the first (*Space Invaders*) and the third (*F-Walls*) task, we explained the game and the inconsistency they need to find. We ran the specific tasks assigned for their group, and then we asked them to put a red mark on the inconsistencies they have found.

For the second task (query vs. visualization), we explained to the users that we would have the AI play the games for ten seconds. Afterward, we would open a panel (the query or the visualization) and ask them to fill out a form based on what the panel was showing to them. Figure 7.2.

7.4 Results

The results are divided into three sections, each related to one of the tasks presented in the previous section.

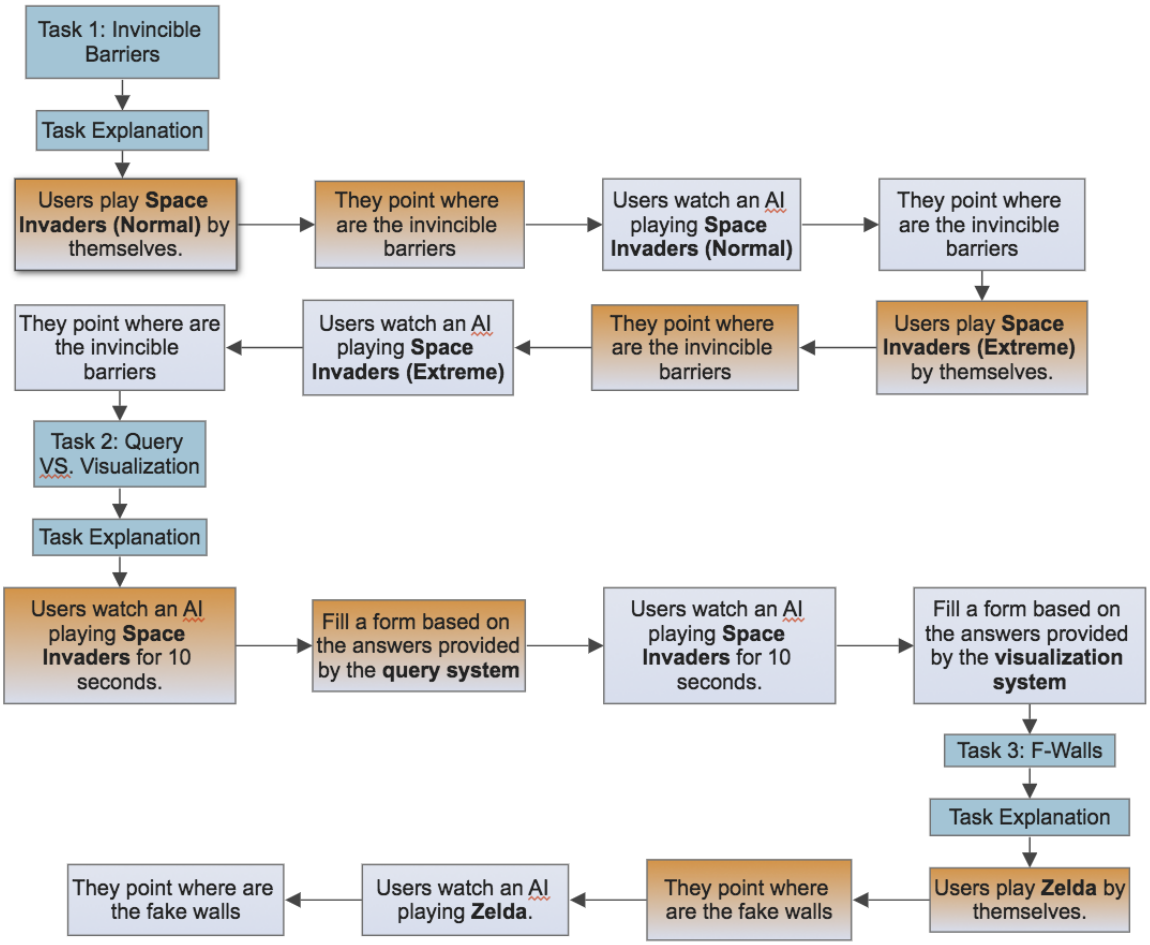


Figure 7.2: This flowchart shows the procedure taken by users of the group A. They started the first and third tasks without AI assistance and started the second task by using the query system. By switching the orange blocks with the gray ones, we have the procedure taken by the group B.

7.4.1 Task 1: Space Invaders

In this task, we investigate how human performance in-game debugging tasks is affected by AI assistance.

7.4.1.1 Normal Case

To evaluate if humans with AI assistance were performing better than humans without assistance, a paired samples t-test (one-tailed) was used to compare the scores of the groups. The result was statistically significant ($p = 1.002e - 05$), and the AI assistance improved the accuracy of the users by almost three times (2.7x). AI assistance also helped users avoid mistakes. The mean showed that while users without AI assistance were making 1.75 wrong

choices, users with AI assistance made none. This result was significant with $p = 0.0038$. We also ran precision and recall tests for both groups. The results for the tasks with AI assistance show that it improves both precision (100%) and recall (~82.29%) over the tasks without AI assistance (precision: ~83.14%, recall: ~52.08%).

7.4.1.2 Extreme Case

We did not achieve statistical significance for the AIs ability to help humans in the extreme variant of Space-Invaders. The results for humans with and without AI assistance were very close, with the AI only offering a mean improvement of 0.25 additional correct responses. There was also a mean reduction of 0.25 erroneous responses with AI assistance. The similarities were reinforced when we ran the precision and recall tests. While the group with AI assistance got precision and recall of, respectively, 100% and ~93.05%, the second group got ~97.81% and ~90.28%.

7.4.2 Task 2: Query VS. Visualization

For this task, we investigated if users can be more accurate in-game debugging tasks if they use a query system rather than a visualization system. McNemar's test was used to compare the data in this task. The result was statistically significant (p-value = 0.0047) and showed that users were more accurate with the query system. Their answers for the two questions were correct ~96.77% and ~87.09% of the times. While with the visualization system, the accuracy was of ~64.51% and ~80.64%.

7.4.3 Task 3: F-walls

Here, we also investigate if a human performance in game debugging tasks is better with AI assistance than without. No statistically significant difference were found. Humans with and without AI assistance could found similar number of fake walls with a mean difference less than 0.24. The same happens for the number of mistakes committed, which has a mean difference less than 0.3. The recall result was of ~87.5% for humans without AI assistance and of ~81.66% for humans with AI assistance. While the precision result was of ~96% for humans without AI assistance and of ~78% for humans with AI assistance.

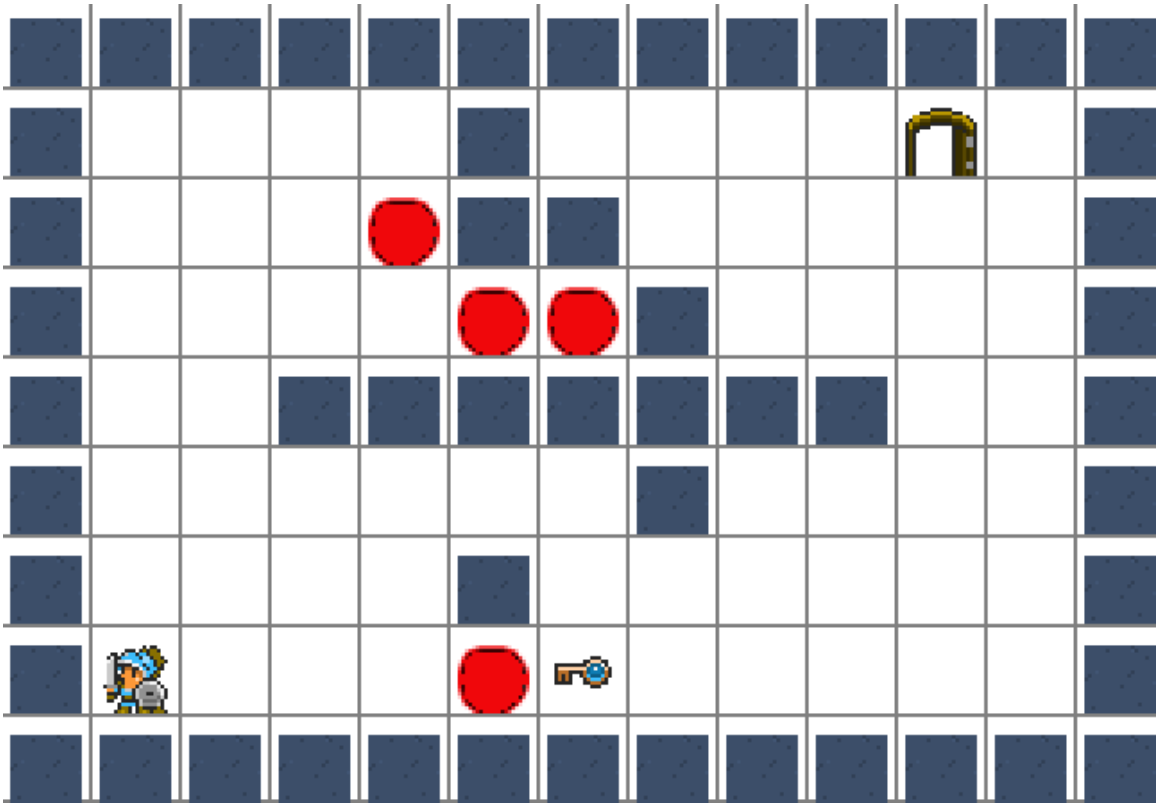


Figure 7.3: Example of a Zelda level. In red, the positions of the fake walls that the user should find.

7.5 Discussing limitations

This work dealt with phenomenological events for building a debugging assistant. This means that the debugger is designed based on game events contained into the game description. As the agents used were designed to win the games instead of looking for inconsistencies, it can get to a point where it will ignore the event exploration to win the game as soon as possible. It will force the designer to run the experiment again and cheer for the agent to try the events the designer expect it to try. A solution would be change the level or the description of the game to force the agent to try the events the designer has in mind. But it goes against the idea of having intelligent agents with the purpose of reducing user workload. Another approach is the combination of methods to support the agent work and provide more accurate diagnostics. The use of a visualization tool can show areas in the map never visited by the agent. The designer can then inspect and figure out if the reason is because of a flaw. It is still not ideal, but it helps more than inspect all the areas by playing the game (exhaustively). Also, human-like agents [70] and agents based on personas [25] have the potential to bring new insights both about the results of the

debug sessions as the way they are designed as well. Another method that can improve the accuracy of the results pointed by the agent is functional debugging. An interface that allows the designer to verify input/output parameters can check whether the game is working as expected. Such configuration can be part or leading the design of an agent that can explore non-phenomenological issues as well. Actually, in a time where the games are designed part by humans, part by AI-algorithms, hybrid (human-AI) debugging solutions should arise.

7.6 Conclusion

The results of the first task (identifying invincible barriers with/without AI) confirmed our expectations that humans perform more accurately with AI assistance than without. For the extreme version of Space-Invaders, by giving advantages to the users (they could not be killed), they were able to perform comparably to users with AI assistance. While this seems a viable way to make debugging easier, we argue this is not the most practical method. If one knows exactly which game parameters can be tweaked to exacerbate bugs, then its likely the game developer has a good idea of where the bug exists already. In the absence of such an accurate hunch, game developers are left having to tweak every parameter in the game until a bug becomes apparent. This is neither easy nor efficient.

The second task (query vs. visualization) also confirmed our expectations that the query system can be better than the visualization for games that can use both techniques. Users spent more time with the visualization system, and they had to use SeekWhence in order to find the right frame of the game that could answer their question. On the other hand, when approached using the query system, these tasks were solved almost instantaneously. The query system shows a panel that displays exactly the answer to the question entered and (if told to) can jump straight to the frame in the replay sequence where that action happens.

The third task (finding fake walls with/without AI) shows that humans alone and humans with AI assistance perform almost equally. This task also showed that humans without AI assistance were slightly more precise than humans with it. The main point of this task is that the game in question does not require dexterity like space invaders. Humans can easily design and execute an efficient bug-finding strategy of bumping every wall to find the fake ones. Whereas in space invaders, subjects had concerns like bombs, barriers, enemies, life, shots, etc., in Zelda, the user was left undistracted to hunt for bugs. While even in Zelda the AI will explore all the possibilities of winning the game, and eventually will find the fake walls, this can be a time-consuming process, and the AI is not guaranteed to touch every wall.

Despite many of the results confirmed our expectations, more tests are necessary in

order to achieve statistical significance for all the cases explored in this study. Concerning to efficiency, we suggest more work on agent customization. The dexterity presented by the agents evaluated in this paper is enough for playing games like Space Invaders in capabilities that overcome humans. However, for finding inconsistencies in levels of a bigger size and complexity, it would be necessary thousands (or more) simulations. Still, because the goal of the agent is beating the game, even with a considerable amount of simulations there are no guarantees it would cover all the cases and finding all the inconsistencies expected by the developers. Therefore we encourage more research on designing general agents oriented to find game design flaws. One way to do it, specially for the case of the *fake walls*, would be the use of an imitation learning agent. It could copy the strategy of a human game tester and replicate it thoroughly the whole game.

Chapter 8

Recommender System for Game Mechanics I

8.1 Introduction

This chapter presents our first attempt in designing a recommender system for games. As previously explained, our system (Cicero) is based on VGDL. Therefore, to build a recommender system out of it, it is necessary to pre-process the language in a way it can be interpreted by algorithms able to use the game content data and provide suggestions. The following sections bring details of our first attempt at building a recommender system for game design. We will often talk about VGDL in this chapter. While we took care to be as didactic as possible, it is hard to guarantee that we covered all the details of the language and how we embed it on our approaches. However, because it is not an extensive one, we suggest the reader go to the chapter 2 and take a look at the section **VGDL language and GVGAI framework** whenever it feels necessary.

8.2 System core functionalities

The core of our recommender system consists of four simple comparisons that allow for a higher level search abstraction. These comparisons involve sprites and interactions, and are described as follows:

- *sprite type check*: verifies that the type of the two sprites compared is the same. If so it returns 1, otherwise 0;
- *sprite similarity*: checks if the sprite types are the same, if so it computes the normalized vector distance between the parameters vectors, and subtracts the result from

a *maximum pre-defined value*. Otherwise it returns a *minimum pre-defined value*. The parameters can either be floating point numbers or sprite name. In the former, it performs a Euclidean distance. In the latter, it uses the *sprite type check* comparison on the sprites referenced;

- *interaction similarity*: verifies that the sprite types involved in two interactions are the same, if not it returns the *minimum pre-defined value* otherwise it subtracts the normalized vector distance between the parameters vectors from the *maximum pre-defined value*. The distance is computed as in the *sprite similarity* case;
- *interaction fitness in sprite set*: measures how much an interaction is suitable for a set of sprites;

From the point of view of the recommendation system, we are interested in performing the following comparisons: 1) game similarity measurement; and 2) sprite fitness in a game.

8.2.0.1 Game similarity measurement

A high-level comparison can be done by measuring the similarity between the games. The quality of the recommendation improves as the similarity between games increases, and vice-versa. This is an asymmetric comparison between two games, a reference game, and the compared game.

The similarity measurement is a two-fold process. For each sprite in the reference game, we select a similar sprite in the compared game. Then we pick the highest interaction set similarity measurements between the sprites previously selected. Those values are summed and averaged on the number of sprites in the reference game.

Algorithm 1 depicts the pseudo-code for the above-mentioned comparison.

In this way the system can measure how each sprite in the reference game is represented in the compared game's mechanics.

8.2.0.2 Sprite fitness in a game

This high-level comparison tries to understand how well a sprite and its interactions can fit in a game. The system iterates through the sprite's interactions and sums the *sprite similarity* metric between the sprites in the interaction and the sprites in the game we are trying to fit the sprite into. The result is then averaged by the number of sprites in the game. See the pseudo-code in Algorithm 2.

Algorithm 1 Game Similarity

procedure MEASUREGAMESIMILARITY(Game ref, Game $other$)
  $similarity \leftarrow 0$
  for all $Sprite\ s1 \in ref.SpriteSet$ **do**
    $bestMatch \leftarrow 0$
    for all $Sprite\ s2 \in other.SpriteSet$ **do**
      if SpriteSimilarity($s1, s2$) >$threshold$ **then**
        $currentMatch \leftarrow$InteractionSetSimilarity($s1.InteractionSet, s2.InteractionSet$)
        $bestMatch =$max($bestMatch, currentMatch$)
    $similarity \leftarrow similarity + bestMatch/ref.SpriteSet.Size()$
  return $similarity$

Algorithm 2 Sprite fitness in game

procedure SPRITEFITNESSINGAME(Sprite $sToFit$, Game g)
  $fitness \leftarrow 0$
  for all $Interaction\ i \in sToFit.InteractionSet$ **do**
    $other \leftarrow i.OtherSprite(sToFit)$
    for all $Sprite\ s \in g.SpriteSet$ **do**
      $fitness \leftarrow fitness+$SpriteSimilarity($other, s$)
  return $fitness$ / $g.SpriteSet.Size()$

8.2.1 Sprite Suggestions

The system can provide a set of suggestions which are selected after analyzing the game library using the core functions. Each suggestion is a sprite that comes with its own interactions extracted from the game in which the sprite is declared. Thus the user can choose a sprite in the set of suggestions taking all, some, or none of the related interactions. The suggestion set is obtained by applying some selection policies to a vector of sprites, which is ordered according to the system's metrics.

Currently, the system can provide suggestions based on two ranking types: 1) item based ranking; 2) user based ranking.

8.2.1.1 Item based ranking

This ranking measures the fitness of all the sprites contained in the game library with respect to the game currently being edited by the user. It uses the *sprite fitness in game* metric to rank all the sprites in the library. Then, it applies the selection policies to filter out the most interesting results. The first selection policy takes the sprite with the highest fitness value, in order to exploit the current game's configuration and to suggest the sprite that is more

suitable for it. On the other hand, the second selection policy picks the sprite with the lowest fitness, aiming at expanding the current game mechanics.

8.2.1.2 User based ranking

The user based ranking consists of two steps, and treats each game as a user, trying to analyze the user's decisions. So a game from the library is seen as a user that has selected a specific set of game elements in the past. In the first step, games are ranked according to the *game similarity measurement*, so to understand how similar the game design editing is, the system compares with design decisions taken in other games. Afterwards, the selection policies pick the worst and best ranked games: this translates in selecting the most similar and most different user. In the second stage, the sprites from those games are ranked using the *sprite fitness in game* metric. At this point, the selection policies choose the best and worst sprites. What the system tries to do is, on one hand, to select the most appealing design decision taken by someone else in other games; while on the other hand, to select the least expected one, so as to introduce novel design decisions to the game.

8.3 The game mechanics recommender graphical user interface

This section describes the system's user interface, and provides a user case where the system is used to generate a game based on suggestions.

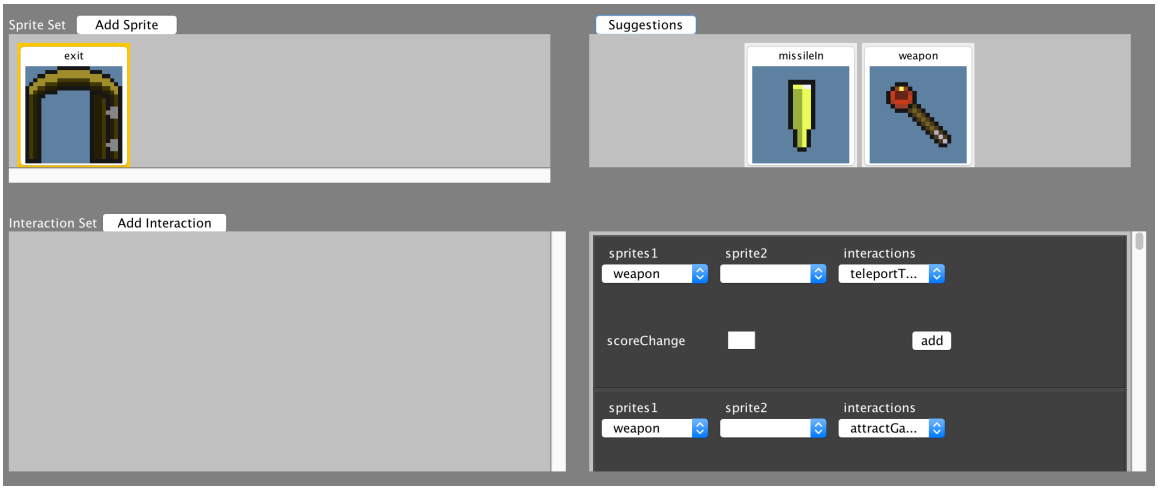


Figure 8.1: The game mechanics recommender's main interface. On the left, the panel has buttons to add sprites and interactions. On the right, the panel displays the sprites and interactions suggested.

The main interface of the game mechanics recommender consists of two group of panels: one for adding sprites and interactions by pressing the “Add Sprite” and “Add Interaction” buttons respectively; the other group shows the suggestions of sprites and interactions whenever the user presses the “Suggestions” button (see Figure 8.1).

When the user hits the “Add Sprite” button, a panel with empty fields appears. These fields can be used to specify the sprites parameters, such as speed, orientation, and type. In the same way, when a user hits the “Add Interaction” button, a panel with empty fields appears to specify the parameters for the interaction. Every time the user requests a suggestion, the current game is compared with contents from other games in the VGDL repository. When the system finds a game with similar content, it retrieves and suggests sprites and interactions.

When the user accepts a suggestion, an edit panel appears as a pop-up. It shows the fields already filled. However, it is up to the user to accept the suggestion as it is or change parameters according to her convenience.

In this description, a simple game is generated according to the user based search to balance the choices between the best and the worst suggestions. The game’s goal is straightforward: the avatar needs to shoot at walls in order to reach the exit. It contains one sprite provided by the user, and two others suggested by the system. First, we accept the best sprite suggested without performing any change. For the second, we pick up the worst sprite. This way we balanced the suggestions in order to try a game description as dissimilar as possible to the ones already available. All the interactions are suggested by the system but with some adjustments. The game design is created after suggestions, and the game is designed without any previous concept in mind.

Figure 8.2 shows the main interface, with the final sample configuration and the game running on a simple level.

Figure 8.2: The final sample game configuration (left) and it running (right)

8.4 Conclusion

n this chapter, we presented our first attempt in designing a recommender system for games. We based this first attempt on Euler distances by describing game elements and its parameters as vectors and then computing their distances. A distance rank was used to define what would be recommended. In the end, despite this approach showed us the idea of a recommender system for games would be something attainable we were not satisfied. The results repeated themselves often. The impression was that the algorithm was always suggesting the same things. The graphical user interface was inexpressive, and in the end, we could not evaluate our system correctly with users. In the next chapters, we will show what we have done to improve our recommender system.

Chapter 9

Recommender System for Game Mechanics II - Pitako

After the first results of our first attempt in design a recommender system for game design, we decided that we needed to use other approach and restart everything from scratch, from the way we were interpreting the data to the way it should be presented on screen. In this chapter we will refer to the GVGAI framework and the VGDL language several times, so the same suggestion for the last chapter is valid for this one too - feel free to to go to the chapter 2 and take a look at the section **VGDL language and GVGAI framework** whenever it feels necessarily.

9.1 Data Interpretation

We decided to take a look at the data we had available. As explained before, everything we have done is based on the GVGAI framework and the VGDL language. The first thing to note is where we can find the core mechanics of a game in such language. In VGDL they are distributed in two sets: the **Sprite Set** and the **Interaction Set**, See Figure 9.1. Therefore, the next step was to identify what methods we could use to interpret the data in the two sets in a way that we could recognize patterns and create a recommendation list out of it.

9.2 Frequent Itemset Data Mining Applied to VGDL games

As stated before, a sprite in VGDL is not only an image. It is the description of behavior onto which you can attach an image. If you are implementing the hero of a shooter game like *Mega Man X (Capcom, 1991)* or *Contra 3 (Konami, 1992)*, you would specify the

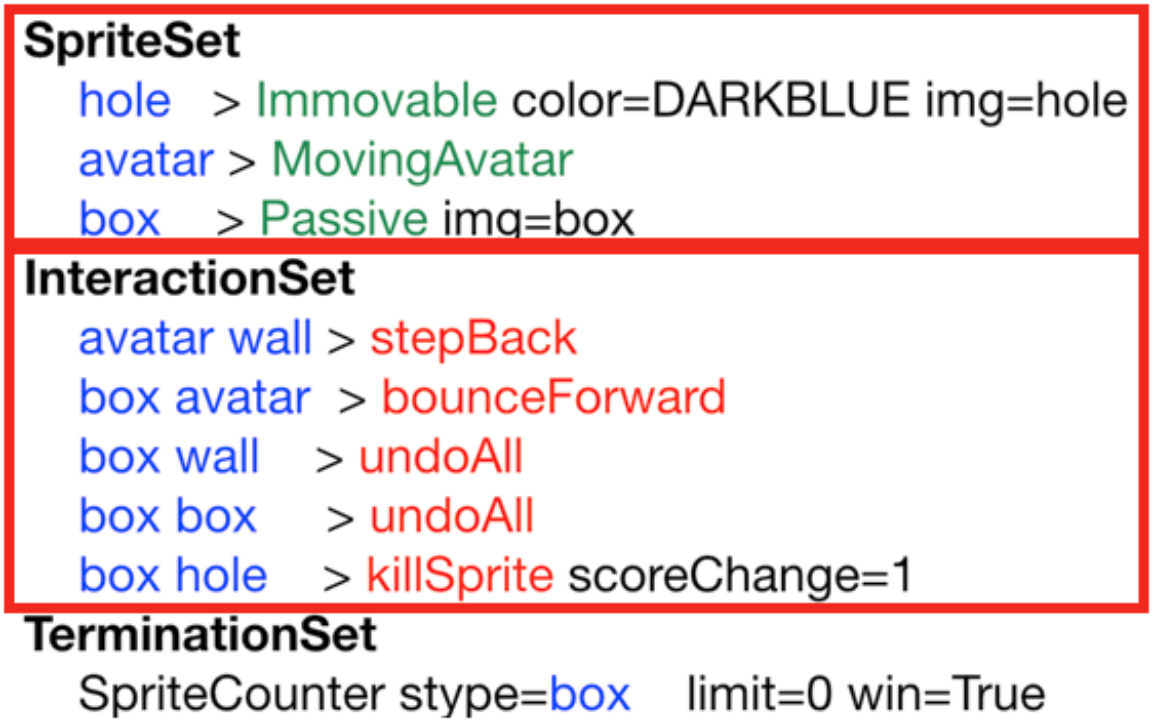


Figure 9.1: This picture brings the example of a VGDL Sokoban version. We highlighted the two sets which represents the core mechanics of a game. The **Sprite Set** and the **Interaction Set**. The idea is to interpret this data in way we can find a set of patterns in different game descriptions.

sprite behavior as a ShootAvatar, and then attach the image of your hero to the defined sprite. Therefore, the set of sprites is the first source of rules in a VGDL game and a natural candidate to be automatically recommended during AI-assisted game design. As presented in the 3, from the family of frequent itemset data mining algorithms, we took the Apriori. It is suitable for our needs of finding patterns and reduce a list of elements to a rank of frequent and infrequent ones.

9.3 Recommending Game Elements Foundations

Although some of the ideas we are about to discuss here were already presented in our first attempt of design a recommender system for games, we expanded the whole concept. Because it had influenced the recommender system redesign, it is important to outline our foundations in this section.

9.3.1 The atoms of a game

Digital games as we know them today, are a creation of the twentieth century. As every invention of the 1900s, the design of the digital games was influenced by the three main scientific discoveries that started to change the world more than a hundred years ago: the atom, the gene, and the byte (or bit) [119]. Conceptually speaking, these three discoveries brought the evidence of the unit, to the indivisible and smallest part of a system as the main component to understand the whole.

From the game literature, we can see that this idea is widely shared. For example, minimalist game design [93] advocates a development practice in which you start your new game project from its core game mechanics. Everything else like graphics, sounds and even new mechanics (derived from the basics one or not) comes after. In the book written by Brenda Brathwaite and Ian Schreiber [17], a best seller in the game design literature, you can find a whole chapter entitled "game design atoms", another example of the atomic principle applied to game development. The authors claim that “by looking at a game as a collection of atoms (mechanics, dynamics, goals, game state, etc.), the process of design itself becomes clearer". A similar approach is put forth in another best seller, written by Paul Schuytema [114]. The MDA - Mechanics, Dynamics, and Aesthetics [62] also brings the same concept of building games starting by its necessary components, and evolve them until the level of creating the art content(aesthetics). For the purpose of our tool, and by following the MDA principles, we are recommending mechanics and dynamics, only, no aesthetics recommendation is provided at this point.

9.3.2 Game design breakdown

Before recommending fundamental components for the design of a new game, we must first develop a comprehensive understanding of similar games, from which to draw patterns to be recommended. We call the process of understanding the fundamental components of similar games by "game design breakdown." It consists of getting a formal game description and rewriting it in a simplified way to have access to its minimal (atomic) elements. The process’ name comes from a similar process in the movie industry, the script breakdown [121], in which the crew read the movie screenplay page by page and define all the minimal technical aspects necessary to shoot the movie. To keep using our atomic metaphor, we took the individual sprite and sprite-interaction descriptions as our atomic elements. Everything they bring with them (their parameters) are the subatomic elements. It allows us to build a data set where we can map games to the elements they contain, and sub-elements to their parent elements.

9.3.3 The catalog

To facilitate the process of knowing, recommending and combine features from different games to design new ones, we took inspiration from chemical and biological methods. For example, the geneticist Thomas Morgan designed a catalog of fruit flies in which he described their minimal features and tracked every mutation to facilitate how to combine them and come up with modified species [92]. Also, the chemical periodic table of elements was designed to make easier the process of synthesizing chemical structures. Therefore we built a catalog of games (Figure 9.2), also based on their minimal components with the intention to inspect their individualities and mix them to generate new games.

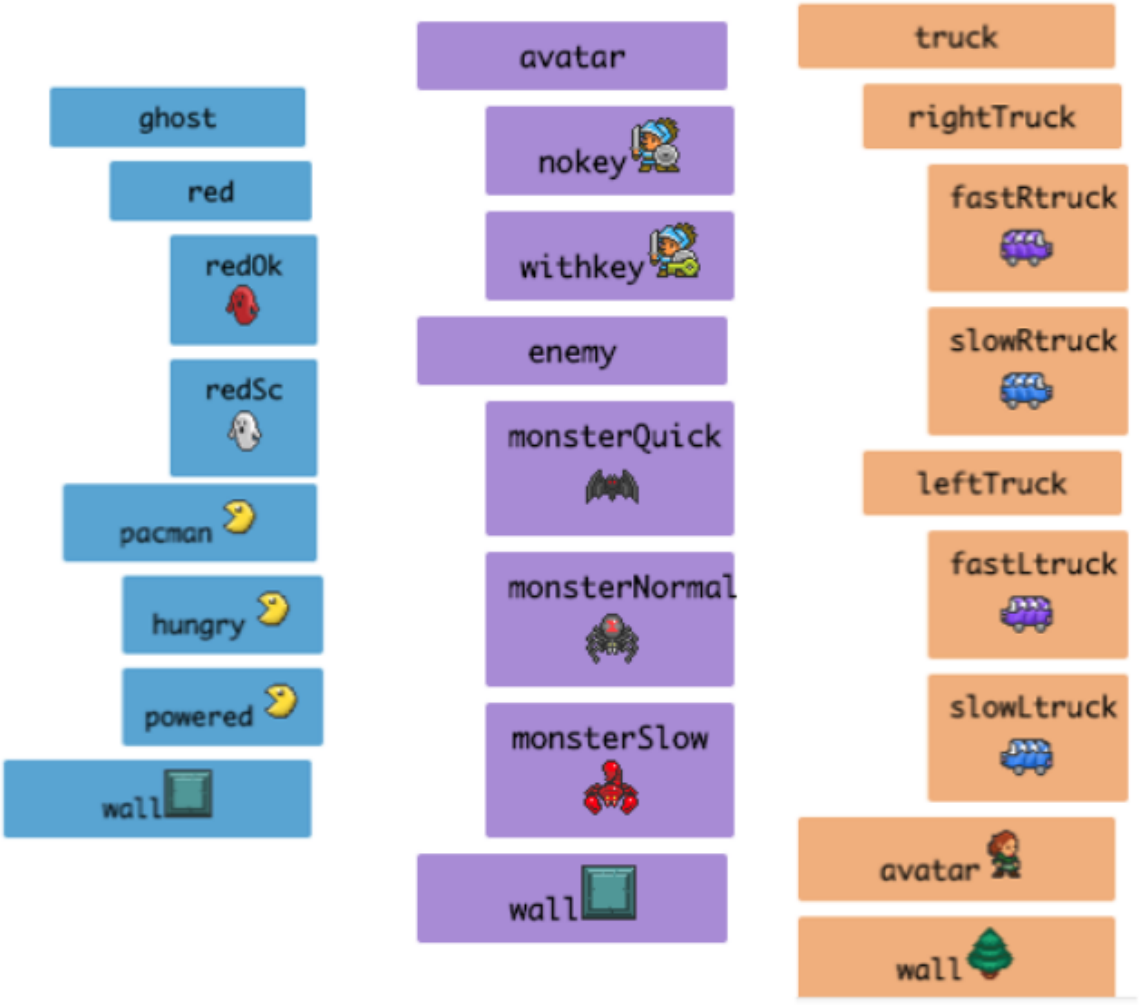


Figure 9.2: From left to right, sprite sets of the games *Pacman (Namco & Atari, 1980)*, *Zelda (Nintendo)* and *Frogger (Konami, 1981)*. By arranging their elements into a data catalog, the search process of finding an element and combine it with one (s) from other games, becomes easy and straightforward.

To use the apriori algorithm to provide sprite suggestions the first step is collecting every sprite in every known game in our catalog (Figure 9.3). In the end, we have a table where each game represents a transaction set, with the sprite behaviors as the items of the transactions.

Figure 9.3: At the left side, the red square shows the **SpriteSet** of the VGDL version of the game Zelda. In the Center, it is exhibited a fragment of the SpriteSet converted to a JSON format. This format makes easier the process of navigating into the games and extract information from them. The **referenceClass** attribute that you can see in all the sprites listed in the JSON description (center figure) defines the behaviour of the sprite. At the right side you have a visualization of the whole SpriteSet. Every game in VGDL is converted to this format, and then, information are extracted, shared, and intertwined when the recommender system is applied to design games.

The final table is, therefore, the algorithm input. After running apriori, we get a table with the item sets whose frequency of occurrences reach the limit of the minimum support provided. Therefore, the table shows us how frequent sprite behaviors are associated with each other in different games.

9.3.3.1 Sprite Recommendation protocol

With knowledge of the association between different sprites in different games, we can then provide suggestions to the designer. The input for our recommendation protocol is the sprite set of the game in development. This set is compared against the set of association rules. All the associations which have the sprite set as a subset are stored in a list. At this point, the association rules give us the sprites candidates to be recommended (Figure 9.4). We remove the ones already presented in the user's game, otherwise, we would give irrelevant recommendations. Which the sprite candidates pointed by the association rules, we go to the catalog and find which games have the same sprites. We then do copies of these sprites and suggest them to be used in the user's game (Figure 9.5). Note that a sprite cannot be suggested directly from an association, because it is necessary to adapt contextual information from the game which it is recommended, and the one who is getting the sprite. We talk more about this process in the following section.

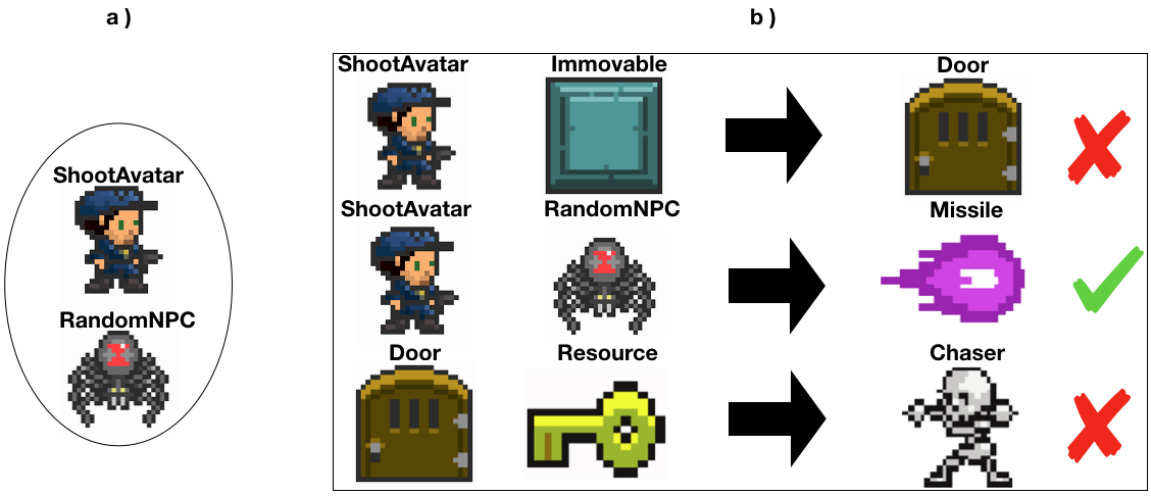


Figure 9.4: In this figure (a) is the user's game's sprite set. In (b) we look at the association set, to find an association rule related to (a). The selected association rule, will give us the candidate sprite to be suggested. We then, go to the games in the catalog to find one with the given sprite. Finally we copy and recommend it to user.

9.4 Blending Game Process

Once the user has selected which sprite to add to the game being developed, we start the procedure to merge the features and behaviors of the sprite into the game; we call this by the blending process, and we are inspired by the logical blend methods present in the work by Eppe et al. [45]. We are extracting information from one game and inserting it

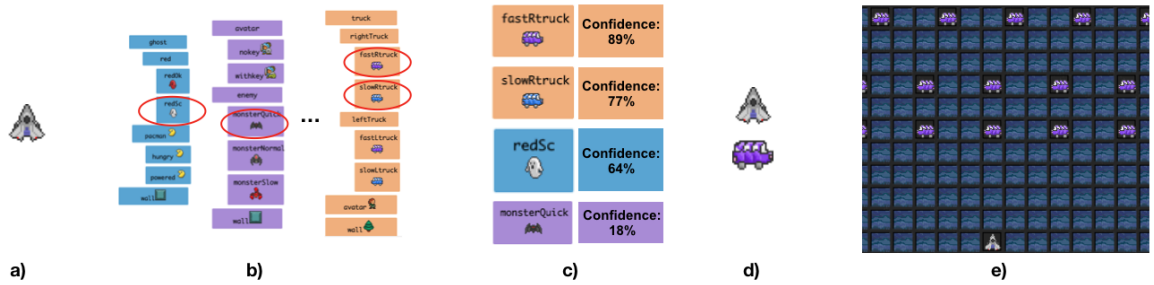


Figure 9.5: The recommendation process starts by getting the designer game description set. In **a)** the user has only one game element. In **b)**, the system is performing a frequent itemset data mining algorithm to identify association(s) that matches the element in the designer's game to elements in the games' catalog. The red ellipses represent elements from different games which are candidates for being recommended. This procedure yields a list of recommendations sorted in descending order by the rule's confidence level **c**. The designer picks the recommendation with the highest confidence level **d**. Then, starts the to use it as a new element to the game. This element came from a clone of the famous game *Frogger*. And in this example, the user is mixing elements from *Frogger* and *Space Invaders*.

into another one. As sprites can have other sprites attached to it, we need to make sure that it will be imported not individually, but with all the other sprites it refers. We use this approach to reduce user workload. For example, if a designer selects a shooter character, the recommender will import the character and the object it uses to shoot. Otherwise, the designer would have the shooter character but would have the extra work to design an object to be shot. In order to create the sprite package to be imported, we use a depth-first search tree navigation to navigate in the game catalog entry recursively so that we can get the sprite and any other it refers to. It captures the context of a game and let it logically prepared to be inserted into another one. Once the package is done, it is imported and inserted in the sprite set of the game in development.

9.5 Apriori algorithm applied to the sprite placement recommendations

Once we have the user decision about the imported sprite, we can suggest positions in the game level map to put the recommended sprite. This process uses a combination of heuristics and the apriori algorithm.

The recommended sprite comes from a single previously designed game. In the VGDL game repository, each game has five levels. The input table for the apriori algorithm is generated by picking the recommended sprite type. The Sprite positions are then stored

and indexed by level. So each level of the game is an association set of the positions of a specific sprite. The positions are the items of the association. Then, after we run the apriori algorithm, the output is a set of the most frequent positions for a specific sprite.

9.5.1 Heuristics for placing sprites in a level

After getting the output of the apriori algorithm with the positions to suggest, we apply filters to provide better recommendations to the user. The use of game level design heuristics is useful for this task.

- **Items should be at a K distance of the avatar**
  We identified that the avatar should begin each level somewhat isolated from other sprites. For example, it is often undesirable to place a sprite adjacent to the avatar as the two sprites may collide on the first timestep, potentially harming or destroying the avatar as soon as the game starts. If the recommended sprite is a collectible item or the final goal of the level, placing it too close to the avatar will afford no challenge in retrieving it. By contrary, sprites which are harmful to the avatar are suggested to be placed close to resource items and exit level doors. The interaction set provides us the context to understand which sprites can be harmful to the avatar player.

- **Ensure that the items will be placed inside the level boundaries**
  As the level being developed can have different dimensions from the level of other games in the catalog, positions of a specific sprite type that cannot fit in the user's level boundaries are not suggested.

After applying the heuristics and filtering the output of the apriori algorithm output (see Figure 9.6 on the following page), the placement suggestions can then be recommended.

9.6 Recommending interaction rules

In VGDL, an interaction is composed of two sprites and the interaction they activate when colliding. Therefore to recommend interactions, first it is necessary to do a map of combinations between all the sprite types (the key) and all the kind of interactions (the value(s)) they can fire when a collision event happens. Secondly, combinations among the elements of the sprite set in development are generated. Finally, a search identifies which combinations in the user's game has an interaction that can be used as a suggestion.

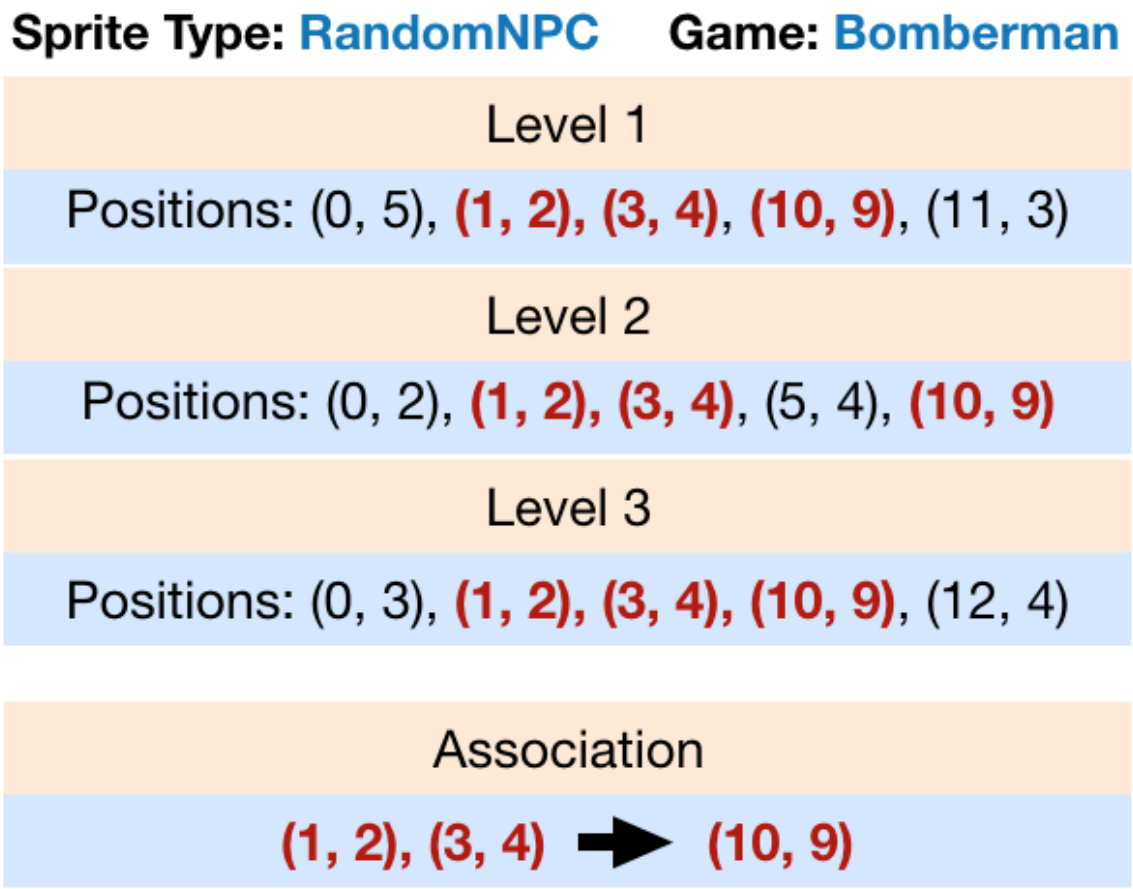


Figure 9.6: For the sprite type **RandomNPC** in the game **Bomberman**, its positions in the three first levels are shown in this figure. The association rule is filtering according the heuristics and then suggested to the user.

9.6.1 Sprite combination map

To generate all combinations of sprites, it is necessary to navigate through the interaction set of all the games in the catalog, build a pair of sprite types and the interaction that is activated when they overlap each other. It is important to note that pairs of sprites with the same type are valid since often two sprites of the same type can interact with each other. The end of this process is a map whose key is the pair of sprites, and the value is a list of the interactions activated by them. Because the list has repeated values, we apply the same confidence function from the apriori algorithm to calculate the interactions most likely to be suggested.

9.6.2 Sprite pairs comparison

We can only recommend interactions for pairs of sprites available in the sprite set of the game in development. Therefore, after combining these sprites, we compare each one of them against the map keys. When we found an entry whose pair (key) is equal to the pair in the user's game, we return the key value, a list of interactions. These are the ones that will be suggested to the user. As opposed to the sprite recommendations, interaction ones are straightforward. Therefore, there is no necessity of performing additional navigation in the game catalog entry, and once the user decides for picking one of the recommendations, it is automatically inserted in the user's game interaction set (and removed from the recommendation set).

9.7 Recommender System UI

For the user interface design of our system, we divided it in three different Graphical User Interfaces (GUIs), one for the sprite set, another for the interaction set, and finally the level layout design to show the recommended positions of where to place the sprites. The UI was also redesigned from scratch. On our first attempt, the final result was inflexible, consequently, it did not allow expressiveness and a proper way to establish good communication with the user. A big part of these issues is related to technical limitations. The first attempt was implemented with languages (Java) not so suitable for designing dynamic UIs. After the first experience and the changes we did in the whole project, we adopted HTML5/CSS/JavaScript for helping us to prototype a UI suitable for conveying our ideas.

9.7.1 Sprite Recommender UI

The sprite recommendations appear on the screen as a list sorted in decreasing order by the confidence of the recommendation. Each list object has information about the game from where the recommendation is coming, the confidence value of the recommendation, the type of the sprite being recommended, and its image (Figure 9.7). By hovering over the image, a pop up shows a list of the sprite's attributes (and their values). We added this extra information to help the user to decide what to pick when having two or more sprites from the same type being recommended. When the user selects a recommended sprite, it is imported to the game sprite set. In the background, the system performs a search in the catalog, when finding the game whose specific sprite is recommended, it extracts all the information from the sprite and others it may carry. For example, if a user is picking a sprite which can shoot, the sprite that represents the projectile behavior is imported as well.

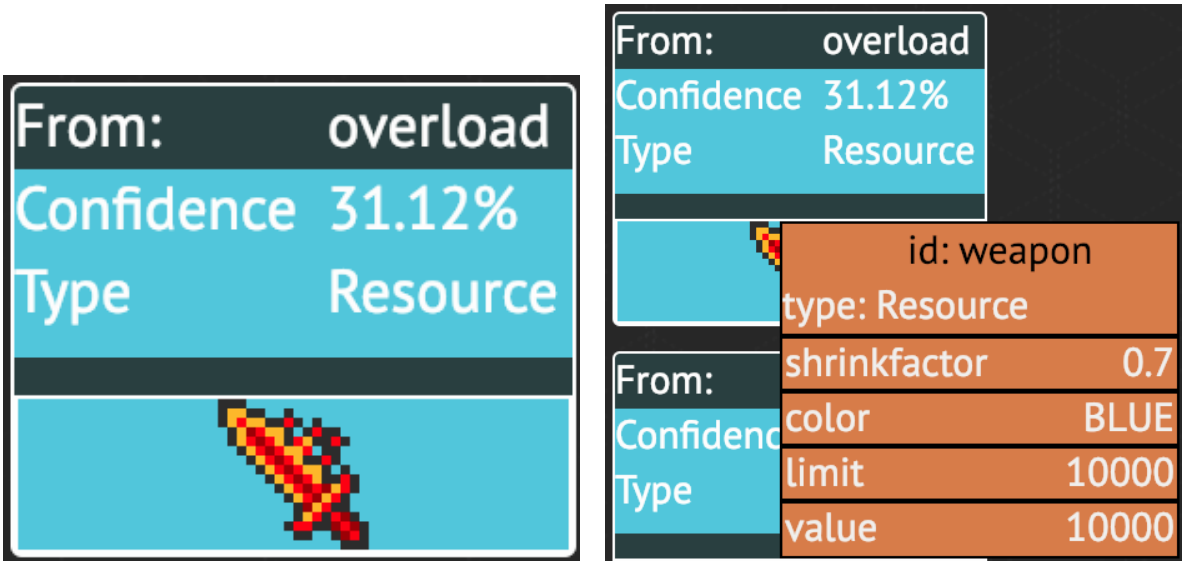


Figure 9.7: The sprite recommendation component displays which game the sprite is imported from, the confidence of the recommendation, and its type (left figure). By hovering over it, the user can see other attributes and values of the sprite (right figure).

9.7.2 Sprite Placement UI

The sprite placement suggestion happens whenever a user is adding a sprite in the level grid layout (Figure 9.8). The process of placing a sprite in the grid is a simple drag-and-drop interaction. Therefore, as soon as the user starts to drag the sprite, green circles highlights the grid square positions recommended to receive the sprite.

9.7.3 Interaction Recommendation UI

Similar to the sprite recommendations, the interaction recommendations (Figure 9.9) shows up as a list sorted by confidence in decreasing order. As the number of interaction suggestions tends to be high, the users can sort the list based on the elements they want to put to activate a specific interaction. When the user selects one of the recommendations, it is added to the interaction set. As mentioned before, interactions are plain descriptions of what happens in the game when a sprite A overlaps a sprite B. Therefore, this is a straightforward import process, and extra search steps are not necessary because interactions do not have attributes which are other interactions (like a sprite having another sprite on its attribute list).

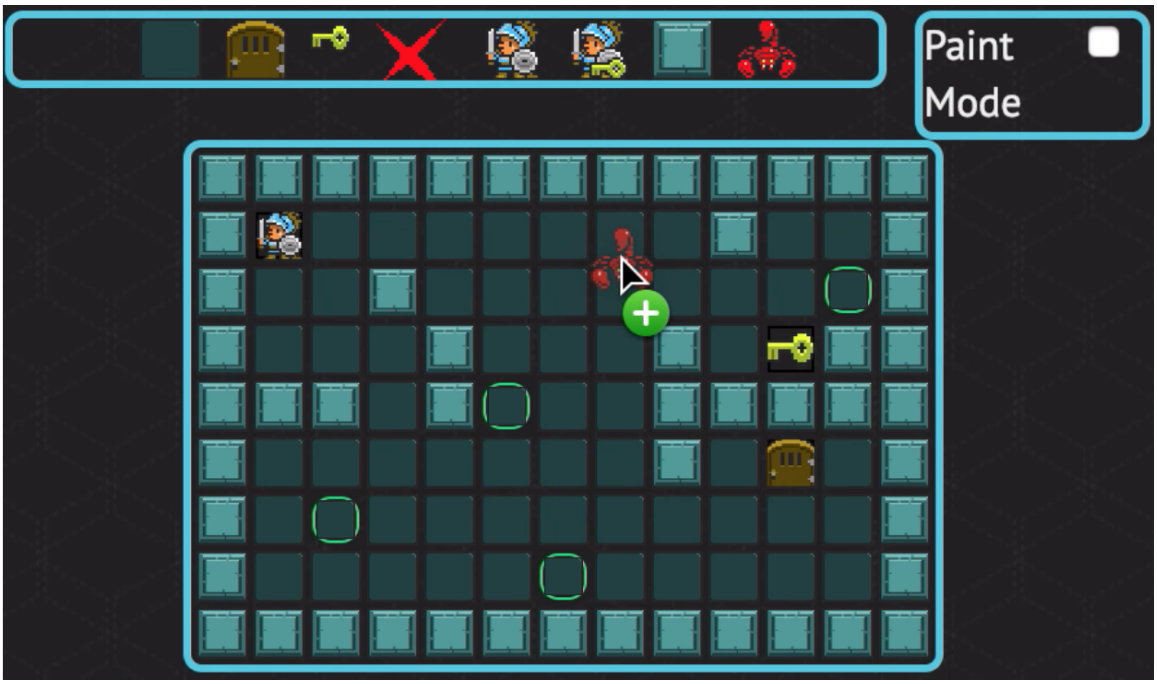


Figure 9.8: An example of recommending placement positions to a sprite. It is suggested, in this case, to place the enemy sprite far from the avatar player (top left corner) to avoid an instant kill when the game starts.

9.8 A sample session

The version of the system introduced in this paper has been used informally by users for an informal test session, a computer science graduate student with experience in game development, and a digital media professional, also with experience in game design. We started their sessions by introducing the system and teaching them how to add sprites and interactions without recommender assistance. Then, we taught them how to use the recommendations. After the tutorial was finished, we asked them to use the tool freely and design a prototype of a hack-and-slash game in the style of the ones available for the Super Nintendo Entertainment System. The sessions were recorded in video. In both of the experiences, the recommender system was used as expected. Half or more of the sprite set was composed by recommended sprites. Users especially benefited from recommendations of sprites able to cast or spawn objects. In most of the cases, they accepted the sprites (and the ones attached to them) exactly as they were suggested, in other cases, they benefited from the customizing available and made some adjustments to fit the sprites behavior to what they had in mind for the game. In both situations, the users' interactions were faster than having to design the equivalent functionality from scratch. More than that, they indicated the suggestions gave them ideas about design elements they were not thinking of before. For

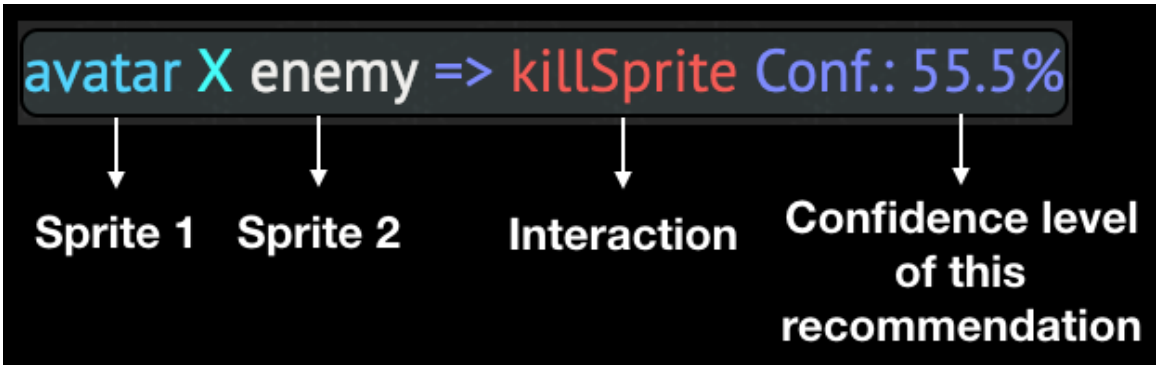


Figure 9.9: An example of a interaction recommendation component. The avatar sprite will be killed (killSprite interaction event) when colliding with a sprite enemy.

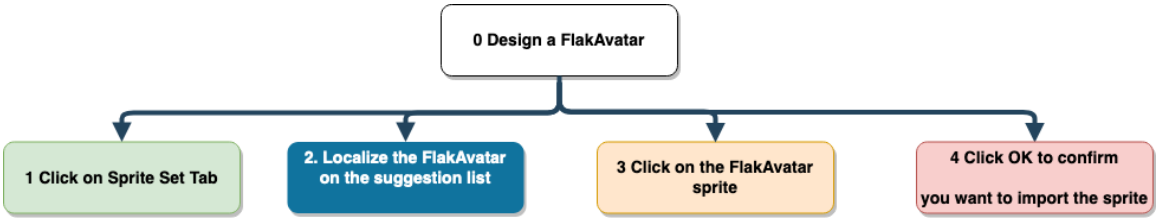
the interactions, almost half of the recommendations were accepted but the users often made adjustments in one of the items. Still, again, it is better than having to describe everything from scratch. The number of recommendations accepted was far higher than what we expected. We can say that half or more of the game design happened in a mixed-initiative way, with a human accepting the suggestions of an algorithm, and in some cases, making modifications to fit their needs. More than that, the users really put in the effort and did not only choose the first recommendations they were given. They really took their time to understand the recommendations, to inspect and test them, to change what they wanted to change, all based on a desire to make the best possible game in tandem with the AI-assistant (Figure 9.11).

9.9 Task Analysis

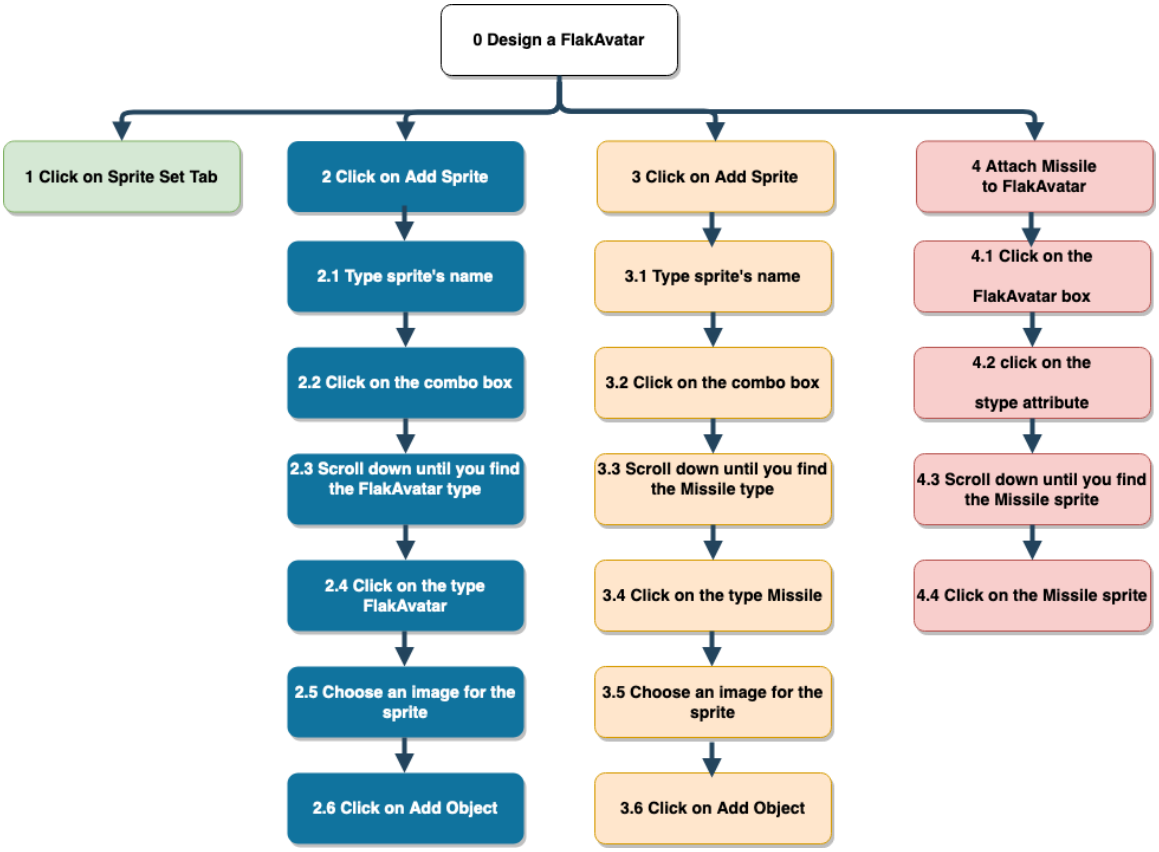
After watching the video, we decided to do a hierarchical task analysis (HTA) to visualize and compare the two approaches of developing game components with and without the assistance provided by Pitako. We took a *FlakAvatar* sprite to exemplify. The behavior of a *FlakAvatar* is the one we can find in space shooter games like *Space Invaders*. The Player moves to the right and the left of the screen and shoots projectiles to destroy its enemies. It is a common behavior of game elements, and in VGDL its implementation is similar to other types like the one implemented (*ShootAvatar*) by the participants in the experience above. The difference is that while the *FlakAvatar* moves only horizontally, *ShootAvatar* moves both horizontally and vertically. The hierarchical task analysis of the implementation without Pitako involves many subtasks since it requires the creation of an extra sprite that needs to be associated to a parent one. To analyze the difference of workload between the two approaches, with AI and without it, we placed the two hierarchies at page 80.

9.10 Conclusions and future work

In this chapter, we discussed the design and implementation of a recommender system for game mechanics. It was our second attempt to design such a system. We took inspiration from previous works which found similarities in games. These similarities were sometimes related to a whole game genre, like puzzles that have the same mechanics as Tetris [68] or a single mechanic feature like jumps in 2D platform games [127]. We developed our system on top of the VGDL and GVGAI framework to be general across game genres and mechanics. First, we reduced all the game descriptions available to a description in which we could have easy access to all the game components, like a catalog of chemical elements or biological species. In general, this second version improves the predecessor in many aspects. The methods are more formal and have been applied in recommender systems and data mining projects, the results do not repeat themselves immediately, and the user interface is flexible and easier to maintain. The recommender system itself is based on association rules and frequent item sets. By applying the apriori algorithm, our search process navigates into our catalog of game elements and finds associations between the sprites, their interactions, and the positions they have in different games and levels. Therefore, our system recommends sprites and interactions, the core mechanics of a VGDL game and also, where to put in the level of the game in development. In the next chapter, we will present a quantitative experiment to prove the effectiveness of design games assisted by a recommender system. Plus, we hope this work will help push the use of recommender systems beyond the e-commerce field, and that it can be used more as a method of (game) design assistance. A preliminary and informal study showed that the mixed-initiative design was well employed by the users, with them having half of their game design done by an algorithm's suggestion.



(a) Task Analysis of designing the spaceship of Space Invaders with Pitako.



(b) Task Analysis of designing the spaceship of Space Invaders without Pitako.

Figure 9.10: A comparison of the airship creation task in space invaders with and without the recommender system (Pitako).

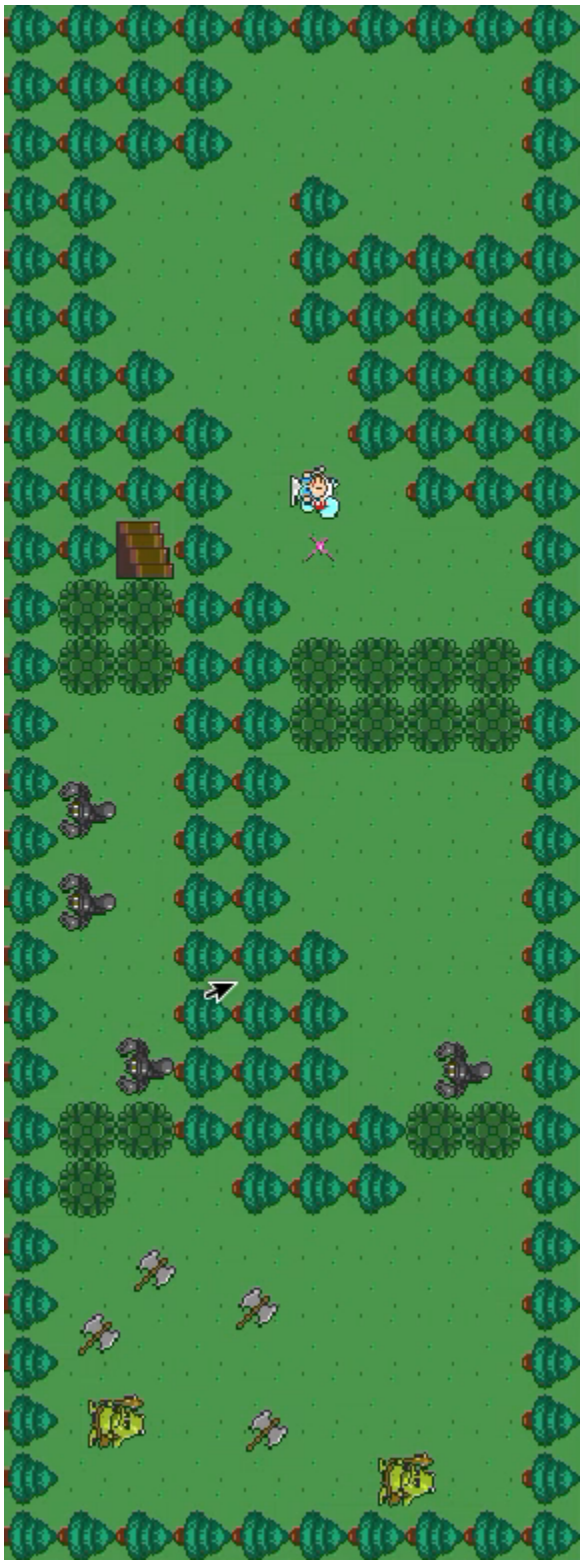


Figure 9.11: An example of a game designed by a user with the help of our recommender system. This game combines elements from *Plants vs. Zombies (PopCap Games)* and *R-Type (Irem, 1991)*

Chapter 10

Pitako's evaluation

10.1 Introduction

In this chapter, we report the results of a quantitative study to show how users react in the presence of the recommender system (Pitako). Eighty-seven (87) participants were divided into two groups, both had to design the same game, however, one of the groups was instructed to use the AI-assistant. In particular, we evaluated how users' level of self-efficacy, affect, workload, and accuracy varied for each group. We hypothesized that the presence of an AI-driven game design assistance tool would **(H1)** decrease the workload, **(H2)** increase self-efficacy, **(H3)** increase affect, and **(H4)** increase accuracy. By having appropriate tools, the users can think more about their design and spend less time dealing with tool issues [117]. We hope the results presented here will foster a new generation of AI-assistants for game design tasks focused both on its technical aspects as the human ones. This formal approach to evaluating an AI-game design tool is the main contribution of this paper.

10.2 User Study

A quantitative research design was used, including three standardized forms and a procedure.

10.2.1 Subjects

Eighty-seven (87) students from our university computer science and interactive media departments participated in the study. They were recruited by their departments' email newsletter and had to fill a preliminary form to be contacted later in case of agreement in participating in the experiment. The recruitment message informed them that experience

with authoring tools was required as well as an interest in playing games. The participants were rewarded by volunteering in this study with a 20 USD Amazon Gift Card.

10.2.2 Research Material

Two versions of Pitako (one with the recommender system and another one without it) were made available to download by the participants. For each software, a video tutorial was available as well.

10.2.3 Sources of Data

Four sources of data were used to evaluate the effectiveness of the two versions of the system. The NASA-TLX questionnaire [57] was used to evaluate the participants' perceived workload in all the dimensions of the questionnaire that covers from mental to physical effort. The PANAS questionnaire [31] was used to measure participants' computational affect. It is a 20-item measure of positive affect (10 items) and negative affect (10 items). All items are rated on a Likert-scale ranging from 1 (not at all) to 5 (extremely). Subjects responded to PANAS items on the basis of to what extent they feel after completing the task. Computer self-efficacy was used to measure how confident users were about their skills with respect to the evaluated systems [88]. The last source of data was the games designed by the users during the experiment. We used them to evaluate how accurate were users in design *Space Invaders* with the recommender system and without it.

10.2.4 Procedure

Once the participants agreed to be part of the experiment, an email was sent to them. The email contained a link in which they could download the software and a document to guide them about how to do the study. In the first part of the experiment, the guide explained to the users how to install the software. Following, the guide explained the task to be executed: design *Space Invaders* without using the recommender system or design *Space Invaders* by using the recommender system. A video showing a prototype similar to the one to be designed was linked in the guide document. Therefore, those not familiar with *Space Invaders* could have a feel about how the mechanics to be implemented works. At this point, the guide provided video tutorial links and instructions to the participants about each one of the components they had to learn and follow to perform the task. In both versions, participants were timed in 30 minutes after they pressed a button agreeing they were ready to start. After the experiment was over, either by the decision of the users, or by time, they

were prompted to fill the forms (NASA-TLX, PANAS, and Computer Self-Efficacy) and submit their game.

10.3 Results

In this section, we will present our results first by listing in the following subsections the hypothesis that our statistical analysis showed significance: workload, affect, and accuracy; and then the results for computational self-efficacy. Please, note that we divided the participants into two groups. The AI group have the participants who have experimented with Pitako (recommender system), while the noAI group have done the experiment without it.

10.3.1 Statistical Methods

We decided to use Wilcoxon-Whitney test because of the data distribution in our samples. We have both normal and non-normal distributions. As a non-parametric test, the Wilcoxon-Whitney deals with both types, whereas no individual parametric test (e.g., Student's T-test), would be appropriate for both.

We have used one-sided tests because we were interested in evaluating if the scores of a group would be higher (or lower) for the specific tests we were running. For example, when testing accuracy, the hypothesis is that the scores in the AI group will be higher. When testing workload, the hypothesis is that the scores will be lower for the AI group.

The Wilcoxon-Whitney was then applied to our hypothesis of reduced workload (H1), increased affect (H2), and increased self-efficacy (H3). Since these hypotheses were tested by using questionnaires with multiple dimensions, Wilcoxon-Whitney was suitable to deal with the different distribution formats each dimension was presenting.

10.3.2 Hypothesis 1 - Reduced Workload

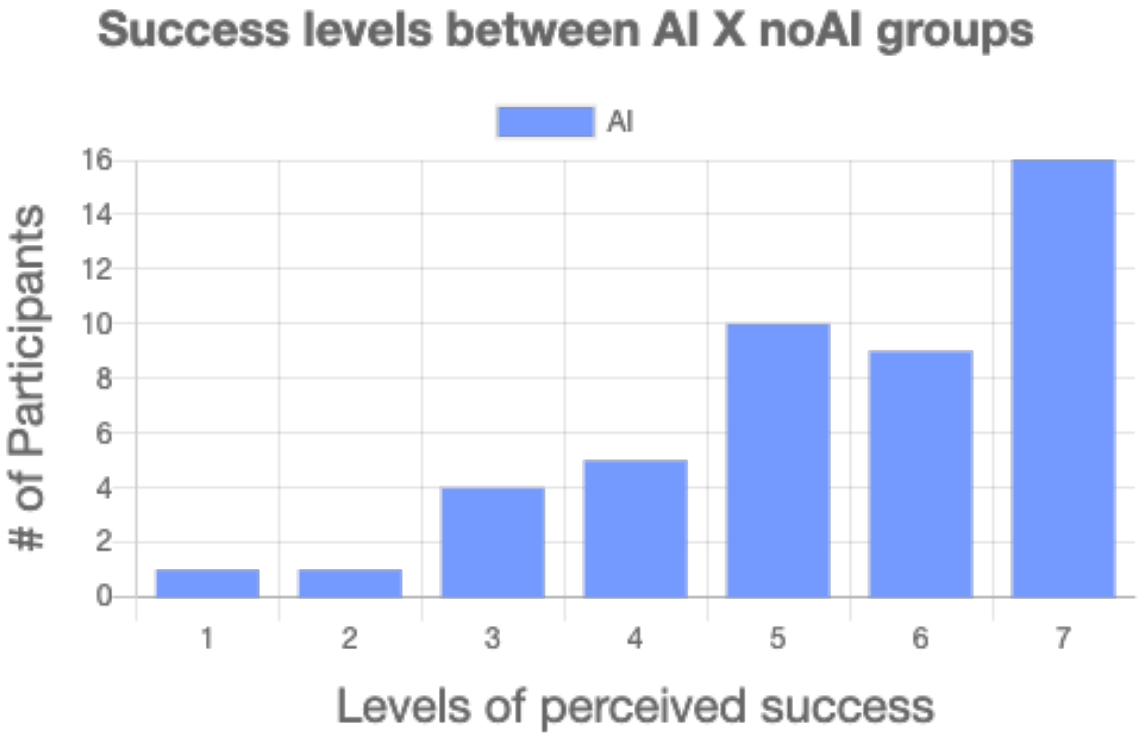
We hypothesized that the recommender system would reduce the amount of workload required to accomplish the goal of designing *Space Invaders* in comparison against designing it without the recommender. Our tests showed statistical significance in four out of five of the questions of the NASA-TLX questionnaire by applying a one-sided Wilcoxon-Whitney test. Users in the AI group reported lower levels of mental effort ($p \approx 0.0003$) and insecurity ($p \approx 0.0002$). They also reported they did not feel the task was rushed ($p \approx 0.0008$) and that it required someone to work hard to accomplish what was required in the task ($p \approx 0.002$).

The only dimension of the NASA-TLX questionnaire for what we could not find statistical significance was perceived success ($p \approx 0.07$).

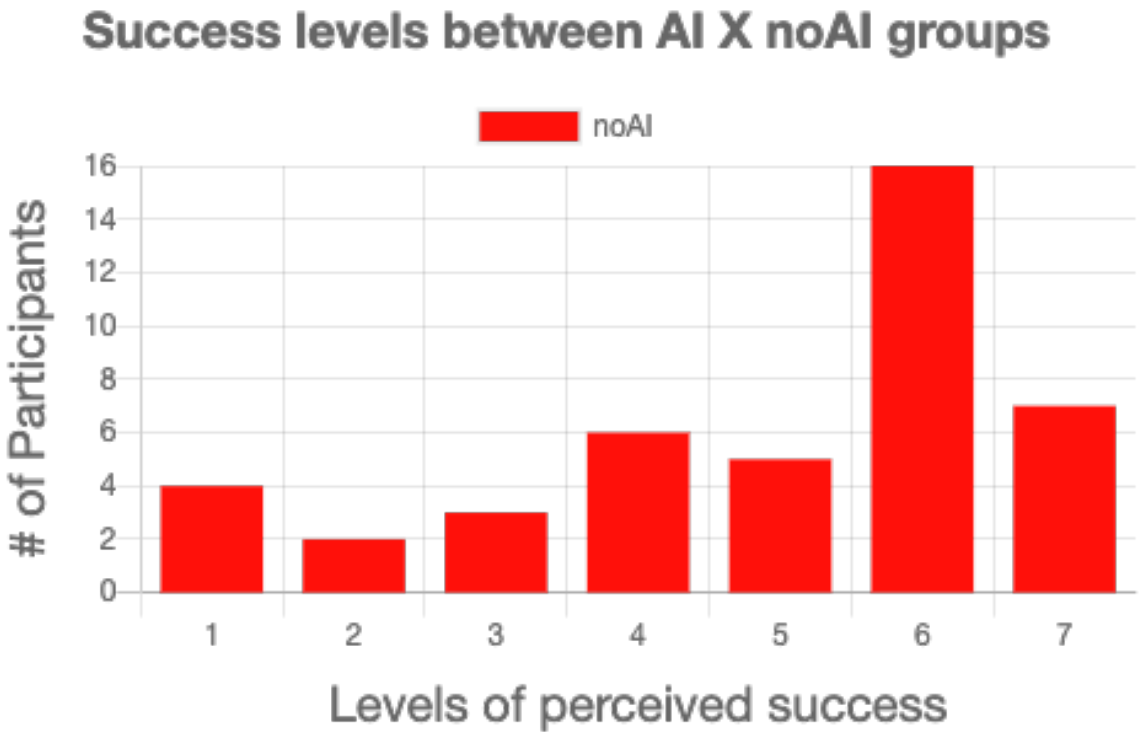
By taking a look at the histograms, we can see that the AI group were more confident to report success. Also, they almost did not report a complete fail (see Figure 10.1(a) on the following page). In opposite, participants in the noAI group were not that confident to report their own success. The report of complete fail was also easily noticeable in this group (see Figure 10.1(b) on the following page). In general, the analysis of the workload dimension of this study showed that the presence of the recommender system can reduce the amount of perceived effort by the users.

10.3.3 Hypothesis 2 - Increased Affect

We hypothesized that the recommender system would increase the level of computational affect. We applied the PANAS questionnaire, and we obtained a statistical significance for the positive and negative scores by using a one-sided Wilcoxon-Whitney test. The experience was more positive for the AI group ($p \approx 2.2e\text{-}16$), and it was more negative for the noAI group ($p \approx 1.437e\text{-}10$). After applying Bonferroni correction, we could not see statistical significance for individual dimensions of the questionnaire. However, it is worth to note that by analyzing the means for the positive and negative dimensions of the questionnaire, users in the AI group showed better levels of affect in thirteen (13) of them. In particular, "interest", "excited", "inspired", "attentive", "active", "distressed", "upset", "guilty", "hostile", "irritable", "ashamed", "nervous", and "afraid" were the ones when the AI group took advantage. "Strong", "enthusiastic", "proud", "alert", and "jittery" were the ones the noAI group got better scores. For "scared" and "determined", the means were too close to draw any conclusion. When we analyzed their scores in the questions in which they were asked about their "upsetness", "irritability", and ‘‘distressed’ levels, we could see that these dimensions are the ones with the biggest variance in this study and they show the AI group faced less discomfort during the experiment. Both groups showed similar levels of "pride" with a slight advantage for the noAI group. It raised a hypothesis that the presence of an AI system performing half (or more) of the task can remove the sense of proud in an individual. In general, the AI group reported lower levels of negative effect, and for positive affect the results of both groups were closer, with a small advantage for those in the AI group (see Figure 10.2).



(a)



(b)

Figure 10.1: From (a) to (c) we can see frames of a Space Invaders gameplay session. The black circles highlights areas where the users found flaws in the game rules.

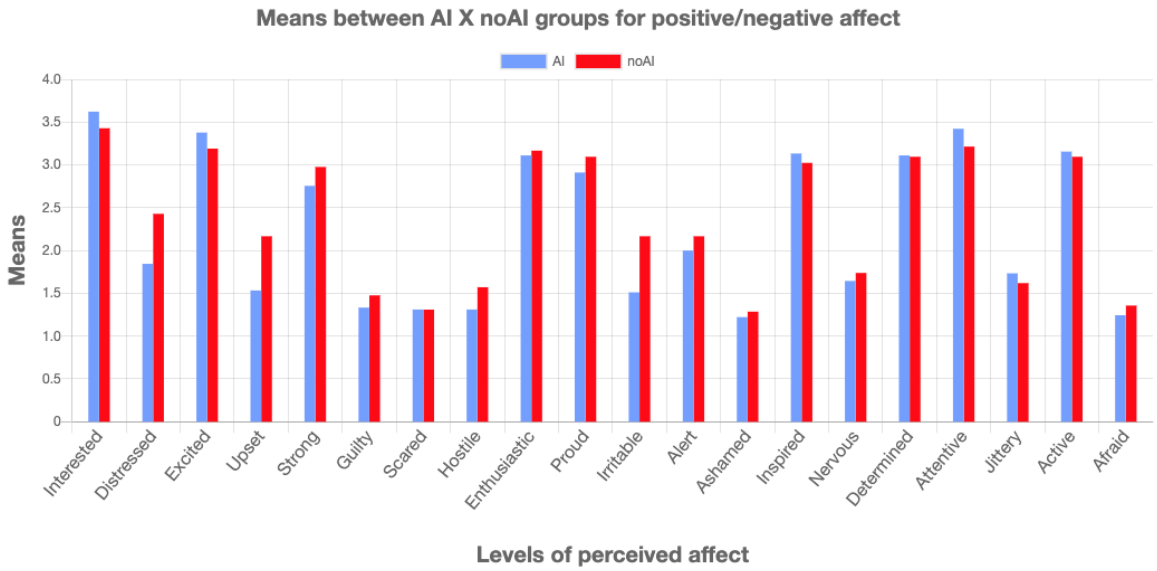


Figure 10.2: Means for the Positve/Negative affect from the two groups observed (AI and non-AI) in this experiment.

10.3.4 Hypothesis 3 - Increased accuracy

For evaluating the accuracy, we asked the participants to submit their games after they were done with the task (either by their own or by time-limit). We designed a program to read their game descriptions and attribute points to it. The program contained all the rules required to run the game exactly as required in the document they received to guide them over the experiment. The rules were divided in two sets, *Sprite* rules and *Interaction* rules to match the VGDL game descriptions sets evaluated in this study. One point was attributed for each rule generated corrected by the users. No half points were attributed, either the rule was correct, or it would not be enough to get a score. Before running the score evaluation, we ran all the games submitted. Those that could not run got a score of zero (0). From the forty-five (45) submissions of the AI group, only four could not run. From the forty-two (42) submissions of the noAI group, twelve of them were not running.

After this initial stage, we executed the grader software over the entries of the two groups. Then with all the scores available, we ran a one-sided t-test to evaluate the hypothesis that the recommender system would increase the accuracy of the participants. The result showed statistical significance ($p \approx 0.004574$). Thirty (30) out of Forty-five (45) submissions from the AI-group got the maximum score (12 points) against twenty (20) out of Forty-two (42) submissions in the noAI group. 67% (AI) against 47% (noAI). In the AI group, excluding the submissions with the maximum scores and the zeroing ones, only two entries got very low scores (2 and 3), most of the others got scores of 8, 9, and 10 points. The noAI group

did not have so low scores after excluding zeroing and maximum scores entries, however, they could not get close to the ideal (12) and reached average values like 4, 5, and 6 points. In general, by looking at the final results, we saw that most of the errors for the two groups (however, with more occurrences in the noAI group) were happening in the interaction set. Alternatively, they were missing interactions or using the incorrect sprites to missing them, applying interactions that were not required also was a common mistake presented. By our surprise, we saw two entries in the AI group that even provided the termination set of the game, i.e., the conditions that define if a gameplay session results in a win or a loose state. Just for the record, no bonus points were given and, of course, the termination set was not used for evaluation in any case.

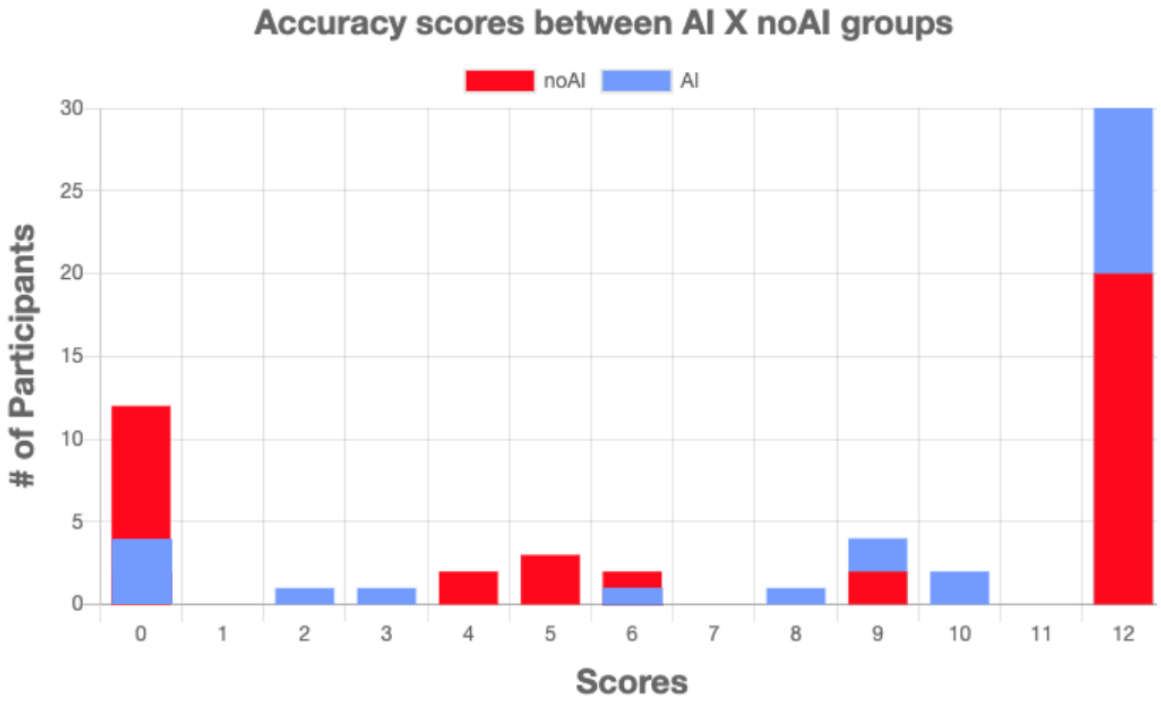


Figure 10.3: Accuracy scores for the two groups. We can see that users in the AI group performed better than the ones in the noAI group. 30 of the users in the AI group delivered their tasks with maximum accuracy (12 points) against 20 from the noAI group. A complete fail (0 points) was less common in the AI group as well. 4 against 12 total fails from the noAI goup.

10.3.5 Hypothesis 4 - Increased Self-Efficacy

We hypothesized that the presence of the recommender system would increase the participant's Self-Efficacy. However, after analyzing that by applying a one-side Wilcoxon-Whitney test, we could not find statistical significance. For all the questions of the computer

self-efficacy scale, just one of them reached a value that could be significant ($p \approx 0.047$), however not considered after applying Bonferroni correction. The question asked if the participant would be able to perform the task if they had never used a similar product before. Participants' from the AI group were more eager to agree than participants' from the noAI group. This was the only question whose difference of means between the two groups was greater than 1.6. All the other questions had a mean difference between 0.27 and 0.82. As the computer self-efficacy scale does not have a method of a evaluate the whole experience as PANAS for example, and because all the questions got results too close for both groups, we do not have how to accept or refute the hypothesis that the recommender system would increase the levels of users' self-efficacy.

10.3.6 Qualitative Analysis

We decided to include an open field where the users would be free to report anything they wanted about the whole experience. It was not mandatory and appeared after the forms previously discussed. Forty people submitted their comments, 22 from the noAI group and 18 from the AI one.

We categorized them into effort and positive/negative impact by analyzing their inputs using the online free trial of the Atlas.ti software. We could use this qualitative approach to support our previous findings. Users in the AI group were more positive about the experience, basically, their speech is all categorized as of positive impact and low effort. One of the participants stated how easy the whole experience was by saying that:

> "*the game dev kit itself makes it super easy to build a game because it tells you what you can choose and based on what you have chosen, it will tell you the possible interactions that can happen between them."*

Another user stated that the system is self explanatory,

> ''*I really liked how the interactions were suggested so it was self explanatory and leads one to create the game"*

Finally, users also expressed their contentment with the study by affirming that

> "*it was exciting and fun to play."*

and that:

> *"overall it was a fun and pleasant experience."*

In the noAI group, users' answers were more often categorized as indicating negative impact and high effort. Some of them were complaining about the task's time,

> *"it requires a little bit more time to watch videos and design level as well. I could complete everything except for the level design."*

Others complained about bugs they found in the tool,

> *"The application broke part of the way through so I couldn't finish the task. I got as far as making the enemies, but the bombs wouldn't go down properly and I couldn't even see the enemy sprite. Then I made some more enemies exactly as the video showed it(down to the sprite), and those wouldn't show up."*

Even when they expressed positive reactions, they were followed by problems they faced during their experience:

> *"I really liked the UI of the tool but I had a lot of trouble with the interactions."*

In general, we saw that users in the AI group would report better experiences and even enjoyment to some extent because the automatic procedures saved them time and effort to learn and even master UI commands. By contrary, participants in the noAI group had to worry about all the procedures to perform the task since they do not had any kind of automatic assistance. It increases their exposure to commit mistakes and even find bugs that are skipped or solved by an algorithm that they did not have access.

10.4 Limitations

A recommender system for game design like the one we presented in this thesis is something that we have not found out in the game research literature. It is also not present in the game

engines and game frameworks available nowadays. The comparison we have done is about a UI system supported by AI against a UI system without the same support. We did it to simulate the current status of game development tools that are currently in use and the one we hypothesized that will bring gains to game developers. The results can show that a system without the AI support makes the task harder for the users and consequently can bring bias to the experiment. One way to address this limitation would be with similar experiments that take into account the recommender system assistance we have presented, and another method that also offer assistance in the same level of game design task, considering the same design choices and features we implemented.

10.5 Conclusion

In this chapter we evaluated Pitako, our recommender system for assisting novice game designers, technically and fundamentally discussed on the two previous chapters. As stated earlier, it provides recommendations based on frequent itemset data mining algorithms. Designers get the suggestions while design their games. Their choices tune the system and it is up to them to explore common choices and design clones with small changes, or getting recommendations that lead them to try something new. Because this tool offers components already created and tends to avoid users effort in design everything from scratch, we hypothesized that such a tool would decrease workload (H1), increase computational affect (H2), increase accuracy (H3), and finally, increase self-efficacy (H4).

We recruited 87 participants and divided them in two groups. We asked them to design the game *Space Invaders*. One group executed the task with Pitako and the other group without it. Our results found with statistical significance that the presence of the recommender system decreases the perceived users' workload, increases their computational affect, and increases their accuracy. No statistical significance was found for the users's self - efficacy.

Computer affect is in particular an interesting way to push this work forward. We found statistical significance difference that the participants' in the AI group had a more positive experience as a whole. However, for particular sub-dimensions of computational affect we could not see (with statistical significance) how participants' got influenced by the AI presence. One of them showed that participants' in the noAI group felt more proud in accomplishing the task. Does that mean that the procedural automatic content suggested (or found) by an AI reduces the proud level of the user? This is still an open question.

Participants also gave us their impressions, that we could categorize by analyzing their free-text answers. The AI group reported a more pleasant experience while the noAI group

reported their frustration. The presence of a recommender system in the AI group allowed the participants to keep their focus (almost) entirely on the task, while the noAI group had to learn and remind UI commands that exposed them to more mistakes and difficulties in accomplishing the task.

We encourage more studies and evaluation of AI-game design assistants with these dimensions in mind: workload, affect, accuracy, and self-efficacy. We are particularly interested in seeing how the experience will change the participants perception when they need to be exposed to the tool for long periods of time, needs to design games of different complexities, and test the tool in both ways (with and without Pitako). Although, it was clear that the AI presence reduces the workload, increases the accuracy, and the positiveness affect levels of an algorithmic experience as a whole, more work needs to be done to identify how to build AI-based tools to boost human performance.

Chapter 11

Conclusion

This dissertation addressed the Thesis Statement introduced in Chapter 1:

> ***We can use tools based on artificial intelligence to design systems able to help humans in game designing tasks.***

To support the statement the following hypothesis was formulated.

- **H1 - Workload** - The presence of AI reduces the users self-perception of workload.
- **H2 - Accuracy** - The presence of AI increases the users' accuracy.
- **H3 - Computer Affect** - The presence of AI increases users levels of computer affect.
- **H4 - Self-Efficacy** - The presence of AI increases users levels of computer self-efficacy.

In the previous chapters, we explained how our statement could be proved in many of the tools used to implement our system, Cicero, an AI-assisted game design tool that assists at many stages of a game design process. For example, it has a recommender system that can help a designer to start the development of a game, and a debugging assistance that can help the designer to test the game when it is getting close to its final stages of implementation. Of course, the tool is flexible enough to allow users to use its features at any point in their development process. Cicero, as mentioned above by our statements and the hypothesis that sustains it, has the goal to use AI to facilitate how to design a game by novice game designers. Besides the two features already stated, it also has playtrace aggregate visualization, a rule statistics report system, a replay analysis tool, and a query system for

game events. We have tested our tool to confirm our hypothesis with 119 participants in total, and in this section, we report our main conclusions and new ways to expand this work.

11.1 Findings

We could accept our hypothesis that the use of an AI-assistant tool reduces user effort in game design tasks. As showed in chapter10, when designing a game with the assistance of an AI tool, users reported feeling less workload than users who had to design the game just by using the regular interface of the system. This hypothesis presented statistical significance for four out of the five dimensions in the NASA Task Load Index questionnaire that we used. The NASA TLX is a post-test form in which users fill a Likert scale set of questions to report they self-perception of workload after performing a given task.

One of the reasons for the reduced workload perception is that when you are designing a game with an AI-assistant, a lot of setup or configuration work will be done by the AI. Therefore you can keep your focus on your design rather than spending time configuring a tool. More than that, by having an AI-assistant working with you, the chances of human error is also reduced since the human does not need to press buttons, mark wrong checkboxes, and slide bar knobs to an inappropriate position. The AI encapsulates everything, and the users just need to configure the game element behavior (if they want to) when the element is already done.

For this task, we presented a Hierarchical Task Analysis to compare how to design a specific game element (chapter9). The HTA with the AI-assistant is shorter and straightforward, it practically does not have any sub-tasks. By the other hand, the HTA without the assistant is longer and full of sub-tasks that requires a high volume of user interactions. One can argue against the level of customization a user might not have by using an assistant. In that case, all the results are open for customization, whether you are using an assistant or not. You can access all the elements created by you or the computer and change whatever you want to. However, the point about having an assistant is precisely to avoid this extra effort, but in the end, we think it is better to left all the final decisions in the hands of the users.

Although we had extensively covered the reduced workload by using Pitako, the recommender system for game mechanics, many of the Cicero's features were designed with this goal in mind, and some of them are even improvements because of users' feedback. For example, SeekWhence intends to reduce gameplay time for testing purposes by providing ways of reproducing a gameplay section (chapter 5). While users appreciated the tool, they gave us the suggestion for improving it by giving them the chance of querying for events that should be answered up front, without the necessity of going back and forth in a video

track, and that was the trigger for developing Kwiri (chapter 6).

...

We tested accuracy for game design and game debugging tasks. In general, for the two scenarios, the participants who had AI assistance outperformed the group of participants without the same assistance. We got statistical significance for the task of designing a game and one of the scenarios of the game debugging tasks. When designing a game, the accuracy of the AI group was perfect for two-thirds of the users, and less than 10% of them got a zero score. More than that, those who did not get a perfect score were able to get very close to it. The group without AI could have half of their users with a perfect score, but those who had not to achieve that mark had a total fail of zero scores (28%) or got scores far away from the perfect mark. For the game debugging tasks, in one of the scenarios, users were almost three times more accurate with AI assistance. They had to detect fake barriers in Space Invaders, and by using an agent and watch its actions, their precision was of 100%. The group without AI not only did not get close from the AI one, they also committed other mistakes more often. Basically, the action of playing a game and simultaneously debugging it is stressed and time-consuming. In a scenario like the ones created by us, the absence of an assistant requires a superhuman spatial memory, and because they are busy trying to stay alive, the only way to complete the task is by guessing.

...

We also confirmed that AI assistance improves levels of computational affect. In the task of designing a game, users in the AI group had a positive experience while users who designed the game without assistance reported a negative one. Their affect scores were obtained by using a PANAS test. After their participation in the task, they filled a form in which they attributed points to 10 categories of positive mood and ten categories for negative ones. After analysis their scores, we could see that users in the AI group scored higher for positive and lower for negative. The Participants in the group without AI reported exactly the opposite. For the individual categories themselves, we could not find statistical significance. However, the averages indicate the group with AI had better scores for 12 of them, what in the end counted in favor of increasing the total positive score. In particular, levels of “distress", “upsetness," and “irritability" were lower in the AI group and higher in the noAI group showing that levels of discomfort were more evident for the group without an assistant.

11.2 Impact downstream on the users and player

We believe that games designed with an AI-assistance have the potential to allow designers to explore more alternatives that they would do if continuing to work with the current available game design tools. Based on our results, and our population our experiments showed high improvements on workload levels, accuracy, and computational affect. It reduces time and effort what allows designers to think more about their games and less about the tools that are used to design them. Based on Shneiderman's classification [118], methods of automatic design alternative explorations, like the one presented by our recommender system, can be included in the *structuralist* and *inspirationalist* schools of creativity. The system offers both the combinatorial hypothesis generation of the *structuralist* school and the playful exploration of the *inspirationalist* one. However, we need to go further to prove that what we claim is correct. We plan to run a new experiment where the designers will use the tool to design a game from scratch (and not a target game like we did for this thesis). The games will then be available for being tested by players that will evaluate them according to their uniqueness. This way we will understand better how AI-assisted game design tools impact both the designer work and the player experience.

11.3 Future Work

We plan to integrate our implementation or even leverage our concepts to work in or with other tools, like commercial game engines like Unity or Unreal. An initial step would be build on the UnityVGDL project, a port of the GVGAI framework for the Unity engine [66]. Also part of our plans is integrating our system with other projects in the GVGAI ecosystem, for example the tutorial generation [52] and the level generation [71].

We intend to provide more customization for the AI-debugging system. During our tests, we found that some games need a specific strategy to be debugged, and this strategy is far away different from the strategies that an agent is using to win a game. We believe that agents based on imitation learning are a good starting point. In one of the cases we tested, looking for fake walls, an imitation learning agent would be very suitable for avoiding a human tester the time-consuming task of hitting its avatar against all walls in a labyrinth and take notes of the Xs and Ys positions where this problem occur.

The recommender system, Pitako, proved to be successful when tested for reducing users' workload and increasing their affect and accuracy. We want to improve it and offer assistance that could suggest game assets. As a reminder, if you take into account the Mechanics, Dynamics, and Aesthetics framework, Pitako works on the level of mechanics

and dynamics. We are interested in leverage it to suggest aesthetics components of a game, like the visuals of characters, scenarios, game items, sound effects, music soundtrack, and level backgrounds. Such a system could work in two ways: by generating the assets according to the mechanics (designed by a human user or an algorithm, or both like in Pitako) or it could generate the assets based on user inputs. The user would ask the recommender to generate assets for a Christmas or Valentine's day game, for example. The system would output previews of the game with the suggested assets, and the user would pick one of the suggestions or even a collection of them. It will be interesting in investigating how game mechanics and game assets are associated and how subjective one can be without lose its grasp to the other. There is always the risk of creating something nonsense here, but in many cases, as the game history taught us, sometimes to be nonsense is very welcome, as examples we can cite games like EarthWorm Jim (1994, Shiny Entertainment, Gameloft, Eurocom, Game Titan) and We Love Katamari (2005, Namco, Now Production).

In general, we want to see this concept of AI-game design assisted systems to offer assistance to other tasks in fields related to creative content and engineering. Screenwriting, 3D modeling and animation, cinematography, audio design, music composition, market campaign, character development, architecture, interior design, circuit design, software development, etc. Most of these tasks already have tools for assisting professionals. Celtx [67] and Final Draft helps writers to write their films or games, 3DSMax [35] and Maya [34] help with 3D modeling and character design in general, there are a myriad of CAD tools for architecture and interior design [14], same is valid for circuit and software design [89]. However, their AI-assistance level is still in need of more exploration as we showed in the 2 chapter. All these tools gained many improvements since their first installment, collaborative versions stored in the cloud, fast response due to distributed and parallel computing, customizable UIs, etc. However, they share the same issues of game design tools, the ones we tried to mitigate with our systems. Too many efforts in tool configuration that takes the designer away of thinking on their ideas. If a set of AI-assisted tools, like the ones created for this thesis, could have the same kind of benefits, i.e reduction of workload, increase of affect and accuracy, for all these other tasks we will give a step forward in creating a generation of tools that can lead their users to be more creative more often. [118].

Bibliography

[1] *World of Tanks*, 2017 - http://worldoftanks.com/ - (accessed February 23, 2017).

[2] Nintendo, 2019, (https://www.nintendo.com/ - accessed February 23, 2017).

[3] Gediminas Adomavicius and Alexander Tuzhilin. Toward the next generation of recommender systems: A survey of the state-of-the-art and possible extensions. *IEEE Trans. on Knowl. and Data Eng.*, 17(6):734–749, June 2005.

[4] Rakesh Agarwal, Ramakrishnan Srikant, et al. Fast algorithms for mining association rules. In *Proc. of the 20th VLDB Conference*, pages 487–499, 1994.

[5] Rakesh Agrawal, Tomasz Imieliński, and Arun Swami. Mining association rules between sets of items in large databases. In *Acm sigmod record*, volume 22, pages 207–216. ACM, 1993.

[6] J. E. Allen, C. I. Guinn, and E. Horvtz. Mixed-initiative interaction. *IEEE Intelligent Systems and their Applications*, 14(5):14–23, Sep 1999.

[7] Kai Arulkumaran, Antoine Cully, and Julian Togelius. Alphastar: An evolutionary computation perspective. *arXiv preprint arXiv:1902.01724*, 2019.

[8] Albert Bandura. The explanatory and predictive scope of self-efficacy theory. *Journal of social and clinical psychology*, 4(3):359–373, 1986.

[9] Nolan Bard, Jakob N. Foerster, Sarath Chandar, Neil Burch, Marc Lanctot, H. Francis Song, Emilio Parisotto, Vincent Dumoulin, Subhodeep Moitra, Edward Hughes, Iain Dunning, Shibl Mourad, Hugo Larochelle, Marc G. Bellemare, and Michael Bowling. The hanabi challenge: A new frontier for ai research, 2019.

[10] Robert A Baron and Jintong Tang. The role of entrepreneurs in firm-level innovation: Joint effects of positive affect, creativity, and environmental dynamism. *Journal of Business Venturing*, 26(1):49–60, 2011.

[11] Martin W Bauer and George Gaskell. *Qualitative researching with text, image and sound: A practical handbook for social research*. Sage, 2000.

[12] Daniel Billsus and Michael J Pazzani. User modeling for adaptive news access. *User modeling and user-adapted interaction*, 10(2-3):147–180, 2000.

[13] Jonathan Blow. Game development: Harder than you think. *Queue*, 1(10):28–37, February 2004.

[14] Martin Bonev, Lars Hvam, John Clarkson, and Anja Maier. Formal computer-aided product family architecture design for mass customization. *Computers in Industry*, 74:58–70, 2015.

[15] Philip Bontrager, Ahmed Khalifa, Andre Mendes, and Julian Togelius. Matching games and algorithms for general video game playing. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.

[16] Brian Bowman, Niklas Elmqvist, and TJ Jankun-Kelly. Toward visualization for games: Theory, design space, and patterns. *IEEE transactions on visualization and computer graphics*, 18(11):1956–1968, 2012.

[17] Brenda Brathwaite and Ian Schreiber. Challenges for game designers, charles river media. *Inc., Rockland, MA*, 2008.

[18] Sergey Brin and Lawrence Page. The anatomy of a large-scale hypertextual web search engine. *Computer networks and ISDN systems*, 30(1-7):107–117, 1998.

[19] J. S. Brown, R. R. Burton, and F. Zdybel. A model-driven question-answering system for mixed-initiative computer-assisted construction. *IEEE Transactions on Systems, Man, and Cybernetics*, SMC-3(3):248–257, May 1973.

[20] Cameron B Browne, Edward Powley, Daniel Whitehouse, Simon M Lucas, Peter I Cowling, Philipp Rohlfshagen, Stephen Tavener, Diego Perez, Spyridon Samothrakis, and Simon Colton. A survey of monte carlo tree search methods. *IEEE Transactions on Computational Intelligence and AI in games*, 4(1):1–43, 2012.

[21] Robin Burke. Hybrid recommender systems: Survey and experiments. *User modeling and user-adapted interaction*, 12(4):331–370, 2002.

[22] Eric Butler, Adam M Smith, Yun-En Liu, and Zoran Popovic. A mixed-initiative tool for designing level progressions in games. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*, pages 377–386. ACM, 2013.

[23] Murray Campbell. Knowledge discovery in deep blue. *Communications of the ACM*, 42(11):65–65, 1999.

[24] Alessandro Canossa. Play-persona: Modeling player behaviour in computer games. 06 2009.

[25] Alessandro Canossa, Truong-Huy Nguyen, and Magy El-Nasr. G-player: Exploratory visual analytics for accessible knowledge discovery. 08 2016.

[26] Alessandro Canossa, Truong-Huy D Nguyen, and Magy Seif El-Nasr. G-player: Exploratory visual analytics for accessible knowledge discovery. 2016.

[27] Jennifer L Claggett and Dale L Goodhue. Have is researchers lost bandura's self-efficacy concept? a discussion of the definition and measurement of computer self-efficacy. In *2011 44th Hawaii International Conference on System Sciences*, pages 1–10. IEEE, 2011.

[28] Deborah R Compeau and Christopher A Higgins. Computer self-efficacy: Development of a measure and initial test. *MIS quarterly*, pages 189–211, 1995.

[29] Cristina Conati, Stacy Marsella, and Ana Paiva. Affective interactions: The computer in the affective loop. In *Proceedings of the 10th International Conference on Intelligent User Interfaces*, IUI '05, pages 7–7, New York, NY, USA, 2005. ACM.

[30] Paul Coulton, Will Bamford, Keith Cheverst, and Omer Rashid. 3d space-time visualization of player behaviour in pervasive location-based games. *International Journal of Computer Games Technology*, 2008:2, 2008.

[31] John R Crawford and Julie D Henry. The positive and negative affect schedule (panas): Construct validity, measurement properties and normative data in a large non-clinical sample. *British journal of clinical psychology*, 43(3):245–265, 2004.

[32] Suzanne C Danhauer, Charles R Carlson, and Michael A Andrykowski. Positive psychosocial functioning in later life: Use of meaning-based coping strategies by nursing home residents. *Journal of Applied Gerontology*, 24(4):299–318, 2005.

[33] Fred D Davis, Richard P Bagozzi, and Paul R Warshaw. User acceptance of computer technology: a comparison of two theoretical models. *Management science*, 35(8):982–1003, 1989.

[34] Dariush Derakhshani. *Introducing Autodesk Maya 2013*. John Wiley & Sons, 2012.

[35] Dariush Derakhshani and Randi L Derakhshani. *Autodesk 3ds Max 2013 essentials*. John Wiley & Sons, 2012.

[36] Heather Desurvire, Martin Caplan, and Jozsef A Toth. Using heuristics to evaluate the playability of games. In *CHI'04 extended abstracts on Human factors in computing systems*, pages 1509–1512. ACM, 2004.

[37] Heather Desurvire and Charlotte Wiberg. Game usability heuristics (play) for evaluating and designing better games: The next iteration. In *International Conference on Online Communities and Social Computing*, pages 557–566. Springer, 2009.

[38] Xiaoqian Ding, Yi-Yuan Tang, Rongxiang Tang, and Michael I Posner. Improving creativity performance by short-term meditation. *Behavioral and Brain Functions*, 10(1):9, 2014.

[39] Alexander Dockhorn and Sanaz Mostaghim. Introducing the hearthstone-ai competition, 2019.

[40] Clive Eastman and John S Marzillier. Theoretical and methodological difficulties in bandura's self-efficacy theory. *Cognitive Therapy and Research*, 8(3):213–229, 1984.

[41] Eric Eaton, Sven Koenig, Claudia Schulz, Francesco Maurelli, John Lee, Joshua Eckroth, Mark Crowley, Richard G. Freedman, Rogelio E. Cardona-Rivera, Tiago Machado, and Tom Williams. Blue sky ideas in artificial intelligence education from the eaai 2017 new and future ai educator program. *AI Matters*, 3(4):23–31, February 2018.

[42] Marc Ebner, John Levine, Simon M Lucas, Tom Schaul, Tommy Thompson, and Julian Togelius. Towards a video game description language. 2013.

[43] IDE Eclipse. Eclipse foundation, 2007.

[44] Magy Seif El-Nasr, Anders Drachen, and Alessandro Canossa. *Game Analytics: Maximizing the Value of Player Data*. Springer Publishing Company, Incorporated, 2013.

[45] Manfred Eppe, Ewen Maclean, Roberto Confalonieri, Oliver Kutz, Marco Schorlemmer, Enric Plaza, and Kai-Uwe Kühnberger. A computational framework for conceptual blending. *Artificial Intelligence*, 256:105–129, 2018.

[46] Christine Franks, Zhaopeng Tu, Premkumar Devanbu, and Vincent Hellendoorn. Cacheca: A cache language model based code suggestion tool. In *Proceedings of the 37th International Conference on Software Engineering - Volume 2*, ICSE '15, pages 705–708, Piscataway, NJ, USA, 2015. IEEE Press.

[47] S. Fraser and D. Mancl. No silver bullet: Software engineering reloaded. *IEEE Software*, 25(1):91–94, Jan 2008.

[48] Frederik Frydenberg, Kasper R Andersen, Sebastian Risi, and Julian Togelius. Investigating mcts modifications in general video game playing. In *2015 IEEE Conference on Computational Intelligence and Games (CIG)*, pages 107–113. IEEE, 2015.

[49] Scientific Software Development GmbH. Atlas.ti, April 2017.

[50] Carlos A Gomez-Uribe and Neil Hunt. The netflix recommender system: Algorithms, business value, and innovation. *ACM Transactions on Management Information Systems (TMIS)*, 6(4):13, 2016.

[51] Gösta Grahne and Jianfei Zhu. Fast algorithms for frequent itemset mining using fp-trees. *IEEE transactions on knowledge and data engineering*, 17(10):1347–1362, 2005.

[52] Michael Cerny Green, Ahmed Khalifa, Gabriella A. B. Barros, Tiago Machado, Andy Nealen, and Julian Togelius. Atdelfi: Automatically designing legible, full instructions for games. In *Proceedings of the 13th International Conference on the Foundations of Digital Games*, FDG '18, pages 17:1–17:10, New York, NY, USA, 2018. ACM.

[53] Saurabh Gupta and Robert P. Bostrom. A revision of computer self-efficacy conceptualizations in information systems. *SIGMIS Database*, 50(2):71–93, May 2019.

[54] Matthew Guzdial, Nicholas Liao, Jonathan Chen, Shao-Yu Chen, Shukan Shah, Vishwa Shah, Joshua Reno, Gillian Smith, and Mark O. Riedl. Friend, collaborator, student, manager: How design of an ai-driven game level editor affects creators. In *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*, CHI '19, pages 624:1–624:13, New York, NY, USA, 2019. ACM.

[55] Jiawei Han, Jian Pei, and Yiwen Yin. Mining frequent patterns without candidate generation. In *ACM sigmod record*, volume 29, pages 1–12. ACM, 2000.

[56] Erik Harpstead, Christopher J MacLellan, Vincent Aleven, and Brad A Myers. Replay analysis in open-ended educational games. In *Serious games analytics*, pages 381–399. Springer, 2015.

[57] Sandra G Hart and Lowell E Staveland. Development of nasa-tlx (task load index): Results of empirical and theoretical research. In *Advances in psychology*, volume 52, pages 139–183. Elsevier, 1988.

[58] Erin J. Hastings and Kenneth O. Stanley. Galactic arms race: An experiment in evolving video game content. *SIGEVOlution*, 4(4):2–10, March 2010.

[59] Christoffer Holmgård, Antonios Liapis, Julian Togelius, and Georgios N Yannakakis. Generative agents for player decision modeling in games. In *FDG*. Citeseer, 2014.

[60] Eric Horvitz. Principles of mixed-initiative user interfaces. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, CHI '99, pages 159–166, New York, NY, USA, 1999. ACM.

[61] Feng-hsiung Hsu. Ibm's deep blue chess grandmaster chips. *IEEE Micro*, 19(2):70–81, 1999.

[62] Robin Hunicke, Marc LeBlanc, and Robert Zubek. Mda: A formal approach to game design and game research. In *Proceedings of the AAAI Workshop on Challenges in Game AI*, volume 4, page 1722, 2004.

[63] Takeo Igarashi and John F Hughes. A suggestive interface for 3d drawing. In *Proceedings of the 14th annual ACM symposium on User interface software and technology*, pages 173–181. ACM, 2001.

[64] A. Isaksen, D. Gopstein, J. Togelius, and A. Nealen. Exploring game space of minimal action games via parameter tuning and survival analysis. *IEEE Transactions on Computational Intelligence and AI in Games*, PP(99):1–1, 2017.

[65] Alexander Jaffe, Alex Miller, Erik Andersen, Yun-En Liu, Anna Karlin, and Zoran Popovic. Evaluating competitive game balance with restricted play. In *AIIDE*, 2012.

[66] Mads Johansen, Martin Pichlmair, and Sebastian Risi. Video game description language environment for unity machine learning agents.

[67] Mike Jones et al. Getting started with celtx: Scriptwriting and pre-production. *Screen Education*, (46):196, 2007.

[68] Jesper Juul. Swap adjacent gems to make sets of three: A history of matching tile games. *Artifact*, 1(4):205–216, 2007.

[69] Jussi Kasurinen, Jukka-Pekka Strandén, and Kari Smolander. What do game developers expect from development and design tools? In *Proceedings of the 17th International Conference on Evaluation and Assessment in Software Engineering*, EASE '13, pages 36–41, New York, NY, USA, 2013. ACM.

[70] Ahmed Khalifa, Aaron Isaksen, Julian Togelius, and Andy Nealen. Modifying mcts for human-like general video game playing.

[71] Ahmed Khalifa, Diego Perez-Liebana, Simon M. Lucas, and Julian Togelius. General video game level generation. In *Proceedings of the Genetic and Evolutionary Computation Conference 2016*, GECCO '16, pages 253–259, New York, NY, USA, 2016. ACM.

[72] Jared L. Katzman, Uri Shaham, Alexander Cloninger, Jonathan Bates, Tingting Jiang, and Yuval Kluger. Deepsurv: Personalized treatment recommender system using a cox proportional hazards deep neural network. *BMC Medical Research Methodology*, 18, 12 2018.

[73] J Laurent, K Potter, and SJ Catanzaro. Assessing positive and negative affect in children: The development of the panas-c. In *26th annual convention of the National Association of School Psychologists, Seattle, WA*, 1994.

[74] Antonios Liapis, Georgios N Yannakakis, and Julian Togelius. Sentient sketchbook: Computer-aided game level authoring. In *FDG*, pages 213–220, 2013.

[75] Weiyang Lin, Sergio A Alvarez, and Carolina Ruiz. Efficient adaptive-support association rule mining for recommender systems. *Data mining and knowledge discovery*, 6(1):83–105, 2002.

[76] Greg Linden, Brent Smith, and Jeremy York. Amazon. com recommendations: Item-to-item collaborative filtering. *IEEE Internet computing*, (1):76–80, 2003.

[77] Yun-En Liu, Erik Andersen, Richard Snider, Seth Cooper, and Zoran Popović. Feature-based projections for effective playtrace analysis. In *Proceedings of the 6th international conference on foundations of digital games*, pages 69–76. ACM, 2011.

[78] F. Lu, K. Yamamoto, L. H. Nomura, S. Mizuno, Y. Lee, and R. Thawonmas. Fighting game artificial intelligence competition platform. In *2013 IEEE 2nd Global Conference on Consumer Electronics (GCCE)*, pages 320–323, Oct 2013.

[79] Simon M. Lucas, Michael Mateas, Mike Preuss, Pieter Spronck, and Julian Togelius. Artificial and Computational Intelligence in Games (Dagstuhl Seminar 12191). *Dagstuhl Reports*, 2(5):43–70, 2012.

[80] Tiago Machado, Ivan Bravi, Zhu Wang, Andy Nealen, and Julian Togelius. Cicero:computationally intelligent collaborative environment for game and level design. 2016.

[81] Tiago Machado, Ivan Bravi, Zhu Wang, Andy Nealen, and Julian Togelius. Shopping for game mechanics. 2016.

[82] Tiago Machado, Dan Gopstein, Andy Nealen, and Julian Togelius. Pitako - recommending game design elements in cicero. In *IEEE Conference on Games*, 2019.

[83] Tiago Machado, Daniel Gopstein, Andy Nealen, Oded Nov, and Julian Togelius. Ai-assisted game debugging with cicero. *2018 IEEE Congress on Evolutionary Computation (CEC)*, pages 1–8, 2018.

[84] Tiago Machado, Daniel Gopstein, Andy Nealen, and Julian Togelius. Kwiri-what, when, where and who: Everything you ever wanted to know about your game but didnâĂŹt know how to ask. In *Knowledge Extraction From Games Workshop. AAAI*, 2019.

[85] Tiago Machado, Daniel Gopstein, Oded Nov, Angela Wang, Andy Nealen, and Julian Togelius. Evaluation of a recommender system for assisting novice game designers, 2019.

[86] Tiago Machado, Andy Nealen, and Julian Togelius. Seekwhence a retrospective analysis tool for general game design. In *Proceedings of the 12th International Conference on the Foundations of Digital Games*, FDG '17, pages 4:1–4:6, New York, NY, USA, 2017. ACM.

[87] Tobias Mahlmann, Anders Drachen, Julian Togelius, Alessandro Canossa, and Georgios N Yannakakis. Predicting player behavior in tomb raider: Underworld. In *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*, pages 178–185. IEEE, 2010.

[88] George M Marakas, Mun Y Yi, and Richard D Johnson. The multilevel and multifaceted character of computer self-efficacy: Toward clarification of the construct and an integrative framework for research. *Information systems research*, 9(2):126–163, 1998.

[89] Lisa G McIlrath. Methods and systems for computer aided design of 3d integrated circuits, March 1 2016. US Patent 9,275,185.

[90] Ben Medler et al. Generations of game analytics, achievements and high scores. *Eludamos. Journal for Computer Game Culture*, 3(2):177–194, 2009.

[91] Kathryn Merrick and Mary Lou Maher. Motivated reinforcement learning for non-player characters in persistent computer game worlds. In *Proceedings of the 2006 ACM SIGCHI international conference on Advances in computer entertainment technology*, page 3. ACM, 2006.

[92] Thomas Hunt Morgan, CB Bridges, and AH Sturtevant. The genetics of drosophila melanogaster. *Biblphia genet*, 2:1–262, 1925.

[93] Andy Nealen, Adam Saltsman, and Eddy Boxerman. Towards minimalist game design. In *Proceedings of the 6th International Conference on Foundations of Digital Games*, pages 38–45. ACM, 2011.

[94] Mark J. Nelson. Game metrics without players: Strategies for understanding game artifacts. In *Proceedings of the 19th AIIDE Conference on Artificial Intelligence in the Game Design Process*, AIIDE'11-19, pages 14–18. AAAI Press, 2011.

[95] Mark J. Nelson and Michael Mateas. A requirements analysis for videogame design support tools. In *Proceedings of the 4th International Conference on Foundations of Digital Games*, FDG '09, pages 137–144, New York, NY, USA, 2009. ACM.

[96] Tien T. Nguyen, Duyen T. Nguyen, Shamsi T. Iqbal, and Eyal Ofek. The known stranger: Supporting conversations between strangers with personalized topic suggestions. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 555–564, New York, NY, USA, 2015. ACM.

[97] Truong-Huy D Nguyen, Magy Seif El-Nasr, and Alessandro Canossa. Glyph: Visualization tool for understanding problem solving strategies in puzzle games.

[98] Christopher Nikulin, Gabriela Lopez, Eduardo Piñonez, Luis Gonzalez, and Pia Zapata. Nasa-tlx for predictability and measurability of instructional design models: case study in design methods. *Educational Technology Research and Development*, 67(2):467–493, 2019.

[99] Peter O'Donovan, Aseem Agarwala, and Aaron Hertzmann. Designscape: Design with interactive layout suggestions. In *Proceedings of the 33rd Annual ACM Conference on Human Factors in Computing Systems*, CHI '15, pages 1221–1224, New York, NY, USA, 2015. ACM.

[100] Changhoon Oh, Taeyoung Lee, Yoojung Kim, SoHyun Park, Saebom Kwon, and Bongwon Suh. Us vs. them: Understanding artificial intelligence technophobia over the google deepmind challenge match. In *Proceedings of the 2017 CHI Conference on Human Factors in Computing Systems*, CHI '17, pages 2523–2534, New York, NY, USA, 2017. ACM.

[101] Santiago Ontañón, Nicolas A Barriga, Cleyton R Silva, Rubens O Moraes, and Levi HS Lelis. The first microrts artificial intelligence competition. *AI Magazine*, 39(1), 2018.

[102] Santiago Ontanón, Gabriel Synnaeve, Alberto Uriarte, Florian Richoux, David Churchill, and Mike Preuss. A survey of real-time strategy game ai research and competition in starcraft. *IEEE Transactions on Computational Intelligence and AI in games*, 5(4):293–311, 2013.

[103] Joseph C. Osborn, April Grow, and Michael Mateas. Modular computational critics for games. In *Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'13, pages 163–169. AAAI Press, 2014.

[104] Rui Pan, Lyn Bartram, and Carman Neustaedter. Twitchviz: A visualization tool for twitch chatrooms. In *Proceedings of the 2016 CHI Conference Extended Abstracts on Human Factors in Computing Systems*, CHI EA '16, pages 1959–1965, New York, NY, USA, 2016. ACM.

[105] Diego Perez, Spyridon Samothrakis, Julian Togelius, Tom Schaul, Simon Lucas, Adrien Couëtoux, Jeyull Lee, Chong-U Lim, and Tommy Thompson. The 2014 general video game playing competition. 2015.

[106] Diego Perez Liebana, Jens Dieskau, Martin Hunermund, Sanaz Mostaghim, and Simon Lucas. Open loop search for general video game playing. In *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, GECCO '15, pages 337–344, New York, NY, USA, 2015. ACM.

[107] Diego Perez-Liebana, Spyridon Samothrakis, Julian Togelius, Tom Schaul, Simon M Lucas, Adrien Couëtoux, Jerry Lee, Chong-U Lim, and Tommy Thompson. The 2014 general video game playing competition. *IEEE Transactions on Computational Intelligence and AI in Games*, 8(3):229–243, 2016.

[108] Fábio Petrillo, Marcelo Pimenta, Francisco Trindade, and Carlos Dietrich. What went wrong? a survey of problems in game development. *Computers in Entertainment (CIE)*, 7(1):13, 2009.

[109] Robin Potanin. Forces in play: The business and culture of videogame production. In *Proceedings of the 3rd International Conference on Fun and Games*, Fun and Games '10, pages 135–143, New York, NY, USA, 2010. ACM.

[110] Eric D Ragan and John R Goodall. Evaluation methodology for comparing memory and communication of analytic processes in visual analytics. In *Proceedings of the Fifth Workshop on Beyond Time and Errors: Novel Evaluation Methods for Visualization*, pages 27–34. ACM, 2014.

[111] Jochen Renz. Aibirds: The angry birds artificial intelligence competition. In *Twenty-Ninth AAAI Conference on Artificial Intelligence*, 2015.

[112] Cinjon Resnick, Wes Eldridge, David Ha, Denny Britz, Jakob Foerster, Julian Togelius, Kyunghyun Cho, and Joan Bruna. Pommerman: A multi-agent playground. *arXiv preprint arXiv:1809.07124*, 2018.

[113] Tom Schaul. A video game description language for model-based or interactive learning. In *Computational Intelligence in Games (CIG), 2013 IEEE Conference on*, pages 1–8. IEEE, 2013.

[114] Paul Schuytema. *Game design: A practical approach*. Number Sirsi) i9781584504719. Charles River Media, 2007.

[115] Ingo Schwab, Alfred Kobsa, and Ivan Koychev. Learning user interests through positive examples using content analysis and collaborative filtering. *Internal Memo, GMD, St. Augustin, Germany*, 2001.

[116] Noor Shaker, Mohammad Shaker, and Julian Togelius. Ropossum: An authoring tool for designing, optimizing and solving cut the rope levels. In *AIIDE*, 2013.

[117] Ben Shneiderman. Creativity support tools: Accelerating discovery and innovation. *Commun. ACM*, 50(12):20–32, December 2007.

[118] Ben Shneiderman. Creativity support tools: Accelerating discovery and innovation. *Communications of the ACM*, 50(12):20–32, 2007.

[119] Mukherjee Siddharta. The gene: An intimate history. *Scribner, New York*, pages 9–9, 2016.

[120] David Silver, Aja Huang, Chris J Maddison, Arthur Guez, Laurent Sifre, George van den Driessche, Julian Schrittwieser, Ioannis Antonoglou, Veda Panneershelvam, Marc Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *Nature*, 529(7587):484–489, 2016.

[121] Ralph Stuart Singleton. *Film Scheduling, Or, How Long Will it Take to Shoot Your Movie?* Lone Eagle New York, 1991.

[122] Adam M Smith, Erik Andersen, Michael Mateas, and Zoran Popović. A case study of expressively constrainable level design automation tools for a puzzle game. In *Proceedings of the International Conference on the Foundations of Digital Games*, pages 156–163. ACM, 2012.

[123] Adam M Smith, Mark J Nelson, and Michael Mateas. Computational support for play testing game sketches. 2009.

[124] Adam M. Smith, Mark J. Nelson, and Michael Mateas. Prototyping games with biped. In *Proceedings of the Fifth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, AIIDE'09, pages 193–194. AAAI Press, 2009.

[125] Brian A. Smith and Shree K. Nayar. Mining controller inputs to understand gameplay. In *Proceedings of the 29th Annual Symposium on User Interface Software and Technology*, UIST '16, pages 157–168, New York, NY, USA, 2016. ACM.

[126] Gillian Smith, Jim Whitehead, and Michael Mateas. Tanagra: A mixed-initiative level design tool. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, pages 209–216. ACM, 2010.

[127] Adam Summerville, Joe Osborn, Christoffer Holmgård, and Daniel W. Zhang. Mechanics automatically recognized via interactive observation: Jumping. In *Proceedings of the 12th International Conference on the Foundations of Digital Games*, FDG '17, pages 25:1–25:10, New York, NY, USA, 2017. ACM.

[128] Julian Togelius, Sergey Karakovskiy, and Robin Baumgarten. The 2009 mario ai competition. In *IEEE Congress on Evolutionary Computation*, pages 1–8. IEEE, 2010.

[129] Julian Togelius, Georgios N Yannakakis, Kenneth O Stanley, and Cameron Browne. Search-based procedural content generation: A taxonomy and survey.

[130] Guenter Wallner and Simone Kriglstein. Visualizations for retrospective analysis of battles in team-based combat games: A user study. In *Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play*, CHI PLAY '16, pages 22–32, New York, NY, USA, 2016. ACM.

[131] Günter Wallner, Simone Kriglstein, Florian Gnadlinger, Michael Heiml, and Jochen Kranzer. Game user telemetry in practice: A case study. In *Proceedings of the 11th Conference on Advances in Computer Entertainment Technology*, ACE '14, pages 45:1–45:4, New York, NY, USA, 2014. ACM.

[132] Fei-Yue Wang, Jun Jason Zhang, Xinhu Zheng, Xiao Wang, Yong Yuan, Xiaoxiao Dai, Jie Zhang, and Liuqing Yang. Where does alphago go: From church-turing thesis to alphago thesis and beyond. *IEEE/CAA Journal of Automatica Sinica*, 3(2):113–120, 2016.

[133] Michael Washburn Jr, Pavithra Sathiyanarayanan, Meiyappan Nagappan, Thomas Zimmermann, and Christian Bird. What went right and what went wrong: an analysis of 155 postmortems from game development. In *Proceedings of the 38th International Conference on Software Engineering Companion*, pages 280–289. ACM, 2016.

[134] David Watson and Lee Anna Clark. The panas-x: Manual for the positive and negative affect schedule-expanded form. 1999.

[135] Georgios N Yannakakis, Antonios Liapis, and Constantine Alexopoulos. Mixed-initiative co-creativity. In *Proceedings of the 9th Conference on the Foundations of Digital Games*, 2014.

[136] Georgios N. Yannakakis, Antonios Liapis, and Constantine Alexopoulos. Mixed-initiative co-creativity. In *Proceedings of the 9th Conference on the Foundations of Digital Games*, 2014.

[137] Youtube. Youtube, April 2017.

[138] Ping Zhang and Dennis F Galletta. *Human-computer interaction and management information systems: Foundations*, volume 5. ME Sharpe, 2006.

[139] Alexander Zook and Mark O. Riedl. Automatic game design via mechanic generation. In *Proceedings of the Twenty-Eighth AAAI Conference on Artificial Intelligence*, AAAI'14, pages 530–536. AAAI Press, 2014.