

Online Creative Collaborative Content Generation

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

M Charity

May 2024

Online Creative Collaborative Content Generation

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

M Charity

May 2024

Approved:



Department Chair Signature

May 1, 2024

Date

University ID: N17818451

Net ID: mlc761

Approved by the Guidance Committee:

Major: Computer Science



Julian Togelius
Associate Professor of
NYU Tandon School of Engineering

30 April 2024

Date



Michael Cook
Senior Lecturer of
King's College London

27th April 2024

Date



Kate Compton
Director of Play
LEGO Group

29th April 2024

Date



Antonios Liapis
Associate Professor
University of Malta

30 April 2024

Date



Wesley Kerr
Head of Tech Research at
Riot Games

29th April 2024

Date

Microfilm or other copies of this dissertation are obtainable from

UMI Dissertation Publishing

ProQuest CSA

789 E. Eisenhower Parkway

P.O. Box 1346

Ann Arbor, MI 48106-1346

Vita

M Charity was born in Richmond, VA on April 3rd, 1998. They graduated from the Center of Information Technology at Deep Run High School in 2016. They earned a Bachelor of Science in Computer Science at Virginia Commonwealth University (VCU) in 2019. Since the age of 11, they have been designing and programming video games and AI systems.

In 2015, their senior year of high school, they met Dr. Bridget McInnes of the Natural Language Processing (NLP) Lab at VCU. She was a mentor to them for a senior capstone project which focused on making a chatbot capable of emulating personalities named GRAM. After graduation, M continued to be mentored by Dr. McInnes and was introduced to the world of research. Throughout college, they continued to explore both NLP under Dr. McInnes, virtual reality with Dr. Victoria Interrante at the University of Minnesota, Twin Cities, and virtual reality game design applied to physical therapy with Dr. James Thomas at the moVR Lab at VCU Medical School. Through both research, academic, and personal work, they continued to design games, AI systems, and web application.

In 2019, M joined as a PhD student in the Game Innovation Lab with Dr. Julian Togelius as their advisor. Building off of their childhood and college interests, they focused their research on combining AI system, game design, and web applications as applied to creative content generation. They have published papers in international conferences and workshops such as IEEE Conference on Games (CoG), the International Conference on the Foundations of Digital Games (FDG), the Procedural Content Generation (PCG) workshop, the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment (AIIDE), the World Congress on Computational Intelligence (WCCI), and the journal IEEE Transactions on Games (TOG.) They have been a co-organizer for the Experimental AI in Games workshop and PCG workshop. They have also been invited speakers to events such as Games for Change, Game Devs of Color, PapersWeLove, the ALOE workshop hosted at NeurIPS, and MIT / Foster America Youth Tech Workshop. During their time as a PhD student, M was a teaching assistant for the Artificial Intelligence, AI for Games, and Game Design classes at NYU and co-taught as an adjunct professor with Dr. Michael Green for the Game Design class. Funding for tuition, travel, and personal expenses was provided by various sources, including the GAANN Fellowship, the National Science Foundation (award number 1956200,) Dr. Michael Cook, the Tandon School Fellowship, and the Game Innovation Lab.

Acknowledgements

First I would like to thank my advisor and mentor, Dr. Julian Togelius, for guiding and encouraging me on this journey and having faith in me – I hope I wasn’t too stubborn of a student. I’d also like to thank my past mentors Ms. Maggie Mustian, Ms. Lynne Norris, Dr. Joseph Palen, Dr. Bridget McInnes, Professor Debra Duke, Dr. Craig Larson, Dr. Victoria Interrante, and Dr. James Thomas – all of whom introduced me to the world of science and research. I’d like to thank my current mentors and committee members whose works inspired my own research direction Dr. Michael Cook, Dr. Kate Compton, Dr. Antonios Liapis, and Dr. Wesley Kerr.

Of course, I’d like to thank my labmates and collaborators through the years of my PhD. Some I spent more time with than others, but each person has left a large impact on my life and my career. In estimated reverse chronological order from when I met them: Rolf Piepenbrink, Asad Anjum, Justin Li, Daphne Cornelisse, Dolphin Nie, In-Chang Baek, Michael Middleton, Shouren Wang, Sudipta Banerjee, Professor Rafael Bidarra, Yash Bhartia, Roman Negri, Mayu Wilson, Steven Lee, Abhi Sen, Maria Edwards, Matt Siper, Anubhav Jain, Isha Dave, Max Sirotin, Rachel Ombok, Lisa Soros, Tiago Machado, Tae Jong, Aaron Dharna, Ruben Torrado, Ming Jin, Rodrigo Canaan, Chang Ye, Philip Bontrager, Christopher “Widget” DiMauro, and Ahmed Khalifa. I’d like to thank my co-teacher and collaborator Dr. Michael Green – you paved the way for my PhD. I want to thank the 4 PhD guys of the lab – Sam Earle, Graham Todd, Zehua Jiang, and Tim Merino – I’ll miss our late night lab parties and jam sessions. I’d especially like to thank Debosmita Bhaumik and Catalina Jaramillo – getting lunch and playing board games with you both were some of my favorite memories in the lab – as well as Dipika Rajesh and Noelle Law – I can’t wait for you both to start your PhDs and we can hang out doing research all over again.

I thank my friends from back home in Virginia: Michael, Rachel, Sam, Tammi, Lee, Mel, Chelsea, and Miles – and the friends in New York and those I met abroad: Laurel, Veronika, Lyric, Arshiya, Monica, Lexa, Sylvia, Sara, Emily, Nat, Adam, Amiya, Raynah, Nayan, Kamali, Tatiana, David, Blaer, Matt N., Kaya, Soo-in, Justin, Matt F., Omar, Younès, Florence, Alberto, Christopher, and Emma – you all added so much life to my PhD and made it such an unforgettable experience.

Lastly, I’d like to thank my sister, Faith, who has watched me grow into the person I am today and is my rock and emotional support. Finally, my cat, Effy – I’m sorry I was always home so late, it was worth it trust me.

M Charity
May 2024

To all the friends I made along the way – both human and AI

ABSTRACT**Online Creative Collaborative Content Generation**

by

M Charity**Advisor: Prof. Julian Togelius, Ph.D.****Submitted in Partial Fulfillment of the Requirements for
the Degree of Doctor of Philosophy (Computer Science)****May 2024**

This thesis explores the integration of artificial intelligence (AI) systems into the collaborative creative processes of game content creation through online interfaces. The systems described here focus on addressing many challenges with content generation as a whole, including generating diverse game content, designing open-ended environments and AI challenges, and exploring the usage of collaboratively made online data for AI research. With the development of these online systems and websites, people can collaboratively generate game content such as game levels, sprites, 3-dimensional structures, and even other AI agents and game environments. This thesis details evaluations and insights gained from user studies conducted on these systems that highlight the strengths and weaknesses of these AI-assisting online systems and how they can be improved to encourage future human-AI collaborations in creative domains.

Contents

Vita	iv
Acknowledgements	v
Abstract	viii
List of Figures	xix
List of Tables	xxii
1 Introduction	1
1.1 Thesis Statement	3
1.2 Contributions	3
1.3 List of Publications	4
1.4 Outline	7
1.5 Notes on Pronouns	9
2 Background	11
2.1 Elements of Games	11
2.2 Abstract Design	13
2.3 Artificial Intelligence	14
2.4 Procedural Content Generation	16
2.5 Evolutionary Algorithms	20
2.6 Mixed Initiative Systems	23
2.7 Crowd-sourced User Content	24
3 Automatically Generating Diverse Game Levels	26
3.1 General Video Game Artificial Intelligence (GVG-AI)	27

3.2	Mech-Elites System	29
3.3	Experiment Setup	32
3.4	Results	33
3.5	Summary	36
3.6	Reflection	37
3.7	Takeaways	38
4	Mixed-Initiative Mechanic Diversity	39
4.1	Baba is You	40
4.2	System Description	42
4.3	Baba is Y'all version 1	52
4.4	Baba is Y'all version 2	52
4.5	User Studies	53
4.6	Summary	59
4.7	Reflection	59
4.8	Takeaways	62
5	Crowd-sourced Data for Game-Playing Agent Training	63
5.1	Competition Framework	64
5.2	Competition Website	67
5.3	Results	69
5.4	Summary	70
5.5	Reflection	70
5.6	Takeaways	71
6	Crowd-sourcing Aesthetic Content Generation and Evaluation	72
6.1	Twitter Bots and the Twitter API	73
6.2	Aesthetic Bot System	74
6.3	Results	79
6.4	Discussion	82
6.5	Summary	84

6.6	Reflection	85
6.7	Takeaways	87
7	Accessible Text-to-Image Generative Models from Crowd-sourced Data	88
7.1	Model Description	89
7.2	Evaluation	91
7.3	Application	92
7.4	Summary	94
7.5	Reflection	94
7.6	Takeaways	95
8	User-Limited Collaboration for Aesthetic Content Evolution	97
8.1	System Description	98
8.2	User Study	101
8.3	Results	102
8.4	Summary	105
8.5	Reflection	105
8.6	Takeaways	107
9	Recommendation and Generation of Abstract Content	108
9.1	System Description	109
9.2	Experiment	114
9.3	Results	117
9.4	Summary	121
9.5	Reflection	121
9.6	Takeaways	122
10	Large Language Models as Game Design Partners	124
10.1	Interaction Methodology	124
10.2	Space Shooter Design	126

10.3 Platformer Design	129
10.4 Roguelike Design	131
10.5 User Study	134
10.6 Results	136
10.7 Discussion	144
10.8 Summary	145
10.9 Reflection	146
10.10 Takeaways	147
11 Evolving Abstract Multi-Agent AI Environments	149
11.1 Amorphous Fortress Engine	150
11.2 Hillclimber Experiment	157
11.3 Quality Diversity Experiment	163
11.4 Summary	170
11.5 Reflection	171
11.6 Takeaways	171
12 Collaborative Abstract Game and AI Environment Design	173
12.1 System Description	174
12.2 Short Study	184
12.3 Future Work	185
12.4 Summary	187
12.5 Reflection	188
12.6 Takeaways	189
13 Conclusions	191
13.1 Online AI Systems	191
13.2 The User and the AI	193
13.3 Future Work	194
13.4 Reflection	195

List of Figures

1.1	An outline of the projects and how they lead into each other. Each project is color coded based on their contribution to the thesis statement	9
2.1	The community-based ZZT game design process showing (a) the base action-adventure ZZT game; (b) the ZZT game editor; (c) the Museum of ZZT BBS website page; (d) a user-created simulation role-playing game using the ZZT engine. The design of the ZZT engine and BBS inspired many of the collaborative AI projects in this thesis.	12
2.2	A screenshot of Spelunky, a roguelike platformer where the sections of the level are randomly generated and populated with items and enemies using procedural content generation (PCG)	16
2.3	Example generation from Kate Compton’s “So you want to build a generator...” Tumblr post coining the “10,000 Bowls of Oatmeal” problem. While each flower generation is computationally “unique”, to a user, they look very similar - hence leading to the 10000 bowls of oatmeal problem.	19
2.4	Screenshot of the Picbreeder online interactive evolver tool. Users can select and evolve AI generated images with an open-ended objective.	21

3.1	An example of an optimum MAP-Elites matrix for the Zelda game mechanics "get key" and "kill enemy".	31
3.2	The percentage of elites that contain a specific mechanic for each game from the Mech-Elites system.	35
3.3	Normalized number of elite cells across generations for each evaluated game.	36
4.1	Simple 'Baba is You' levels with different satisfied starting rules.	40
4.2	System flowchart of the Baba is Y'all website system	43
4.3	Editor Module	44
4.4	Mutator Module	45
4.5	A sample population selecting a map with the best fitness value during an evolution iteration	46
4.6	Objective Module	47
4.7	Map Module	48
4.8	Rating module in the Baba is Y'all version 2 site	49
4.9	User profile tab on the main screen of the Baba is Y'all version 2 site	51
4.10	Baba is Y'all Version 1 Main Screen (from April 2020)	52
4.11	Baba is Y'all Version 2 Main Screen (as of September 2021)	53
4.12	Survey results for users' reports on the features they used	55
4.13	Sample levels generated for the system. The left column is user generated levels, the middle column is evolver module levels, and the right column is mixed-initiative user and evolver levels	56
4.14	Distribution of rule types in the Baba is Y'all v2 system by authorship	57
5.1	Example of the ASCII representation of the Baba is Y'all level map as interpreted by the framework engine	64
5.2	The evaluator screen of the offline Keke AI Competition offline interface.	65

5.3	A screenshot of an example profile page on the Keke AI Competition website. Here, teams can view such as statuses such as rank, submission timestamp, submission count, and more.	67
5.4	Sample levels from the evaluation set of the Keke AI Competition - each designed to exploit different agent search algorithms	69
6.1	The tilesets used in the Aesthetic Bot system. The tilesets are referred to as such from left to right: Zelda, Pokemon, Among Us, Pacman, Dungeon	74
6.2	Diagram of the APM learning process on the sliding window tile patterns and vote percentages from Twitter.	75
6.3	Evolution and learning pipeline of the Aesthetic Bot system	76
6.4	Sample Twitter post of the Aesthetic Bot system in with a user-made and artificially generated map being compared in a poll	78
6.5	Level editor for the Aesthetic Bot system where users can design and submit their own maps to be used in future polls.	79
6.6	Percentage of votes received for the user maps.	80
6.7	Different groups of user made maps that were submitted to the database. From top to bottom are “level”-like maps, “hacked” maps, and “non-ludic” maps.	83
6.8	Some examples of user maps influencing future map generations. The bot tries to recreate the more complex styles and designs in the user maps.	84
6.9	Generated maps for the Pokemon tileset over time. The sprite composition and tile patterns shifts towards different ”types” of maps.	84
7.1	Diagram of the \$5 Model architecture	90
7.2	Pokemon-tileset maps generated from unseen text prompts using the \$5 model	92
7.3	Model output vector interpolation between sprites	93

7.4 Screenshots of the demo game utilizing the \$5 dollar model. On the left, the user inputs the text for the next map to be shown. On the right, the map is generated from the description showing “An island of rocks in the ocean”	93
8.1 Screenshot of the interactive evolutionary Minecraft house generator web application with 6 structure options the user can select to evolve.	98
8.2 A diagram of the preprocessing and training steps for the system . .	100
8.3 A diagram of the evolutionary process for the system	101
8.4 Open-ended houses evolved by participants labeled a) ‘a more expensive house’ b) ‘a tower with a cloud idk’ c) ‘A Mario game level’, and d) ‘A single block’	103
8.5 Results from the Likert-scale user experience questions.	104
9.1 Sample game features from the Castle Crashers game description extracted using a part-of-speech grammar	110
9.2 Diagram of the conceptual feature recommendation process feeding into the generator model for training	111
9.3 Sample data retrieved from the ‘CapableOf’ and ‘UsedFor’ lists of the 3 words ‘pirate’, ‘bear’, and ‘paper’	113
9.4 A screenshot of the Google Form shown to users for prompt #5. Each feature set is anonymized with A, B, or C.	116
9.5 Results from the 6 voting polls - blue segments for the real games and orange segments for the fake games.	118

10.1 Example of the freeze ability being implemented for the space shooter genre game with guidance from ChatGPT. The freeze ability is (A) first suggested by ChatGPT from an initial prompt. (B) The user then asks ChatGPT for help with implementing the code in Unity engine in C#, (E) the output code from ChatGPT is attempted to be implemented 3 times by the user until it is successfully integrated into the game without error. The user also asks for (G) explanations about the generated code and (C) further suggestions to add game feel	126
10.2 Screen-captures of the 3 shooter games created for the experiment.	126
10.3 Screen-captures of the 3 platformer games created for the experiment.	129
10.4 Screen-captures of the 3 roguelike games created for the experiment.	131
10.5 Screen-capture of the website where users could play each of the 9 games and provide feedback.	134
10.6 Page of the anonymized base platformer game shown to participants with instructions for how to play the game below.	135
10.7 A poster used to recruit participants for this study. The left image shows the AI generated inspiration and the right shows the final human-edited version that was released.	136
11.1 Screenshot of a generated environment in the Amorphous Fortress framework containing multiple instances of autonomous agents	151
11.2 The ‘dork’ entity (highlighted in green on the map on the left) and its FSM definition (right window) with the active node (move) and most recently transitioned conditional edge (step 3) outlined in green.	154
11.3 Two example agent interactions from Zelda: Tears of the Kingdom remade in the Amorphous Fortress framework	157
11.4 Best and current fitness scores of the 5 experiment trials (averaged and min/max range)	161
11.5 Initial and final renderings of the fortress for 3 seeds	162

11.6 Diagram of the MAP-Elites algorithm applied to the Amorphous Fortress QD experiment	164
11.7 Diagram of the calculations for MAP-Elites BCs, fitness values, and entropy value	167
11.8 Results of the AF-QD experiment: exploring number of instances at the end of the simulation vs. total number of nodes . The graph on the left shows the heatmap with respect to the fitness measurement—the proportion of FSMs explored in the fortresses. The graph on the right shows the same archive of fortresses measured with the heatmap showing the measure of FSM entropy sizes.	168
11.9 Growth of the population number for each entity class in the MAP-Elites archive. The vertical dashed blue line is the number of iterations the fortress was simulated for during evolution. The cell with the highest entropy of entity class definitions achieves an ecological equilibrium. This behavior is distinct from the extremes of the archive, which either overpopulate or rarely add any additional entities.	169
12.1 The entity editor screen for Amorphous Fortress Online labeled as follows: A) FSM canvas visual editor; B) Entity selection dropdown; C) Canvas node list; D) Name and character input area; E) Entity node and edge list edit area; F) Fortress entity list	175
12.2 The fortress editor screen for Amorphous Fortress Online labeled as follows: A) Fortress name input area; B) Fortress placement canvas C) Entity palette; D) Font selection; E) Notes input area; F) Seed input area	176
12.3 The text editor screen for Amorphous Fortress Online labeled as follows: A) Fortress text editor area; B) Compiler console; C) Raw text paste area; D) Validation check button; E) Save fortress button	177

12.4 The fortress play screen for Amorphous Fortress Online labeled as follows: A) Name and author of the fortress; B) Main fortress view area; C) Fortress simulation log; D) Simulation menu bar; E) Play screen settings area	178
12.5 The X-Ray feature (right) highlighting the Knight entity K (left) in the Thief Castle fortress (#36.) The green outlined FSM components indicate active or recently activated behavior logic. Currently its $1:0-1: \text{within } 3$ T edge.	180
12.6 Main screen fortress selection	181
12.7 Fortress info viewer	182
12.8 When logged in, a user can save an entity from one fortress to use in a new fortress with the backpack feature.	183
12.9 Frequency of entity action nodes	184
13.1 Thanks for reading! Let's make more online AI systems :D	196

List of Tables

3.1	Constrained MAP-Elites dimensions for the GVG-AI game Zelda	33
3.2	Constrained MAP-Elites dimensions for the GVG-AI game Solarfox	33
3.3	Constrained MAP-Elites dimensions for the GVG-AI game Plants	34
3.4	Constrained MAP-Elites dimensions for the GVG-AI game RealPortals	34
4.1	Baba is Y'all Ruleset	42
4.2	Level statistic between selected questionnaire responses and submitted levels	57
6.1	Average prediction values for each model trained on the pairing data ([U]ntrained, [P]retrained, [2 W]eeks, [6 W]eeks) for each tileset. The values indicate the percentage of votes the maps are expected to receive.	81
6.2	Average percent error on the test dataset for the original APM network architecture and the POST network architecture. The APM slightly outperforms the POST network for poll prediction accuracy.	82
8.1	Table of statements used in the Likert-scale experience reporting of the survey.	104
9.1	Similarity matrix for the terms 'knight', 'castle', and 'assassins' compared to some game entities from 'Nidhogg'.	111

9.2 Table of hand-authored and model-generated features for each game prompt.	114
9.3 Table of ChatGPT responses given the same prompts used in the user study	120
10.1 Demographic survey questions asked in the user study separated by general topic.	137
10.2 Scores for each game and genre using the formula S_G and category criteria Models with the highest score are colored in green, second highest in yellow, and lowest in red. (AI = ChatGPT)	139
10.3 Statistic Tests on the Space Shooter Genre games based on all participant responses (KW = Kruskal-Wallis Test Statistic; F = Friedman)	140
10.4 Statistic Tests on the Platformer Genre games based on all participant responses (KW = Kruskal-Wallis Test Statistic; F = Friedman)	140
10.5 Statistic Tests on the Roguelike Genre games based on all participant responses (KW = Kruskal-Wallis Test Statistic; F = Friedman)	140
10.6 Statistic Tests on the Space Shooter Genre games based on the designer group participant responses (KW = Kruskal-Wallis Test Statistic; F = Friedman)	141
10.7 Statistic Tests on the Platformer Genre games based on the designer group participant responses (KW = Kruskal-Wallis Test Statistic; F = Friedman)	141
10.8 Statistic Tests on the Roguelike Genre games based on the designer group participant responses (KW = Kruskal-Wallis Test Statistic; F = Friedman)	141
10.9 Statistic Tests on the Space Shooter Genre games based on the non-designer group participant responses (KW = Kruskal-Wallis Test Statistic; F = Friedman)	142

10.10 Statistic Tests on the Platformer Genre games based on the non-designer group participant responses (KW = Kruskal-Wallis Test Statistic; F = Friedman)	143
10.11 Statistic Tests on the Roguelike Genre games based on the non-designer group participant responses (KW = Kruskal-Wallis Test Statistic; F = Friedman)	143
11.1 List of action nodes to define the agents	152
11.2 Entity FSM conditional edge definitions (ordered by least to greatest priority)	153
11.3 List of configuration parameters that can be set for the Amorphous Fortress Python framework	156
12.1 Control scheme options for player controlled entities	179

Chapter 1

Introduction

I love making things. I think it's even cooler when I can make a system that procedurally makes a lot of things for me. I also love when I see a system making things behind the scenes for a game. I see these systems most commonly in the roguelike video game genre. They procedurally generate levels and items, or simulation environments where the non-playable characters walk around and interact with my player character. This artificially-made content and these interactions felt so real and unique - as if they were designed just for my play experience and could never be replicated the exact same way again. As such, I wanted to explore how to design systems like them to incorporate into my own games.

Upon discovering the research area of procedural content generation (PCG), I learned that there was so much more to these systems than what I had only seen in the video games I played. While many of these experimental systems focused on level generation, I thought it was interesting that there was a lack of user-integration with the content design process overall. Some of the generated content felt too artificial at times, and could be improved if a user was able to make minor edits or provide feedback to the generator system. Mixed-initiative systems and other collaborative PCG systems also existed in the field. They were intended to be a remedy to the artificial quality of the content produced, but I believed these systems were still too limiting. They were only useful to the domain

they were designed for or constricted by the continuous interaction and refinement that was required to make the system effective. Unfortunately, many of these experimental systems had also not been implemented into practice for actual game development.

Artificially intelligent (AI) systems are becoming more and more prominent in our daily lives, mostly as personal assistants in our smart-phones and as realistic conversationalist like with ChatGPT¹. Content generation in particular has become more prominent with text-to-image generation tools such as with StableDiffusion and DALL-E. These systems are all accessible from a browser, whether on a desktop computer, a laptop, or a smartphone. While they use large compute and processing units server-side, they still allow everyday users without access to this processing power to contribute and create new content through their website. However, these large systems in particular, while general, are not catered specifically to game design and limits their application and utility for creative content generation.

AI could help with game development as well, without needing large super-computers or millions of samples of training data, and should be able to be used and modified without a developer needing a strong machine learning or technical background. I believe AI should be accessible to everyone and they can be valuable design partners for our creative endeavors. I want to make PCG systems that passionate game designers like myself who work at a small-scale independent or hobbyist level could use to help with their game design process. I want to develop AI that works with human game designers as collaborators and design partners to make these games as if they were apart of the development team. I also want to make AI systems which anyone can use, even if they lack the resources.

This thesis proposes experimental work focusing on AI systems that generate content meant for game designers and developers by working collaboratively with humans to create said content. Many of these systems focus on design using online interfaces to improve the collaborative creative feedback loop between human and

¹At the time of writing, ChatGPT's most freely available version used GPT-3.5

AI. Through this study, I hope to lay the foundation for future collaborative AI systems and find ways of encouraging human-AI interactions for game development.

1.1 Thesis Statement

Artificial intelligence as a tool is becoming more and more prevalent in our daily lives. AI can even be used to help us design creative content more effectively. However, interacting with this tool and communicating our needs and creative intent is still a challenge. I propose the following thesis statement which serves as the motivating factor for this dissertation and the research work presented:

When applied to creative domains, online systems allow users to more effectively design, share, and evaluate content with AI tools.

1.2 Contributions

The systems described in this thesis document look to address many problems with content generation AI tools – some even highlighted newer problems after they were developed. However, these systems contributed many insights to the field of content generation for games, mixed-initiative systems, and AI research. The most notable of these contributions are the following:

- **Collaboratively generating diverse game content online:** With AI facilitation, game content such as levels and entire simulation games can be developed and shared with other users with online systems. This thesis describes the development of these systems, their usage, and how the collaborative content created has filled missing elements of game assets that could not otherwise be created solely by humans or AI.
- **Designing online systems for open-ended environments and problems:** These systems look to explore how users can help AI models learn to evaluate and generate content based on metrics that are difficult to measure or

define – i.e. open-ended challenges. Examples of these open-ended problems include aesthetic evaluation and thematic relations between entities, while the open-ended environments include simulation environments that allow for emergent behavior from either the AI or the player.

- **Utilizing online data to further AI research:** While many of these systems are still active and regularly collecting data, the data has also been used for other systems such as training data for generative models and for AI competitions. This has simultaneously created new challenges for AI research while creating a source of novel datasets that are both human and AI made.
- **Gathering feedback from designers for future AI-assisted tool development:** We study the effects that these AI tools have had on users' design processes and if and how these tools have enhanced their creative outputs in level design and game design. These studies serve as insight into how to improve collaborative AI systems and how to make them more accessible for users who are not as familiar with AI tools.

1.3 List of Publications

The work in this thesis discusses the systems created and described in these 12 peer-reviewed works. All of the following publications have been accepted to international conferences in games, AI, and game AI research venues (with the exception of the Amorphous Fortress Online work – which at the original time of writing, was under review.) They are listed based on how they are presented in the chapters of the thesis:

- M Charity, Michael Cerny Green, Ahmed Khalifa, and Julian Togelius. Mechanic Elites: Illuminating the Mechanic Space of GVG-AI. In Proceedings of the *15th International Conference on the Foundations of Digital Games*, 2020.

- M Charity, Ahmed Khalifa, and Julian Togelius. Baba is Y'all: Collaborative Mixed-Initiative Level Design. In the *IEEE Conference on Games*, 2020.
- M Charity, Isha Dave, Ahmed Khalifa, and Julian Togelius. Baba is Y'all 2.0: Design and Investigation of a Collaborative Mixed-Initiative System. In *IEEE Transactions on Games*, 2022.
- M Charity, and Julian Togelius. Keke AI Competition: Solving puzzle levels in a dynamically changing mechanic space. In the *IEEE Conference on Games*, 2022.
- M Charity and Julian Togelius. Aesthetic Bot: Interactively Evolving Game Maps on Twitter. In Proceedings of the *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2022.
- Timothy Merino, M Charity, and Julian Togelius. Interactive Latent Variable Evolution for the Generation of Minecraft Structures. In the Proceedings of the *18th International Conference on the Foundations of Digital Games*, 2023.
- Timothy Merino, Roman Negri, Dipika Rajesh, M Charity, and Julian Togelius. The 5 Dollar Model: Generating Game Maps and Sprites from Sentence Encodings. In the Proceedings of the *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2023.
- M Charity, Yash Bhartia, Daniel Zhang, Ahmed Khalifa, and Julian Togelius. A Preliminary Study on a Conceptual Game Feature Generation and Recommendation System. In the Proceedings of the *IEEE Conference on Games*, 2023.
- Asad Anjum, Yuting Li, Noelle Law, M Charity, and Julian Togelius. The Ink Splotch Effect: A Case Study on ChatGPT as a Co-Creative Game Designer. In the Proceedings of the *19th International Conference on the Foundations of Digital Games*, 2024.

- M Charity, Dipika Rajesh, Sam Earle, and Julian Togelius. Amorphous Fortress: Observing Emergent Behavior in Multi-Agent FSMs. Accepted to *Artificial Life Conference - ALIFE for and from Games Workshop*, 2023.
- Sam Earle, M Charity, Mayu Wilson, Dipika Rajesh, and Julian Togelius. Quality Diversity in the Amorphous Fortress (QD-AF): Evolving for Complexity in 0-Player Games. Accepted to *NeurIPS 2023 - Agent Learning in Open Endedness Workshop*, 2023.
- M Charity, Mayu Wilson, Steven Lee, Dipika Rajesh, Sam Earle, and Julian Togelius. Amorphous Fortress Online: Collaboratively Designing Open-Ended Multi-Agent AI and Game Environments. Submitted to *IEEE Conference on Games*, 2024.

The following additional 5 works are not discussed in this dissertation. However, they were also completed during my PhD and peer-reviewed and accepted to various international conferences. They are listed in chronological order.

- M Charity, Dipika Rajesh, Rachel Ombok, and Lisa B. Soros. Say “Sul Sul!” to SimSim, a Sims-inspired Platform for Sandbox Game AI. In Proceedings of the *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2020.
- Catalina Jaramillo, M Charity, Rodrigo Canaan, and Julian Togelius. Word Autobots: Using Transformers for Word Association in the game Codenames. In Proceedings of the *AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, 2020.
- Michael Cerny Green, Ahmed Khalifa, M Charity, Debosmita Bhaumik, and Julian Togelius. Predicting Personas Using Mechanic Frequencies and Game State Traces. In *IEEE Congress on Evolutionary Computation (CEC)*, 2022.
- Michael Cerny Green, Ahmed Khalifa, M Charity, and Julian Togelius. Persona-driven Dominant/Submissive Map (PDSM) Generation for Tutorials.

In Proceedings of the *17th International Conference on the Foundations of Digital Games*, 2022.

- M Charity, Nasir Memon, Zehua Jiang, Abhi Sen, and Julian Togelius. Diversity and Novelty MasterPrints: Generating Multiple DeepMasterPrints for Increased User Coverage. In *International Conference of the Biometrics Special Interest Group (BIOSIG)*, 2022.

1.4 Outline

The chapters of this work focus on the systems that contribute to the thesis statement. Each chapter describes 1-2 systems that are mostly chronological.

- **Chapter 3: Automatically Generating Diverse Game Levels** describes the Mech-Elites system – a quality diversity level generator. While not an online system, the results of this work act as a prologue and igniting point for the later projects.
- **Chapter 4: Mixed-Initiative Mechanic Diversity** describes the Baba is Y'all system – an online mixed-initiative level editor that collaboratively designed levels for the puzzle game Baba is You. This work highlighted many ongoing challenges for AI-assisting tools that led to the development of the projects in the rest of this thesis.
- **Chapter 5: Crowd-sourced Data for Game-Playing Agent Training** describes the Keke AI Competition – a game-playing AI competition that looked to improve the agent solver in the Baba is Y'all system by using the co-designed levels submitted to the site.
- **Chapter 6: Crowd-sourcing Aesthetic Content Generation and Evaluation** describes the Aesthetic Bot system – a Twitter bot that looked to solve the issue of user-fatigue as well as attempt to train a model on aesthetic-based content generation.

- **Chapter 7: Accessible Text-to-Image Generative Models from Crowd-sourced Data** describes the \$5 Model – a relatively small text-to-image generative AI model that is trained on the evaluations and crowd-sourced labels and map designs of the Aesthetic Bot system.
- **Chapter 8: User-Limited Collaboration for Aesthetic Content Evolution** describes the Minecraft House Interactive Evolver system – an online tool that evolves Minecraft houses based on a user’s aesthetic preference. A user study was also conducted to study general opinions on AI collaboration.
- **Chapter 9: Recommendation and Generation of Abstract Content** describes a recommendation and generator model for game features. A user study was conducted to study user opinions on AI content suggestions and output.
- **Chapter 10: Large Language Models as Game Design Partners** describes the Ink-Splotch experiment. This study evaluated the internal and external effects of AI tools – particularly ChatGPT – as a collaborative game design partner.
- **Chapter 11: Evolving Abstract Multi-Agent AI Environments** describes the Amorphous Fortress framework – an abstract multi-agent simulation system. This system evolved environments with agents that demonstrated novel interactive behaviors. While also not an online system, it serves as a prologue to the following chapter’s system.
- **Chapter 12: Collaborative Abstract Game and AI Environment Design** describes the Amorphous Fortress Online system. Like the Baba is Y’all system, this tool allowed users to collaboratively design multi-agent environments but in an open-ended domain. This system serves as a starting point for future AI research in recommendation systems, generative models, and AI competitions.

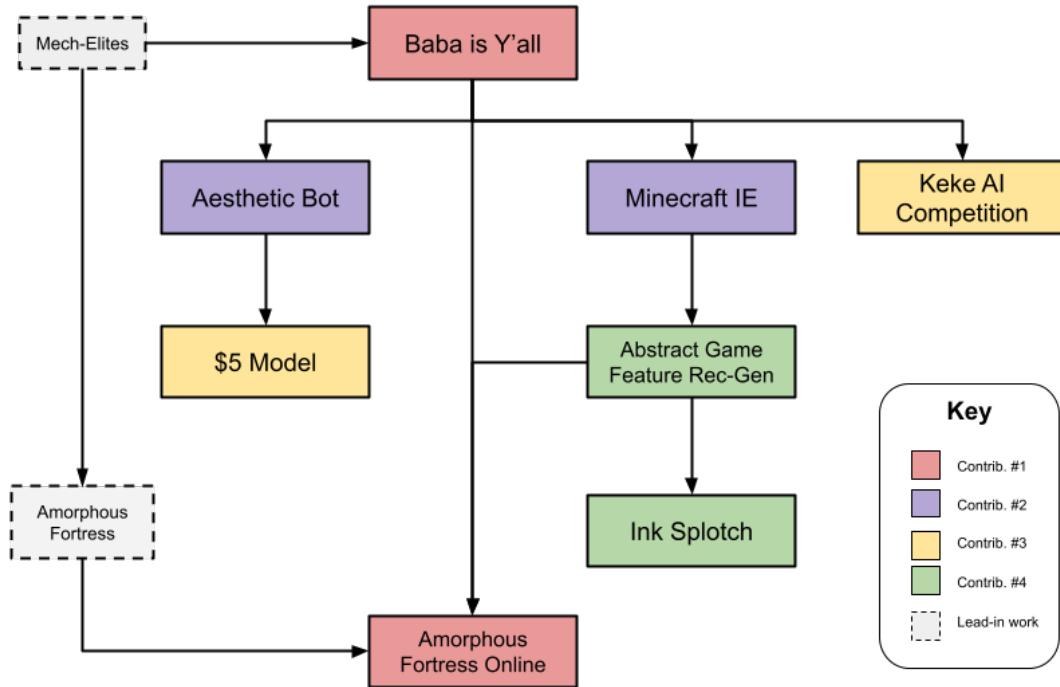


Figure 1.1: An outline of the projects and how they lead into each other. Each project is color coded based on their contribution to the thesis statement

- **Chapter 13: Conclusions** summarizes the thesis, reflects on the contributions of the online systems, and proposes future work based on the challenges presented by the systems and the user study results described in previous chapters.

1.5 Notes on Pronouns

Although I am the first author of many of the works discussed in this dissertation document, I could not have done any of these projects without the collaborative effort of my co-authors and the guidance of my mentor. Throughout the rest of this thesis, the pronoun “we”, “us”, or “our” will be used in favor of “I” or “my” when describing projects to refer to the effort of all researchers involved. When referring to the pronouns of a *player* or *user*, unless referred to as such, this dissertation will use the pronouns “they” and “their” for gender inclusivity. When I refer to *I*, I mean solely myself, M Charity. When I refer to *you*, the reader, I will address

“you” appropriately.

Chapter 2

Background

This thesis covers many different topics in games, AI, and games AI research. The following sections are meant to provide background and explanations on as many relevant topics that the research works and systems in this document either mention or are based on. Many references to related systems in the research field and game design field as a whole are also provided as context.

2.1 Elements of Games

2.1.1 Game Mechanics, Rules, and Features

“Game rules”, as defined by Sicart, describe how the game is played and controlled while “game mechanics” are methods agents take to invoke those game rules (e.g. the player touches the flagpole to activate the game rule “win the level” in Super Mario World). Mechanics typically include the actions that a player can perform in the game (i.e. ‘run’, ‘jump’, ‘shoot’) [109].

Mullich defines “game features” more broadly - which represent descriptive aspects of a game’s design, art, or technical capabilities to explain the overall game experience [86]. These features are typically found in the game’s summary text and use higher-level language to communicate the most interesting aspects of the game without giving away too many gameplay details. There is a lack of research work

that look to design games from abstract game features; works that focus on the design of the game in its early concept stage. Having a system that can interact collaboratively with a user during this phase of game design could be beneficial and create a new path of research for AI-assisted game design.

As Togelius et al. [126] demonstrate, games can be evolved by exploring and mutating their rules. Previous works such as ANGELINA [32] and Sturgeon-MK3 [35] also use game rule manipulation to create and recreate games and environments without having to directly modify the underlying source code of an engine or system. Board games also benefit from the automated creative process and can result in interesting and aesthetically satisfying games without any human intervention [14]. Previous works investigated automatically discovering and extracting game rules and mechanics for generating tutorial instructions [49], levels [34], or new games [59, 130, 144].

2.1.2 Open-Endedness in Games

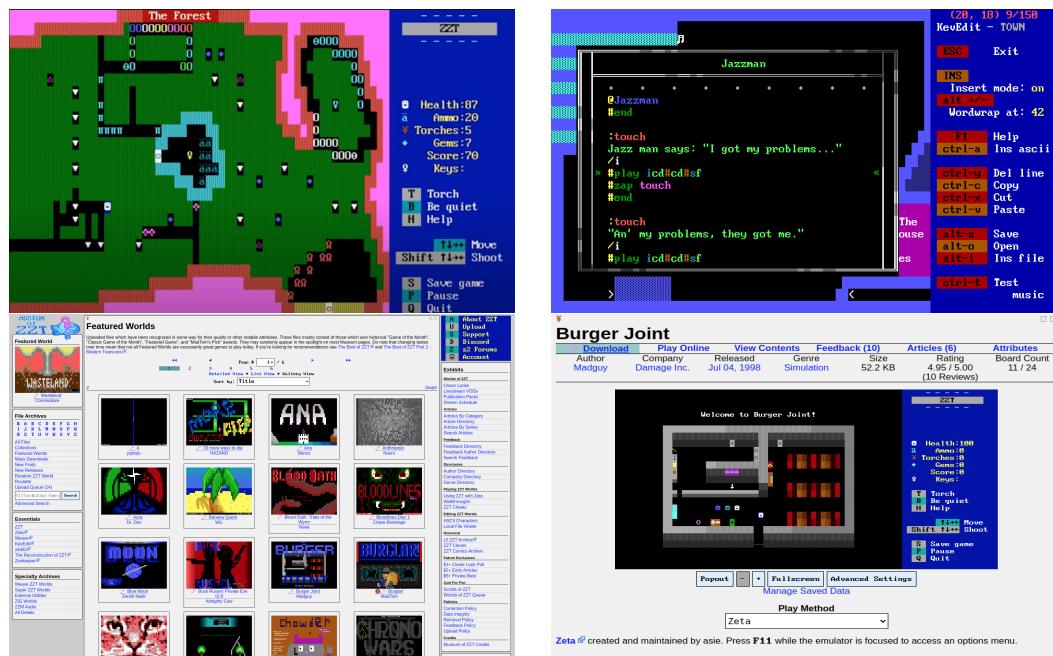


Figure 2.1: The community-based ZZT game design process showing (a) the base action-adventure ZZT game; (b) the ZZT game editor; (c) the Museum of ZZT BBS website page; (d) a user-created simulation role-playing game using the ZZT engine. The design of the ZZT engine and BBS inspired many of the collaborative AI projects in this thesis.

Open-endedness and creativity in game design also makes use of community-made content generation to engage players with the system. However, with open-ended games, there is no limit to the domain or genre. The custom object-oriented programming language and bulletin board system (BBS) of the shareware game *ZZT* (see Figure 2.1) is an example of how players can remix and design the grid-world and ASCII characters to make new games including puzzle games, RPGs, and visual novels [5]. *LittleBigPlanet* is a platformer game designed around crowd-sourced content generation where users can share asset creations and upload their own levels using the game’s base mechanics. The sequel game, *LittleBigPlanet2*, expanded the genre domains outside of platformers including racing games, arcade multiplayer games, and even movies. This later iteration of the series added more complex mechanics such as simple AI behavior programming through the Sockbots and logic gate programming for switches and other mechanisms.

2.2 Abstract Design

2.2.1 Aesthetic Evaluation of Content

The Stanford Encyclopedia of Philosophy describes aesthetics as a subjective taste in a sense of beauty and ugliness. There is no hard truth to what aesthetic is and they do not necessarily need to be related to visual tastes [140]. David and Globe note that the visual aesthetics of a system, such as one used for user interfaces and human computer interaction, can influence a user’s perception of ease of use for the system overall, and also infer how much a user may engage with it. Well designed visual aesthetics of a system can also foster collaboration between users, as used in Google Docs and online learning tools. Thus, the study of aesthetic design is critical in order to create collaborative systems that not only allow for a more friendly first-impression of the system but also encourage frequent user engagement. [36]. Games often add extra visual effects to make it more visually aesthetically pleasing and appealing to the player. This can be done through

decorating the world itself or by adding effects to make the game experience more interactive and responsive called game feel [123]. Previous work has been done to automate this process, such as in the Juicer project [54]. Aesthetic design is subjective to a human experience - determining what looks good or feels good in a game is harder to describe or quantify than analyzing raw metrics of performance.

2.2.2 Designer Intent

For creative projects such as video games, the designer’s intent is expressed thoroughly through the different aspects of the game. This expression can take form in the game’s art, music, level design, narrative, and general gameplay—all of which contribute to the overall player experience and feel of the game. Automating game design while maintaining original game designer intent is a challenge, especially when given limited context and understanding on the AI’s part. Nelson et. al. first explored how the communication and discrepancies between AI game design assistants and human users affect the quality and experience of a game [88]. This was later expanded by Treanor et. al. to build entire games from user ‘ideas’ in a mixed-initiative process [130]. Various other works have looked to model and expand the creative intent of the designer and apply the output to generative game content [53, 69, 98] or games themselves [126] and then evaluate them on the player experience.

2.3 Artificial Intelligence

2.3.1 Finite-State Machines

Finite state machines (FSMs)—as defined by Georgios and Togelius [138] – are an abstract representation of an interconnected set of actions, states, and transitions represented by a graph. These graphs typically represent conditional relationships between nodes and actions. FSMs are one of the key components of classic game AI. They have been used to create simple non-playable character behaviors in a

large number of games, including Pac-man, Half-life, and F.E.A.R [93]. Previous works have also explored the use of FSMs as solver agents in AI competitions such as Starcraft [114] and Pommerman [142] and as agents in a mixed-initiative simulation system [92]. Finite state machines lack dynamicity compared to other graph-based artificially intelligent agents, such as behavior trees. They also lack in depth and complexity compared to behavior trees; which are more commonly used to create more realistic and varied NPC behaviors in games.

2.3.2 Multi-Agent Simulations

Simulation games such as the *Sims*, *Dwarf Fortress*, and even *Conway's Game of Life* depend on multiple autonomous agents interacting with each other in order to create unique experiences in the system. Sometimes these interactions lead to emergent behaviors – behavior unseen or unexpected by the human designers and players for these simulation games. As such, simulation design with multi-agent systems in game AI research can create open-ended challenges through emergent design and unique environments for training AI such as reinforcement learning agents. Earle et. al [40] introduce a training environment for reinforcement learning agents in the game SimCity and examine population behaviors of cellular automata in Conway's Game of Life. Green et. al [50] use a minimal clone of RollerCoaster Tycoon to generate a diversity of theme park layouts. Suarez et. al. take the mechanics of a large MMORPG to simulate and discover emergent agent behaviors and interactions in a confined simulation environment [118]. Simulation games provide a perfect testing ground for artificial agents to either engage with or generate artifacts that can be extrapolated to real-world scenarios.

2.3.3 Emergent Behavior

In the game AI research field, emergent AI has become an increasingly notable area of study. The breadth of interactivity afforded by non-playable characters and back-end artificially intelligent and adaptive systems allow for more individualized



Figure 2.2: A screenshot of Spelunky, a roguelike platformer where the sections of the level are randomly generated and populated with items and enemies using procedural content generation (PCG)

player experiences, some of which the game designers may have never intended. In games, many emergent phenomena result from NPCs interacting with each other based on some set of rules, such as in Dwarf Fortress, or from adapting to player behaviors and decisions, such as the AI director in Left 4 Dead. In games research, recent work has explored how artificial generators can appear “haunted” to a player when the generator creates highly abnormal samples [67] or when narratives emerge from level generations that stem from the individual player experience and interaction with the generated artifact [89]. The focus of this paper is to explore how emergent AI behavior from generated FSMs could inspire narratives or descriptions of entities that have emerged through abstract code, or how multi-agent systems could interact without any pre-made definition to create more complex situations not foreseen by the designer.

2.4 Procedural Content Generation

Procedural content generation (PCG) is defined as the process of using a computer program to create content with limited or indirect user input [108]. Such

methods can make a quicker and more efficient automated content creation process, and also enable aesthetics based on generation. Using PCG systems and other artificially generated tools can speed up the development process for designers as they make their games.

PCG has been used in games from the 1980’s Rogue to its descendent genre of the *roguelikes* used in games such as Nethack (Nethack Development Team, 1980,) Spelunky (see Figure 2.2) (Mosmouth, LLC, 2008,) and Hades (Supergiant Games, 2020), as well as games that revolve around terrain and world generation such as Minecraft (Mojang, 2011) and No Man’s Sky (Hello Games, 2016). It has been used to build levels such as The Binding of Isaac (Edmund McMillen, 2011), facilitate enemy encounters such as in the Pokemon series (GameFreak, 1996), and for item or weapon generation such as in Borderlands (Gearbox Software, 2009). In academia, PCG has been explored in many different game facets for generating assets [45, 103], mechanics [15, 60], levels [116], boss fights [112], narratives [3], tutorials [18, 58], or even other generators [42, 56].

A plethora of AI methods underpin successful PCG approaches, including evolutionary search [127], supervised and unsupervised learning [75, 121], and reinforcement learning [57]. The results of these implementations have led to PCG processes being able to generate higher quality, more generalizable, and more diverse content. The generation process for PCG levels typically focuses on generating towards an objective or fitness that can be calculated based on some metric (i.e. playability, solution length, entropy.) While they may be successful in their endeavor of producing levels that guarantee completion of this objective, most PCG systems do so without taking into account the look or visual appeal of the level. PCG systems may also stray away from original designer intent due to lack of direct control or human feedback with the generation process.

2.4.1 Generator Models

Deep neural networks can be used to produce game content ranging from levels [6], sprites [104], animations [76], and 3-dimensional (3D) assets [137]. Such examples of these models include Generative Adversarial Networks (GANs) [46] which use two nearly symmetric deep neural networks – a generator network and a discriminator network – that are adversarially trained to produce content that resembles the training data but is novel. Likewise, variational autoencoders (VAEs) [65] are made up of a symmetrical network that trains output to be represented by a probabilistic latent vector. By modifying the latent vector that is utilized by the generator model in a GAN architecture or the probabilistic latent vector in the middle of the variational autoencoder network, one can control the content output.

Text-to-Image generation employs natural language image descriptions as a means to condition generative image models. This class of models aims to generate high quality images that semantically align with the provided input prompt. With the rise of conditional diffusion [110] and Stable Diffusion models [101], text-to-image generation has seen a huge increase in popularity.

2.4.2 Large-language Models

The development and growing accessibility of large language models (LLMs) for the use of game AI research has become increasingly relevant. LLMs have been utilized as game playing agents to perceive and interact with simulation environments [78] and complex domains such as Minecraft [135] and Nethack [66]. For content generation, LLMs have been used to generate levels for previously researched PCG domains such as Sokoban [125] and Mario [119]. LLMs have also been used for narrative generation to facilitate story telling [122] and quest generation [131]. For data augmentation, these models can be applied to text-to-asset input prompts to train other generative models such as for sprites and levels [83].



Figure 2.3: Example generation from Kate Compton’s “So you want to build a generator...” Tumblr post coining the “10,000 Bowls of Oatmeal” problem. While each flower generation is computationally “unique”, to a user, they look very similar - hence leading to the 10000 bowls of oatmeal problem.

2.4.3 Evaluating Generated Content

While both pure procedural content generation and mixed-initiative content generation systems can lead to an expanded area of creativity and artifacts overall, measuring what is “good” content is a difficult task in itself. Without some standard of measurement, generated content can fall into the predicament of what Compton defines as the “10000 bowls of oatmeal” situation – where a generator can create content that lacks in noticeable differences from other content while still being mathematically “unique”. These minimal changes make the content seem repetitive or completely unnoticeable to an end user, and therefore cause the generated content to lose implicit meaning or value [27]. Figure 2.3 shows an example of sameness across generated content from the Compton’s ‘Flowers’ generator¹. Related to the evaluating aesthetic content subsection (see 2.2.1), this metric is hard to quantify and mainly relies on human evaluation and interaction.

¹<https://www.galaxykate.com/apps/Prototypes/LTrees/>

2.5 Evolutionary Algorithms

Evolutionary Algorithms, centered around the idea of “survival of the fittest”, iteratively mutate feature vectors (genomes) to create a population, score the population based on an objective criterion, or fitness, and select the best genomes for the next generation. These algorithms have been used to generate game AI components such as dungeons [8] and enemies [94]. The (1+1) evolution strategy, also known as the “Hillclimber” algorithm [91], is a greedy evolutionary algorithm that incrementally searches for the local optima to find the best genomes among a population – typically comparing the best current sample fitness score against the fitness score of a mutated version of the sample (1+1). The algorithm continuously chooses the genome with the higher fitness score between the parent and the child of a given population to serve as the parent of the next population. $\mu + \lambda$ is a variation of the hillclimber, where a population of samples is maintained and a certain number of the best, μ , are retained each generation, while a number of the lower fitness samples, λ , are replaced with mutated versions of the better fitness samples.

2.5.1 Interactive Evolution

Contrary to pure PCG systems, interactive evolution (IE) is a content generation method where the evaluation of the system’s content output is dependent on the user’s selection [37]. Picbreeder (see Figure 2.4) is such an example of a system where users are able to select from a set of different images generated by an evolutionary algorithm. Images are evolved and removed based on a subset of images selected by a single user. This guides the system towards evolving the population of images to either a singular goal image or to fit the user’s personal preference [106]. Liapis et al. also looked to use machine learning and evolutionary techniques with constrained, limited user selection to generate content based on visual and aesthetic appeal of the designer in various projects ranging from

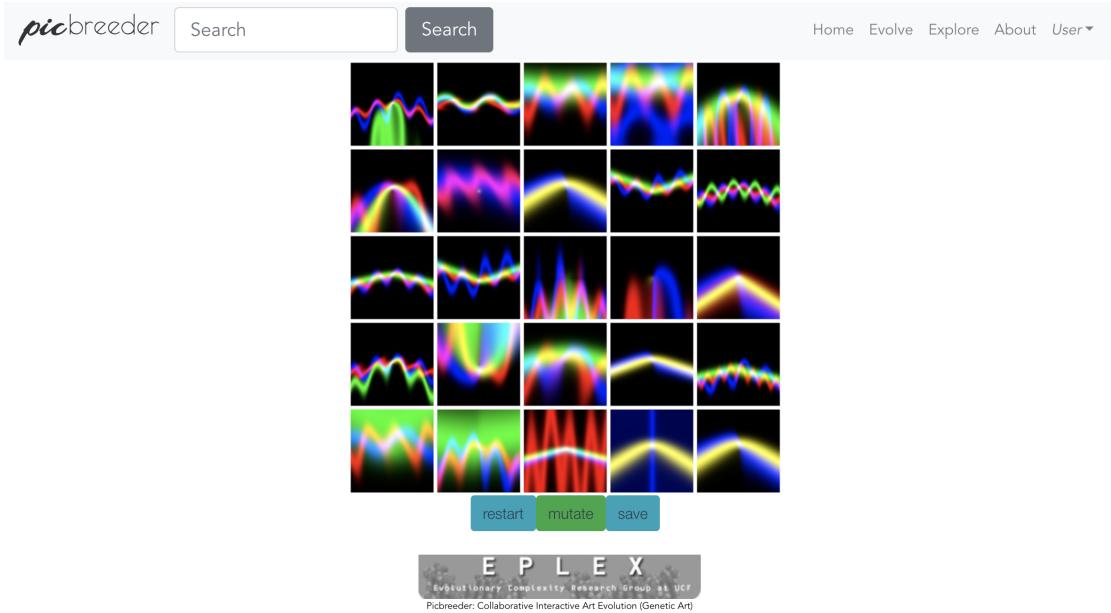


Figure 2.4: Screenshot of the Picbreeder online interactive evolver tool. Users can select and evolve AI generated images with an open-ended objective.

spaceship designs to map terrains [68, 70, 72, 73, 74].

2.5.2 Latent Variable Evolution

Deep Interactive Evolution using Latent Variable Evolution (LVE) involves mutating and evolving a population of vectors that is passed through a deep neural network to create new samples. Combined with IE, Latent Variable Evolution allows controllable evolution of generated samples. Bontrager et al. introduces and demonstrates its effectiveness in generating images [13]. Schrum et al. experiment with using Interactive Latent Variable Evolution (ILVE) to evolve generated levels for tile based 2D video games, The Legend of Zelda and Super Mario Bros [105]. Thakkar et al. use LVE with an autoencoder generative model to generate levels for the game Loderunner [124]. Volz et al. use LVE with a Generative Adversarial Network to evolve levels for Super Mario Bros [134].

2.5.2.1 User Fatigue

However, IE systems can also exhaust a user and lead to participant fatigue - a phenomena where a content-focused system can cause the participant to become

bored, disinterested, or drained from repetitive evaluation of the generated content. This is especially hindering when these types of systems require so much feedback in order to effectively learn [55]. The more effort required from the user to evaluate (i.e. playing a level, filling out forms, asking for specific critiques) the less likely they are to continue providing feedback, causing the system to stagnate in its ability to generate or discern high quality content.

2.5.3 Quality Diversity Algorithms

Quality diversity (QD) algorithms search and explore spaces for solutions that are both varying in content and optimal based on a predefined fitness value. These algorithms often search based on evolutionary methods - sampling and mutating from previously seen content in order to improve in quality and find more solutions.

2.5.3.1 MAP-Elites

Multi-dimensional Archive of Phenotypic Elites (MAP-Elites), is a QD algorithm introduced by Mouret et al. [85] that searches a multi-dimensional search space for high-performing, diverse solutions. MAP-Elites maintains a map of the best recorded solution across a particular set of feature dimensions. This map is constructed by dividing the solution space into a group of cells based on a pre-defined behavior characteristics. Any new solution found will not only be evaluated for fitness but also for its defined characteristics then placed in the correct cell in the MAP-Elites map. If the cell is not empty, both solutions compete and only the fitter solution survives. Because of the map maintenance and the cell competition, MAP-Elites can guarantee a map of diverse and high quality solutions, after a finite number of iterations through the generated population.

MAP-Elites has shown remarkable success in generating content for games such as dungeons as shown by Alvarez et al. and Guerrero et al. [2, 51] utilizes MAP-Elites to generate a team of agents for automated gameplay. MAP-Elites has also been used to create, replicate and explore real-world adaptability by simulated

agents in virtual open-ended environment as studied by Norstein et al. [90]. Pierrot et al. evolve repertoires of full agents to combine any RL algorithm with MAP-Elites to dynamically learn the hyperparameters of the RL agent [100]. This approach not only alleviates the user’s workload but also enhances performance in the evaluated environments.

2.5.3.2 Constrained MAP-Elites

A variant of the MAP-Elites algorithm, known as Constrained MAP-Elites enhances the content diversity by evaluating samples on a conditional criteria. This algorithm, based on the Feasible Infeasible 2 Population method (FI2Pop) [64], determines first whether a sample is feasible or infeasible through a constraint check. This check is done separately from the fitness and behavior characteristic evaluation. Future generations can then be sampled from either the feasible or infeasible populations to create a more rigourously evaluated archive. Khalifa et al. [58] and Alvarez et al. [2] use the Constrained MAP-Elites methodology for level generation.

2.6 Mixed Initiative Systems

Mixed-initiative AI systems involve a co-creation of content between a human user and an artificially intelligent system during the generation process [139]. Previous mixed-initiative systems include selecting from and evolving a population of generated images [106], composing music [80, 128], and creating game levels through suggestive feedback [79]. Mixed-initiative and collaborative AI level editors for game systems have thoroughly been explored in the field as well through direct and indirect interaction with the AI back-end system [10, 16, 38, 52, 74, 107, 113, 143]. These recent developments look more into how the human users are affected through their relationship with collaborating with these AI systems and how it can be improved through examining the dimensionality of quality diversity algorithms,

the evolutionary process, or the human-system interaction itself [1].

2.6.1 Collaborative Game Design

Collaborative game design systems have been explored in the field as well, most notably Cook’s ANGELINA [32, 33] and Puck [29] systems. These systems look to design content for multiple facets of game design outside of just level or art-based asset design, such as generating the game mechanics [34], game feel or “juice” for the game levels [54], sounds and audio for the game soundtrack [17], or the entire game itself [30, 31]. Some works use mixed-initiative human interaction to collaboratively design novel games using rules and mechanics given by the user [84, 87, 120, 130].

Compton’s definition of casual creation tools and survey of existing systems also strongly fit for the objective of collaborative game design and mixed-initiative systems, where the end goal may not just involve creating a productive output, but facilitating a creative experience [28].

2.7 Crowd-sourced User Content

Online game content design and collaborative, community-made content generation exists across multiple genres of games – typically in the form of levels for games with existing mechanics such as in *Super Mario Maker* or *FreeRider Online* or also in the form of in-game assets such as the texture pattern sharing feature of the Animal Crossing series. With an online sharing system for a game, a community of players can form around designing unique content for a game that others can use. In AI research, user created content – generated with or without AI assistance – can help to train and design AI systems capable of generating unique content. This design feedback loop between AI model and user can encourage more user engagement with the system and guide the generative design process.

In the areas of game AI research and the independent game development, much

work has been done to develop game engines that facilitate online communities. User-bases for micro-game engines and fantasy console engines (i.e. Bitsy, PICO-8) allow users to share games made with these specific engines on platforms such as Itch.io or the website's BBS. The accessibility and simplicity of these game engines are what make them so popular and distinguishable in their design [136]. The ANGELINA system by Cook et. al. uses automated game design principles to design new games across multiple genres[32, 33]. An Itch page² around these generated games has been made to document the works output by the system.

²<https://gamesbyangelina.itch.io/>

Chapter 3

Automatically Generating Diverse Game Levels

This chapter acts as an introductory work for the content generation systems described in later chapters. The Mech-Elites system looked to find and generate tutorial levels based on the *critical mechanics* of a variety of games. These critical mechanics are mechanics that are necessary to successfully play and complete any level in the specified game. The system aimed to generate an archive of simple, minimalist levels – each level requiring the player to activate the critical mechanics either in isolation or combination. This system generated critical mechanic tutorial levels for the following GVG-AI framework games: Zelda, Solarfox, Plants, and RealPortals. To create the archive of levels, the quality-diversity algorithm MAP-Elites [85] and the genetic algorithm Feasible Infeasible 2-Population (FI2Pop) [64] was combined together into the hybrid genetic algorithm known as Constrained MAP-Elites (CME) [61] to generate diverse, minimalist tutorial levels for each game.

3.1 General Video Game Artificial Intelligence (GVG-AI)

The GVG-AI framework is a platform for automatic general video game research [96]. The framework provides multiple tracks including game playing [97], level generation [62], learning [129], rule generation [59], and two-player game-play [44].

Every game in the GVG-AI Framework is described using the Video Game Description Language (VGDL) [43]. A VGDL game consists of two file types: the game description file and one or more level files. Four parts make up the game description file: a *Sprite Set*, which determines which game objects exist and how they look and behave; an *Interaction Set*, which describes how sprites interact; a *Termination Set*, to dictate how the game ends; and finally a *level mapping* between game sprites and their ASCII representation in the level files.

VGDL is encompassing enough to describe a wide variety of simple 2D games, yet remains easy to read for humans. Some of them are adaptations of known games, such as *Pacman* (Namco, 1980), *Plants vs Zombies* (PopCap Games 2009), and *Galaga* (Namco 1981), others are demakes of big games, such as *The Legend of Zelda* (Nintendo, 1986) and *Pokemon* (Game Freak, 1996), while others are brand new games, such as *Wait for Breakfast* where the player must remain idle in order to win; a game that is difficult for artificial agents to solve.

The four GVG-AI games used in this work are Zelda, Solarfox, Plants, and RealPortals. Each was selected based on a previous work [11], which categorized GVG-AI games based on how they were played. These four games contain a diverse array of mechanics, terminal conditions [time-based (Plants), lock-and-key (Zelda and RealPortals), and collection (SolarFox).] Thus, we selected these as a representative set of the GVG-AI framework’s games.

- **Zelda:** is a GVG-AI adaptation of the dungeon system in *The Legend of Zelda* (Nintendo 1986). The player must pick up a key and unlock a door in

order to beat a level. Monsters populate the level and can kill the player, causing them to lose. The player can attack monsters with a sword and gain points.

- **Solarfox:** is a GVG-AI adaptation of *Solar Fox* (Bally/Midway Mfg. Co 1981). The player must dodge both enemies and their flaming projectiles in order to collect “blibs” in the level. The player gains a point for each blib collected, and victory is granted after collecting all blibs in the level. Several levels contain “powered blibs,” which are worth no points. If a player collides with a powered blib, it will spawn a “blib generator,” which can spawn more blibs.
- **Plants:** is a GVG-AI adaptation of *Plants vs. Zombies* (PopCap Games 2009), a tower defense-style game. If the player survives for 1000 game timesteps, they win. Zombies spawn on the right side of the screen and move left – the player loses if a zombie reaches the left side. Plants, which the player must grow in specific “marsh” tiles, can destroy zombies by automatically firing zombie-killing peas. Each zombie killed is worth a point. Occasionally, zombies will throw axes, which destroy plants, therefore, the player must regrow plants to maintain protection.
- **RealPortals:** is GVG-AI 2D adaptation of *Portal* (Valve 2007). The player must reach the goal, which can be behind a locked door that needs a key. Movement is restricted by water, which kills the player if they touch it. To succeed, players need to be creative in overcoming this hazard by using portals which can teleport them across the map. Players can pick up wands which allow them to toggle between the ability to create *portal entrances* and *portal exits*. Players can also push boulders into the water to transform the water into solid ground, creating land-bridges on which they can walk.

3.2 Mech-Elites System

The Mech-Elites system uses a multi-step evolutionary process in order to generate novel, minimalist, and playable levels that demonstrate each critical mechanic combination. First, a population of randomly generated levels is made using GVG-AI’s built-in random level generator class. These levels are represented as a 2D array of tiles – each tile corresponding to a game sprite from the game’s VGDL description file. The levels are mutated by randomly changing tiles. The mutated levels are evaluated for playability, mechanics performed, and evolutionary fitness, using the AdrianCTX competition agent [97] – an MCTS-based agent and winner of the planning track in the 2016 GVG-AI competition. These levels are placed into feasible and infeasible MAP-Elites matrix archive based on these evaluated metrics. Levels evaluated with the highest fitness and that pass the constraint check for a cell are considered “elite” for the matrix. 20 “infeasible” levels are stored in the matrix for a given cell, while 1 “feasible” level is stored in the same cell. The new population for the next generation in the evolutionary process samples members from both the infeasible and feasible populations with a 50% chance to sample from the “feasible” population. Levels from the random generator are also added to the population to prevent a local optimum from being reached during generation.

3.2.1 Constraints

This system uses a constraint check to determine whether a level goes into the feasible (passes the constraint) or infeasible (fails the constraint or has a weaker fitness value than the feasible level) archives of the MAP-Elites matrix. Two components determine whether a level passes the constraint check for this system: *playability*, whether a level is able to be completed successfully by the player within a certain threshold of timesteps and without them dying; and *accessibility*, a measurement that ensures the player does not die within the first few timesteps

of a game – this was evaluated by using a DoNothing agent that remained idle in the level.

Equation 3.1 details the *playability* component of the constraint calculation

$$P = \frac{win}{|T_{win} - T_{ideal}| + 1} + \frac{(1 - win) * 0.25}{|T_{survival} - T_{ideal}| + 1} \quad (3.1)$$

Here, win is 1 if the agent finished the level successfully and a 0 otherwise. T_{ideal} represents the ideal time to finish the level (manually set by the experimenters), and T_{win} and $T_{survival}$ represent the finishing time of the agent before successful completion and unsuccessful completion respectively. The constraint threshold is chosen based on the intention that a level will both be winnable and pass the idle agent tests, but may be within a certain range for T_{ideal} . Equation 3.2 describes the *accessibility* component of the constraint

$$A = \begin{cases} 1 & \text{if } \frac{N_{pass}}{N_{total}} \geq 0.5 \\ \frac{N_{pass}}{N_{total}} & \text{otherwise} \end{cases} \quad (3.2)$$

Here, N_{pass} represents the number of times the idle agent survived until the ideal time, and N_{total} represents the number of idle agent tests. These two components are added together to formal the final constraint value.

This constraint is implemented to allow the evolutionary algorithm to still select from infeasible levels in the hopes that they could eventually evolve into feasible levels or express different critical mechanic combinations via mutation. Elite / feasible levels can be replaced by newer levels that have higher fitnesses, and are thus moved to the “infeasible” population.

3.2.2 Behavior Characteristics

The mechanics performed by the AdrianCTX agent correspond to the behavior characteristic (BC) dimensions of the MAP-Elites matrix. These mechanic determine where along the axis in the MAP-Elites matrix an evaluated level is stored.

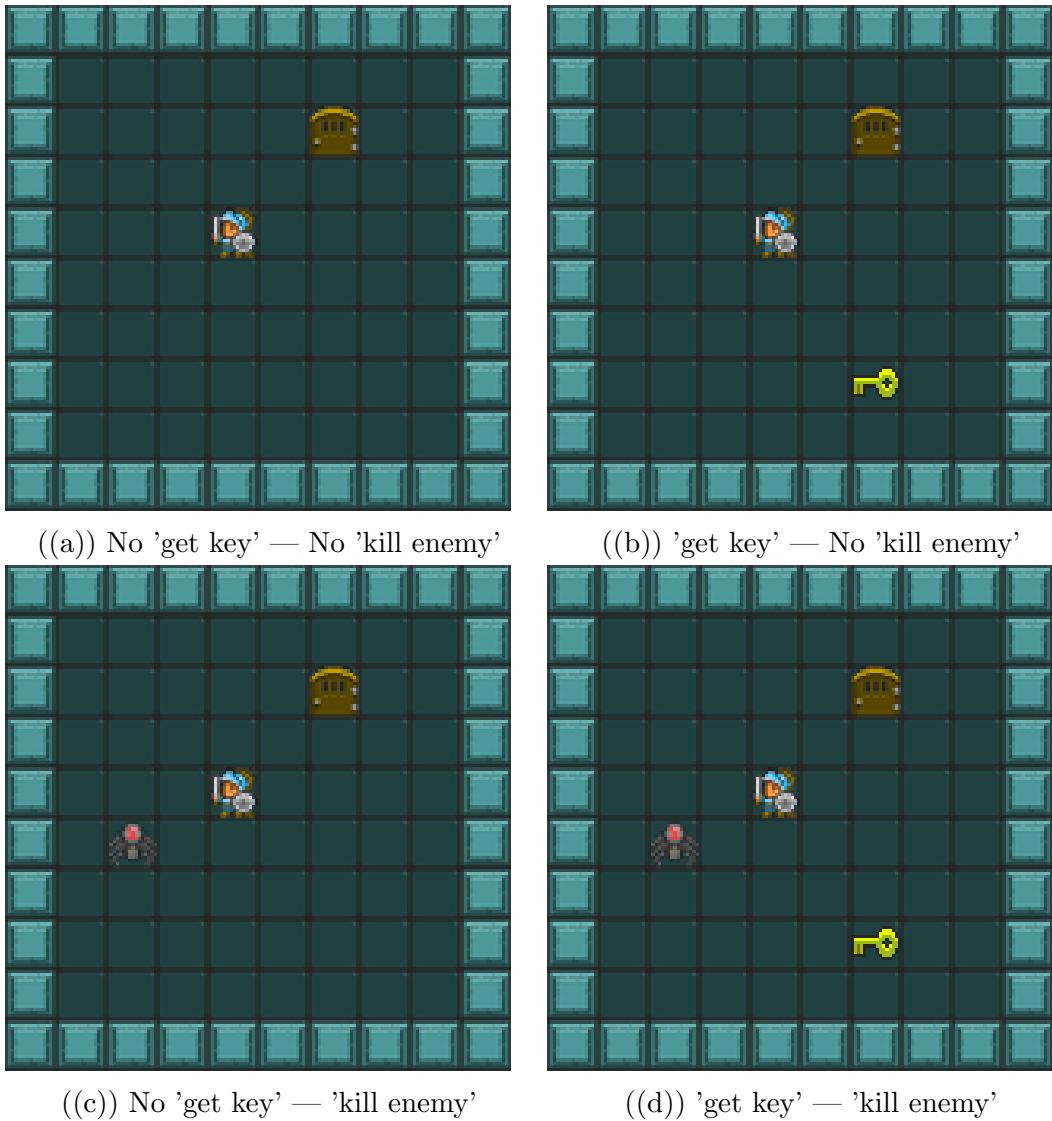


Figure 3.1: An example of an optimum MAP-Elites matrix for the Zelda game mechanics "get key" and "kill enemy".

The activated mechanics – specified by the game's critical mechanic list – are stored as binary representations: 1 if a mechanic was performed during evaluation and 0 if not. For example, if the game mechanics for Zelda consisted of the list "get key" and "kill enemy", the CME behavioral characteristic will be 2 binary dimensions which will create 4 cells ("get key" & "kill enemy", No "get key" & "kill enemy", "get key" & "kill enemy", and "get key" & "kill enemy"). Figure 3.1 shows examples of an ideal matrix for the game Zelda based on these mechanics. With this, the number of possible cells that can be generated for each game is 2^n with n being the number of game mechanics measured for a game.

3.2.3 Fitness

To encourage the generation of minimalist levels, the fitness of a level was set by calculating its tile entropy (distribution of tiles) and derivative-tile entropy (relative placement of tiles) based on Khalifa et al's work [58]. This was to create simple but expressive tutorial levels without noise and extraneous elements. The fitness value determined whether or not the evaluated level would replace a level in the cell of the MAP-Elites matrix with the same behavior characteristics.

Equation 4.1 was used for the level fitnesses:

$$\text{fitness} = H(\text{lvl}) * w + H(\Delta\text{lvl}) * (1 - w) \quad (3.3)$$

where $H(\text{lvl})$ represents the raw tile entropy of a level, $H(\Delta\text{lvl})$ represents the entropy of the derivative of the same level, and w represents the pre-set weight value. This equation was based on the entropy tile fitness equation used by Khalifa et. al [58], however, the level derivative is calculated differently. Since the level was not separated and mutated based on vertical slices, the derivative was calculated based on the vertical and horizontal changes instead of only horizontal changes. For each tile, we count the number of different neighboring tiles on the north, south, east, and west of it and use the total count as the derivative value.

3.3 Experiment Setup

For the four GVG-AI games tested in this system (Zelda, Solarfox, Plants, RealPortals) we extracted their critical mechanics using the AtDelfi system [18]. Tables 3.1, 3.2, 3.3, and 3.4 list the critical mechanics measured for each GVG-AI game used in the experiment. A population size of 50 levels is made for each iteration of the evolutionary process – 20% of which are randomly generated levels and 80% are sampled from the MAP-Elites matrix populations. The process is run for 500 iterations for each game.

The *idle agent* for the constraint calculation (see subsection 3.2.1) is run a total of 5 times on the level, of which it must survive 3 in order to pass the constraint test. Both the idle agent and the AdrianCTX evaluator agent’s “ideal times” (T_{ideal}) were set to 70 timesteps for all experiments.

*	Dimension	Description
z1	space-nokey	Agent pressed the space bar when the avatar did not have a key
z2	space-withkey	Agent pressed the space bar when the avatar had a key
z3	stepback	A sprite ran into another sprite
z4	kill-nokey	A sprite killed the avatar when the avatar did not have a key
z5	kill-withkey	A sprite killed the avatar when the avatar had a key
z6	sword-kill	The agent killed an enemy sprite with a sword
z7	getkey	The agent picked up a key
z8	touchgoal	The agent touched a goal with the key and won the game

Table 3.1: Constrained MAP-Elites dimensions for the GVG-AI game Zelda

*	Dimension	Description
s1	hit-wall	Agent hit a wall
s2	hit-enemyground	Agent touched enemy ground
s3	hit-avatar	An enemy sprite hit the Agent
s4	touch-powerblib	The agent touched a powerblib
s5	spawn-more	A turning powerblib created a blib
s6	change-blib	A turning powerblib changed into a normal blib
s7	overlap-blib	A powerblib overlapped with a blib
s8	get-blib	The agent got a blib
s9	reverse-direction	A sprite hit a wall and reversed direction
s10	enemy-shoot	An enemy fired a missile at the avatar

Table 3.2: Constrained MAP-Elites dimensions for the GVG-AI game Solarfox

3.4 Results

Every mechanic in each game, except 3 from the RealPortals domain, was successfully expressed by a generated archive level. Various combinations of

*	Dimension	Description
p1	space	Agent pressed the SpaceBar
p2	hit-wall	A sprite touched a wall
p3	kill-plant	A zombie destroyed a plant
p4	zombie-goal	A zombie sprite reaches the goal
p5	pea-hit	A pea sprite hits a zombie sprite
p6	tomb-block	A tomb sprite blocks a pea sprite
p7	make-plant	Agent placed a plant on a marsh tile

Table 3.3: Constrained MAP-Elites dimensions for the GVG-AI game Plants

*	Dimension	Description
r1	space	Agent pressed the SpaceBar
r2	change-key-blue	changes blue avatar's current resource to a key
r3	hit-wall	A sprite touched a wall
r4	drown	destroy any sprite that falls in the water
r5	toggle-blue	avatar changes current portal shot to blue
r6	no-lock	any sprite tried to move through a lock
r7	no-portalexit	any sprite tried to move through an exit portal
r8	teleport-exit	orange avatar steps through the entrance portal
r9	no-moving-boulder	sprite tried to move through a moving boulder
r10	no-idle-boulder	sprite tried to move through an idle boulder
r11	change-key-orange	changes orange avatar's current resource to a key
r12	toggle-orange	avatar changes current portal shot to orange
r13	teleport-entrance	blue avatar steps through exit portal
r14	get-weapon	avatar picks up a weapon
r15	get-key	avatar picks up a key
r16	back-to-wall	portal turns back into a wall
r17	fill-water	moving boulder falls into water to fill it
r18	open-lock	avatar unlocks a lock
r19	touchgoal	The agent touched a goal and won the game
r20	make-portal	wall turns into a portal
r21	portal-missile-velocity	send a missile through a portal at the same velocity
r22	cover-goal	goal is covered by a missile
r23	blue-missile-in	send a blue missile thru a portal entrance
r24	orange-missile-in	send an orange missile thru a portal entrance
r25	portal-boulder	send a boulder through a portal at the same velocity
r26	stop-boulder-key	moving boulder stops after hitting a key
r27	stop-boulder-wall	moving boulder stops after hitting a wall
r28	stop-boulder-blue-toggle	moving boulder stops after hitting a blue portal toggle
r29	stop-boulder-orange-toggle	moving boulder stops after hitting an orange portal toggle
r30	stop-boulder-lock	moving boulder stops after hitting a lock
r31	teleport-boulder	sends a boulder to the other portal
r32	stop-boulder-boulder	moving boulder stops after hitting another boulder
r33	stop-boulder-avatar-blue	moving boulder stops after hitting the blue avatar
r34	stop-boulder-avatar-orange	moving boulder stops after hitting the orange avatar
r35	roll-boulder	boulder moved over a tile

Table 3.4: Constrained MAP-Elites dimensions for the GVG-AI game RealPortals

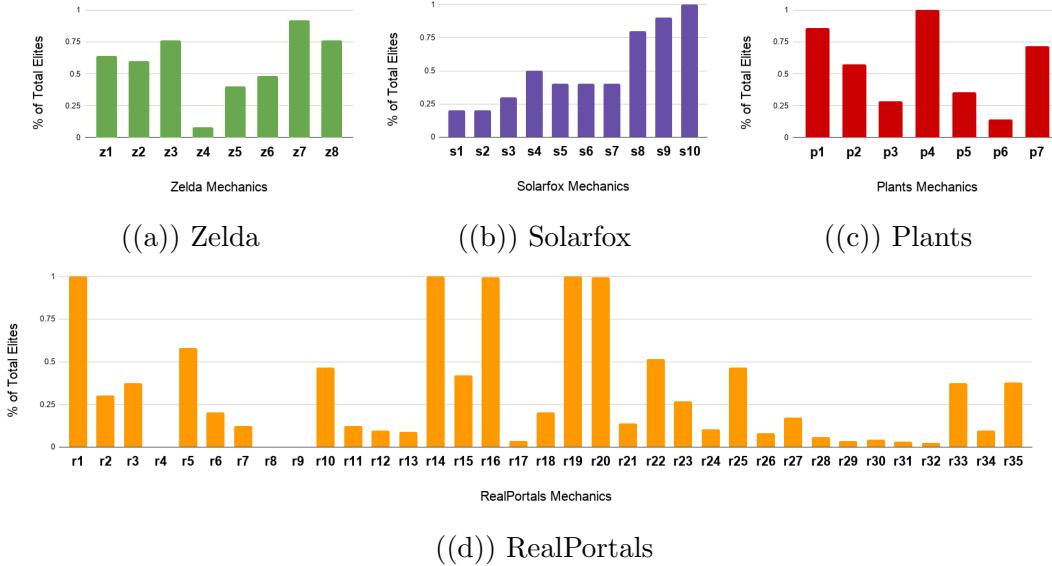


Figure 3.2: The percentage of elites that contain a specific mechanic for each game from the Mech-Elites system.

mechanics, some dependent on each other or mutually exclusive, were also present as elite levels. Figure 3.2 shows the distribution of mechanics for each game.

For Zelda, 55 out of the possible 256 cells of the MAP-Elites matrix were filled with 25 of these cells containing an elite feasible level. Solarfox had 52 out of the possible 1024 cells populated and 10 cells containing elite levels. The Plants game MAP-Elites matrix resulted in 31 out of the 128 cells filled and 14 elites. Finally, the RealPortals game, with over 34 billion possible cells for mechanic combinations had 6966 cells populated and 231 elite levels. The normalized elite counts across generations for our experiments is displayed in Figure 3.3.

Due to the constraints and fitness metrics, the elite levels from each game were guaranteed to be playable with the minimalist design while still expressing a range of critical mechanics – much like a tutorial from a game. While the levels expressed diversity in mechanics, many of the levels in the archive were uniform in design and lacking elements relative to their original human made levels (i.e. no water tiles in RealPortals and no marsh tiles in Plants.) These levels were sparsely populated most likely due to the minimalist objective from the fitness. Some elements allowed for the *possibility* of mechanics to be activated but were not *guaranteed* to be activated by the evaluating AdrianCTX agent. For example, having an enemy on

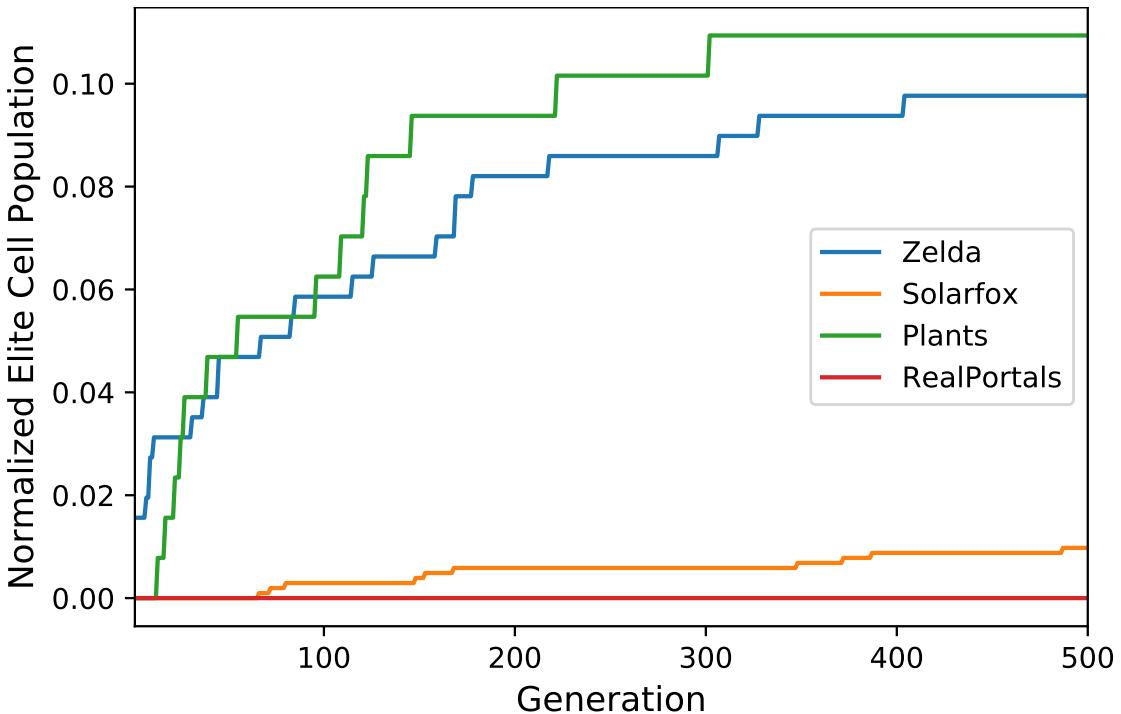


Figure 3.3: Normalized number of elite cells across generations for each evaluated game.

the map of the Zelda game levels sometimes did not guarantee the agent triggering the “kill enemy” critical mechanic. Because of this, it’s possible that a single level without any mutation could be placed into multiple cells but is limited to the agent’s deterministic action on the level. However, the collection of generated levels showcased important mechanics of the game in a way that was ideal for tutorial-like levels – the main intent for the development of the project.

3.5 Summary

This system aimed to generate minimalist levels across a variety of game domains in the GVG-AI framework that expressed each game’s critical mechanics in both isolation and combination. While substantial portions of the possible mechanic combinations were found using the quality-diversity algorithm MAP-Elites, there was a significant percentage of missing levels for each game’s archive. The design of these generated levels also lacked in diversity and aesthetic quality although their mechanic expression space maintained diversity. To remedy these

issues, human-feedback and mixed-initiative methodologies can be applied during the generative process to semi-automate the search and generation of new levels.

3.6 Reflection

The evolutionary algorithm used in the Mech-Elites system was able to show how content could be optimized for a domain over time if given clear constraints and objectives. But would the system have been able to find the missing mechanics if given a longer generation time? There were some mechanics in the games that also depended on each other to be activated. For example, the “kill-withkey” mechanic in Zelda could not be activated unless “getkey” also was. Therefore it was possible that some levels with specific mechanic combinations could not be made.

What about the aesthetic of the levels? The objective function was chosen to try to create “minimalist” designed levels by reducing the number of extraneous or differing neighboring tiles in the level. But was this objective function too limiting and potentially the cause of why some mechanic combinations weren’t able to be found? Were these levels also truly minimalist? Is it possible for a content generation system to have decorative elements that are structured and make the content more aesthetically pleasing but serve no mechanical or gameplay purpose?

The performance of the agent could have also affected the defining characteristic of the level. While the level was tested multiple times by the same agent, one level could have differing activated mechanics; therefore the same level could belong to multiple cells but have no change in the content. If the experiment had tested the generated levels on a combination of different agents with different gameplaying behaviors, maybe it would cover more of the mechanic space in the archive with fewer generations.

This system was the start of my research direction into human-AI collaboration. The evolutionary system in this study was capable of making diverse content across

a variety of games. However, with human assistance and evaluation, it could make content that was higher in quality and more mechanically diverse – filling in the missing level definitions of the archive.

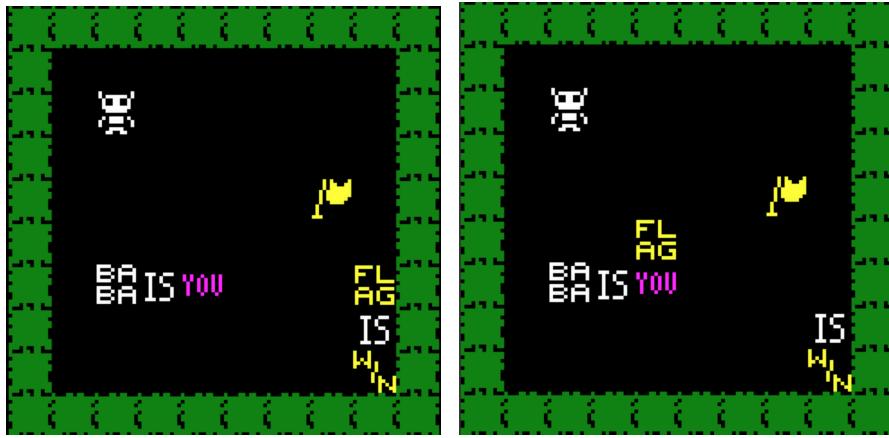
3.7 Takeaways

- Content generation systems alone may not be able to cover an entire space of content possibilities
- Evolutionary systems allow for explainability in content generation
- Too constraining of an objective/fitness function can limit the output content possibility space
- Multi-step evaluations allow for higher quality content generation, but can limit or slow the search for diversity in the content

Chapter 4

Mixed-Initiative Mechanic Diversity

The results of the Mech-Elites system highlighted the need for more exploration in a MAP-Elites matrix. This exploration would help search for the missing elite cells of a diverse sample space. This chapter describes the Baba is Y'all system – a mixed-initiative system that looks to solve this issue by encouraging users to collaborate with an evolutionary generator and design levels for the puzzle game Baba is You. This mixed-initiative collaboration uses an online crowd-sourced level editor website to design, evaluate, and share levels. Like the Mech-Elites system, a matrix of levels is maintained using the MAP-Elites algorithm. Users are suggested by the system to fill in missing cells through the website's interface and edit page. The system's game domain was based on the Baba is You game - a Sokoban-like puzzle game where players push word blocks in a grid-world environment to manipulate the object and sprite interactions and properties of the level itself. This system was intended to facilitate a collaborative interaction between an online community of human designers and the back-end AI system of the website. Through constant interaction with the site, both users and the system could collaboratively fill in the missing diversity samples in the level matrix while improving the aesthetic quality of the levels over time. Two systems are described



((a)) ‘Flag is Win’ is satisfied at the ((b)) ‘Flag is Win’ is not satisfied
start at the start

Figure 4.1: Simple ‘Baba is You’ levels with different satisfied starting rules.

in this chapter: Baba is Y’all version 1 (BiY v1) released in April 2020 [22], and Baba is Y’all version 2 (BiY v2) released in May 2021 [20].

4.1 Baba is You

Baba is You (Arvi “Hempuli” Teikari, 2019) is a puzzle game where players can manipulate the rules of a level and properties of the game objects through Sokoban-like movements of pushing word blocks. Since the mechanics act as both constraints and solutions for the player, the game provides an interesting study for procedural level generation where game levels and game rules are intertwined.

Every “Baba is You” level must have a win condition and a controllable character that can reach the defined win condition in order to successfully solve the level. These two rules are defined by “X-IS-WIN” and “Y-IS-YOU” respectively - where X and Y are any type of game objects that can exist in the game level. Figure 4.1 shows two simple levels from “Baba is You”. Figure 4.1(a) shows a simple level where the player controls the “Baba” object (with the “Baba-IS-YOU” rule is currently active) and tries to reach the “Flag” object to win the level (with the “Flag-IS-WIN” rule is active at the start of the level). The rules do not have to be initially made at the start of the game and can be manipulated at any point while

the player is attempting to solve the level. Figure 4.1(b) shows the same level as figure 4.1(a) but the rule “Flag-IS-WIN” is not satisfied at the beginning. The player will need to push the word “Flag” to complete the sentence “Flag-IS-WIN” - thereby activating that rule so they can win the level.

Rules for the level are defined by statements readable as English text and read from up-down and left-right. So “X-IS-YOU” would be interpreted as a valid rule, but not “YOU-IS-X.” As such, all rules must take one of these 3 formats:

- **X-IS-(KEYWORD)** – where KEYWORD belongs to a word in the keyword word class that manipulates the property of the game object class X (i.e. “WIN”, “YOU”, “MOVE”, etc.)
- **X-IS-X** – a reflexive rule stating that the game object class “X” cannot be changed to another game object class.
- **X-IS-Y** – a transformative rule changing all game objects of class X into game objects of class Y.

The objects in the game do not hold traditional semantic meanings or game mechanic properties. For example, a player can control Baba, Keke, walls, or any other object or multiple objects at once in the game so long as it can be represented by a physical object(s) and the rule “X-IS-YOU” is connected, where X represents the physical object that the player is currently controlling. This applies to all rules created initially before the player starts to solve the level and any rules that are created or removed while the player is solving the level. Game object classes that are affected by any rule have their properties updated accordingly. For example, creating the rule “Keke-IS-MOVE” within the game would make all Keke objects start autonomously moving within the game after each action is taken thereafter. If the rule is removed after a future action is taken, all Keke objects would stop moving subsequently.

Table 4.1: Baba is Y'all Ruleset

Rule Type	Definition
X-IS-X	objects of class X cannot be changed to another class
X-IS-Y	objects of class X will transform to class Y
X-IS-PUSH	X can be pushed
X-IS-MOVE	X will autonomously move
X-IS-STOP	X will prevent the player from passing through it
X-IS-KILL	X will kill the player on contact
X-IS-SINK	X will destroy any object on contact
X-IS-[PAIR]	both rules 'X-IS-HOT' and 'X-IS-MELT' are present
X,Y-IS-YOU	two distinct objects classes are controlled by the player

4.2 System Description

The jam version of the *Baba is You* game¹ was recreated in Javascript to be implemented in the level editor website Baba is Y'all. This version uses a smaller subset of rules than the fully released version of the game². Table 4.1 shows the list of 9 rules that can be made in the Baba is Y'all engine and the rules used as behavior characteristics for the MAP-Elites matrix that sorts the levels in the database. We wanted to have a substantial range of possible rules that were intuitive to understand, but also limit the scope of the domain's mechanic space (see Chapter 3's GVG-AI critical mechanic tables.) Baba is Y'all is a mixed-initiative tool that allows users to edit, save, and play community uploaded Baba is You levels similar to that of *Super Mario Maker*. The website is made up of multiple modules – some including AI recommendation and generator systems – to facilitate mixed-initiative level design and encourage the creation of levels that have mechanics that have not been previously combined.

The following subsections detail the modules in the Baba is Y'all system. These descriptions are based on the most recent version of the Baba is Y'all website from the journal publication [20]. Many of these modules were retained from the first proof-of-concept prototype system [22], however additional features were added to create a seamless and more engaging user experience. The website is split into

¹<https://hempuli.itch.io/baba-is-you>

²https://store.steampowered.com/app/736260/Baba_Is_You/

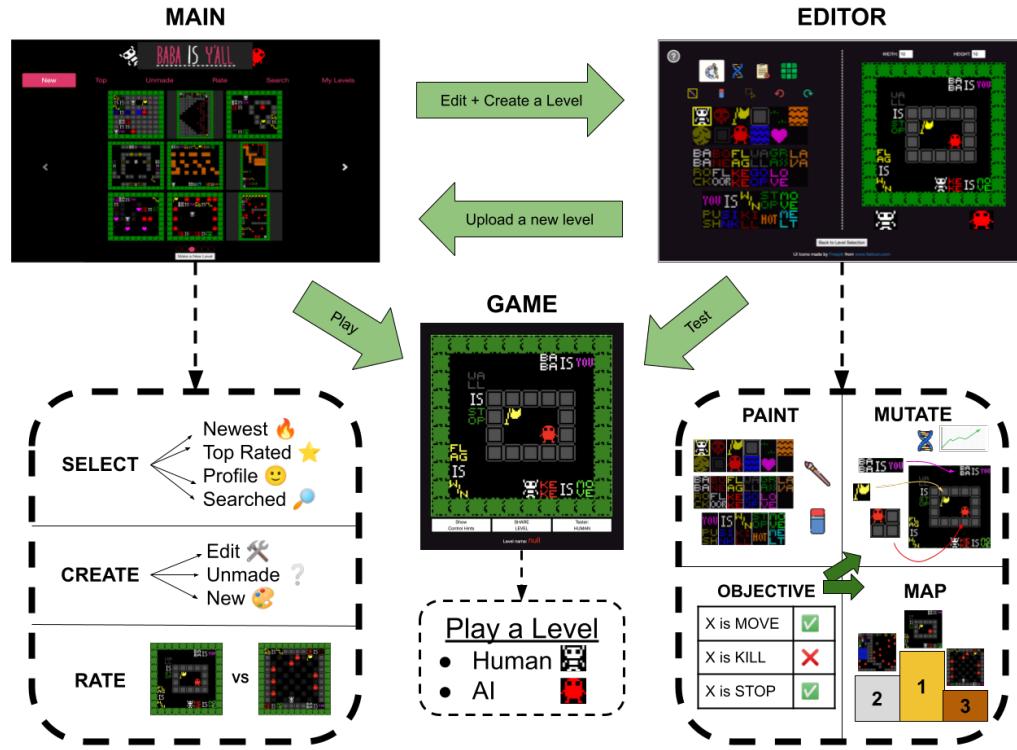


Figure 4.2: System flowchart of the Baba is Y'all website system

three screens: the level editor screen containing the editor, mutator, objective, and map modules; and the game screen containing the game module; and the home screen containing the map module, rating module, user profile page, and search function – where users can search for levels by author, level name, or map.

4.2.1 Editor Module

The editor module allows users to directly edit Baba is You levels. Much like a paint editor, users can select from a palette of sprites and words and place them directly on the map. There are 11 object class sprites – entities associated with words and sprite objects (e.g. Baba, Keke, Bone, Goop) and 21 keyword class objects (e.g. “YOU”, “IS”, “SINK”, “BONE” (literal word)) taken from the original jam game. As a basis, users can select from a blank map, a basic map that contains the rules “X-IS-YOU” and “Y-IS-YOU” placed with *X* and *Y* randomly chosen sprite objects and their associated keywords, a noisy, randomly generated map, or an elite map chosen from the Map Module (see 4.2.4). A level must be



Figure 4.3: Editor Module

tested and solved in the Game Module (see 4.2.5) before it can be submitted to the website database. In BiY v2 [20], users can seamlessly switch from the editor module to the mutator, objective, and map modules to easily switch between human edits and AI edits for a better mixed-initiative collaborative interaction.

Figure 4.3 shows a screenshot of the editor screen on the website. Users can select from the available sprite tiles on the left panel and edit the map on the left with the paintbrush cursor. Other editing tools such as an erase and select tool and undo and redo buttons are featured on this screen.

4.2.2 Mutator Module

The mutator module (referred to as the generator module in BiY v1) is an evolutionary content generator system that uses the tile-pattern Kullback-Liebler Divergence (ETPKLDiv) algorithm [77]. The ETPKLDiv algorithm uses a 1+1 evolution strategy, also known as a hillclimber, to improve the similarity between the current evolved levels and a reference level. The algorithm uses a sliding window of a fixed size to calculate the probability of each tile configuration (called tile patterns) in both the reference level and the evolved level and tries to minimize

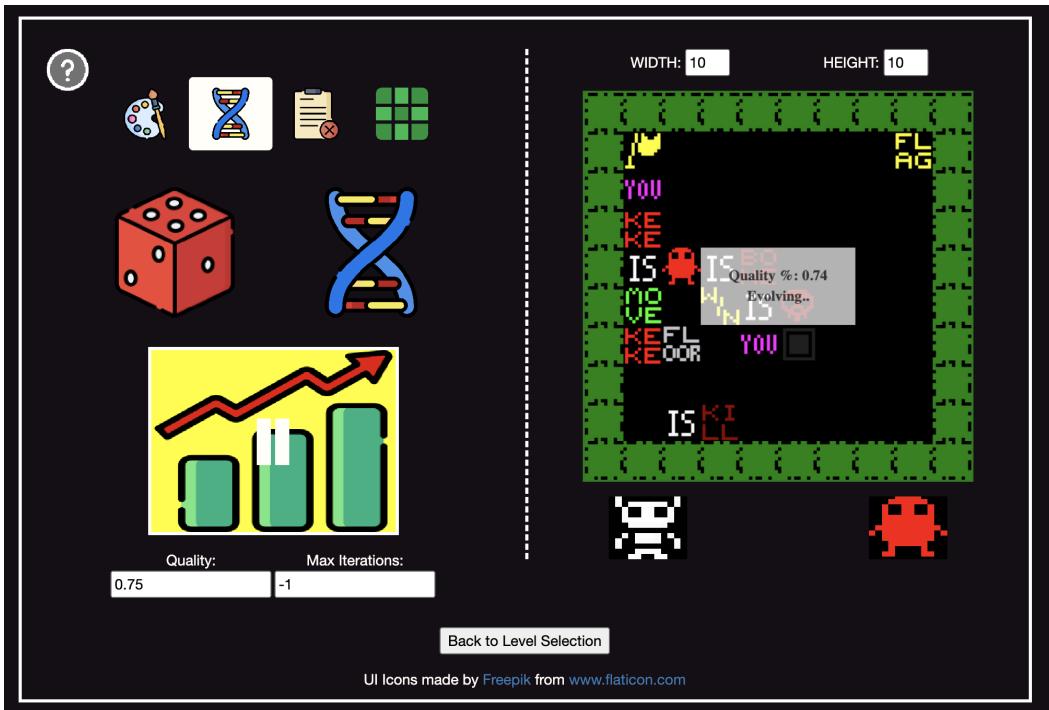


Figure 4.4: Mutator Module

the Kullback-Liebler Divergence between both probability distributions.

The mutator module mutates maps by randomly replacing 3x3 window tile sections since rules in *Baba is You* require 3 horizontal or vertical tiles. These sections are replaced with sections from elite maps to create maps of similar aesthetic quality to that of previously submitted maps. The generator uses the following fitness function:

$$f = \min(f_{kl}) + u + p + 0.1 \cdot s \quad (4.1)$$

The fitness of a level is intended to minimize the number of *useless* sprites (u) – object and word sprites that are not required for the solution or that cannot be manipulated in the level, check for playability of a level (p) – if the level contains a “WIN” and “YOU” rule in order to trigger the win condition, and having an appropriate ratio of empty spaces in the level (s) to prevent too much noise and sparsity. These properties are combined with the original ETPKLDiv fitness calculation by Lucas and Volz (f_{kl}) that checks the map similarity against previously submitted maps in order to retain aesthetic quality to form a final

fitness value. The minimum f_{kl} value is used based on the comparison of the edited map to all submitted maps in the system database.



Figure 4.5: A sample population selecting a map with the best fitness value during an evolution iteration

Figure 4.5 demonstrates an example of the selection process from the evolutionary algorithm's population using the fitness function defined. For this example, the selected objective rules the user wants the level to evolve towards is to contain or potentially contain the rules 'X-is-STOP' and 'X-is-MOVE.' The values at the top of the maps represent each map's fitness value. The first map from the left has a bad fitness value from having too many irrelevant sprites, not fitting the objective criteria, and is definitively unsolvable, since no 'WIN' sprite tile or 'YOU' sprite tile exists. The middle map is substantially better and fits the objective criteria, but has many unreferenced tiles (as no word sprite block for 'GRASS', 'LAVA', or 'LOVE' exists on the map.) The evolver is intended to maximize towards minimalism, so it would likely try to remove these sprites. However, these tiles could arguably serve as decoration instead and make the level more aesthetically pleasing and a user could edit them manually back into the level if they wanted. The two levels on the far right are also lower in quality and fitness from not including the word blocks necessary to match the objective request and from having too much empty space. Therefore, the second map from the left will be selected to be shown to the user during the evolutionary process and selected as the most viable map for the user.

Figure 4.4 shows a screenshot of the mutator screen on the website. Users can click the graph to allow the evolution to edit the map. The dice icon will randomly

change tiles in the map and the DNA icon will replace a section of the map with a section from a randomly chosen Elite map. Users can also specify the fitness value and maximum number of iterations to let the evolver run for.

4.2.3 Objective Module

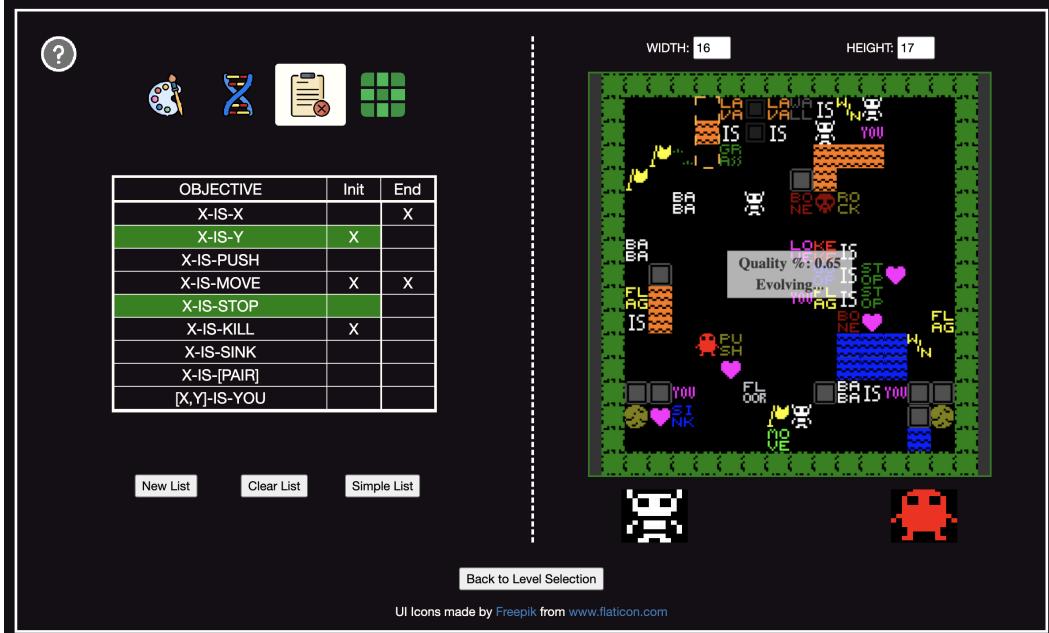


Figure 4.6: Objective Module

The objective module helps the mutator module evolve levels towards a specific set of starting and ending rules in the Baba is You game. The mutator module will add relevant keywords (e.g. “KILL”, “SINK”, “PUSH”) in order to encourage the activation of these rules and fulfill the objectives. These rules can either be specified by the user in the rule selection table by selecting from the options specified in Table 4.1. The objective module can also reference the Map module to guide the generation towards unfilled MAP-Elites matrix cells for the mechanic combinations. Levels are ranked for utility in the mutator module’s fitness function based on the prediction of a rule to be activated and similarity to previously submitted levels. This module was intended to encourage more diversity with the levels submitted to the system while allowing the mutator module to utilize dynamically added content from the growing database to generate new content.

Figure 4.6 shows the objective module screen. Here users can toggle on or off which objectives to have the mutator module and map modules prioritize for evolution and selection respectively. Two modes are available: an advanced mode that specifies the start and end objectives, and a simple mode that allows an objective to be active at the beginning or end. Users can also clear the list or use a random selection of objectives that have missing level cells in the Map module.

4.2.4 Map Module

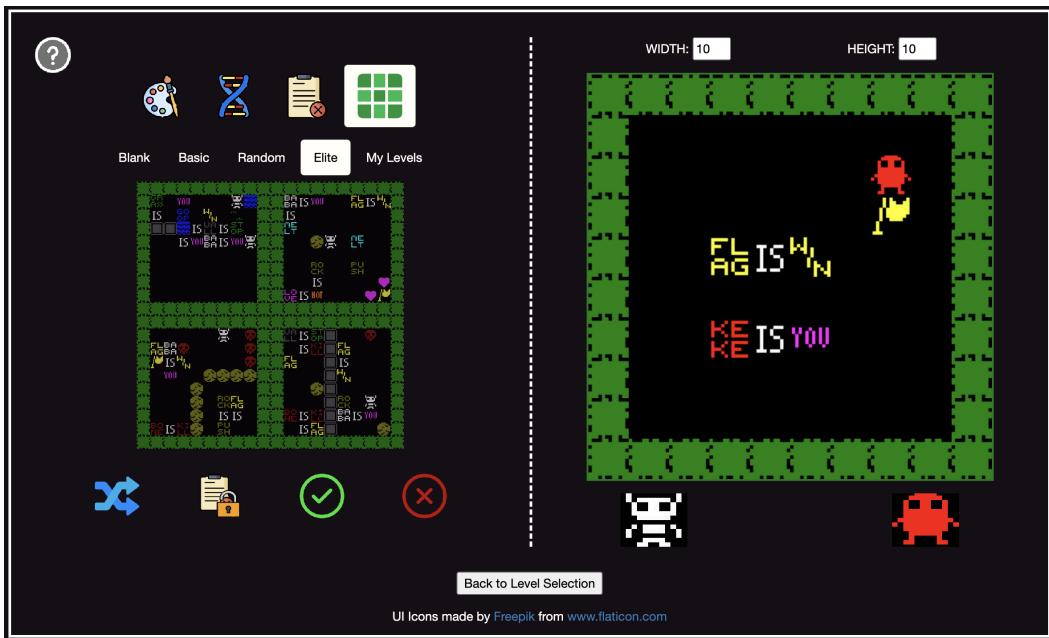


Figure 4.7: Map Module

The Map module acts as a core for maintaining diversity in the Baba is Y'all system. It utilizes a MAP-Elites system, similar to that of Mech-Elites, and arranges the levels based on the Baba is You property rules that are active at the start and end of the level. The system checks which of the 9 rules (see Table 4.1) are active at the start and or end of the level, thus creating a matrix of 2^{18} cells total. The quality of a level – which determines whether it is an elite for a cell – is based on the rating of the level evaluated by users with the rating module (see 4.2.4.1). Elite levels are sampled by the mutator module for mutating tile patterns and as comparison for similar objectives (see 4.2.3). The Map Module also shows

users which cells have yet to be filled on the home screen of the Baba is Y'all site to encourage users to create levels fulfilling the missing rule mechanic combinations. This global objective of designing the missing levels via mixed-initiative human-AI collaboration is what makes Baba is Y'all distinct from other community-based game level editors and quality diversity generators. The Map Module encourages the expansion of the rule mechanic combination space while creating a feedback loop to maintain aesthetically pleasing and diverse levels.

Figure 4.7 shows the map module screen. Here users can select from the blank, basic, random, elite, and profile account levels to use as a starting point for their map design.

4.2.4.1 Rating Module

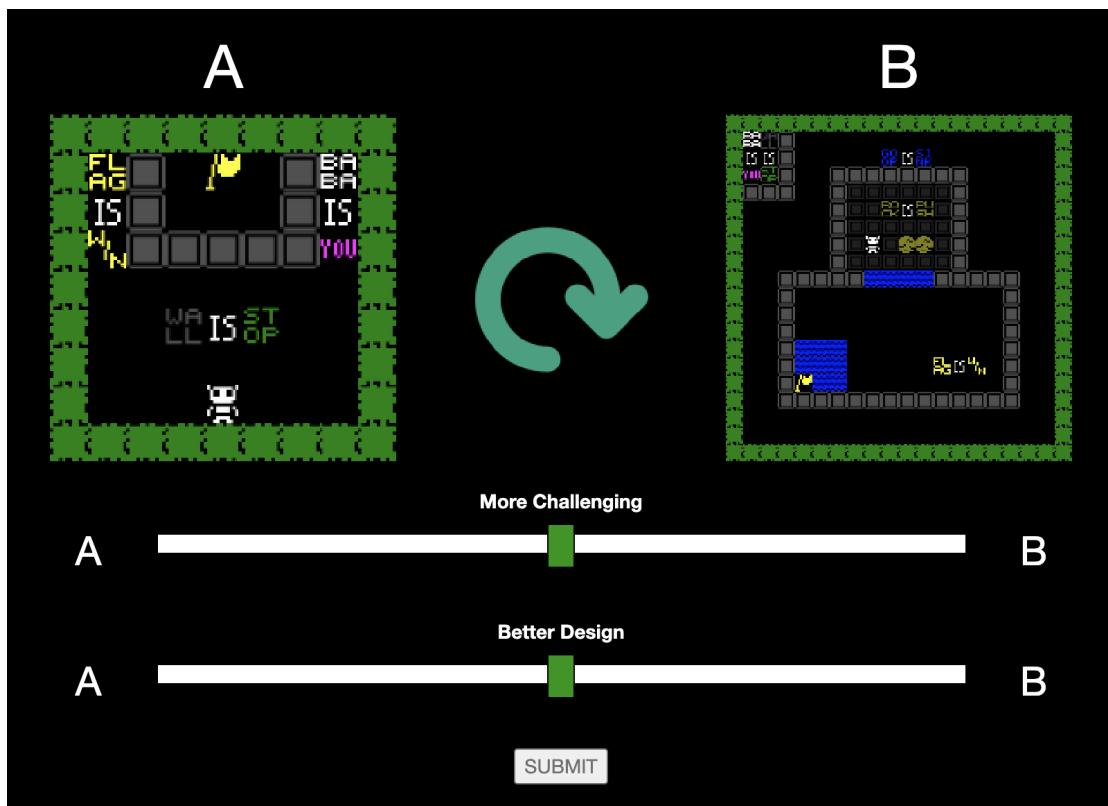


Figure 4.8: Rating module in the Baba is Y'all version 2 site

The quality of a level is determined by user evaluation in the rating module. Levels submitted to the database are evaluated in an A-B comparison. The user determines which of the two levels is better based on challenge level and quality of

aesthetic design. Both levels can also be played in the game module (see [4.2.5](#)) before rating. Figure [4.8](#) shows the rating screen comparing two submitted levels with the criteria sliders.

4.2.5 Game Module and AI Solver

The game module runs the Baba is You game engine for a level. When the “WIN” condition has been met for the level, the game module saves the solution and the rules active at the start and end of the level to be interpreted by the Map Module (see [4.2.4](#)).

The game module also provides a client-side AI solver called “Keke” (based on the sprite character traditionally used as an NPC in the Baba is You game) that uses a greedy best-first tree search algorithm to quickly attempt to solve a level. The branching inputs are based on the five directional inputs a player can perform in the game: left, right, up, down, or do nothing. The algorithm uses a heuristic function defined as follows:

$$h = (n + w + p)/3 \quad (4.2)$$

where h is the final heuristic value for placement in the priority queue, n is the minimum Manhattan distance from any player object to the nearest winnable object (objects with the property “X-IS-WIN” active), w is the minimum Manhattan distance from any player object to the nearest word sprite, and p is the minimum Manhattan distance from any player object to the nearest pushable object (objects with the property “X-IS-PUSH”).

As an update for BiY v2, the agent can run for a maximum of 10000 iterations and can be stopped at any time by the user. After stopping, a user may also attempt to solve part of the level themselves and the KEKE solver can pick up where the user left off to attempt to solve the remainder of the level. This creates a mixed-initiative approach to solving the levels in addition to editing the levels.

However, even with this collaborative approach, the system still has limitations and difficulty solving levels with complex solutions - specifically solutions that require back-tracking across the level after a rule has been changed. The solver runs on the client side of the site and is limited by the capacity of the user's computational resources.

4.2.5.1 User Profiles

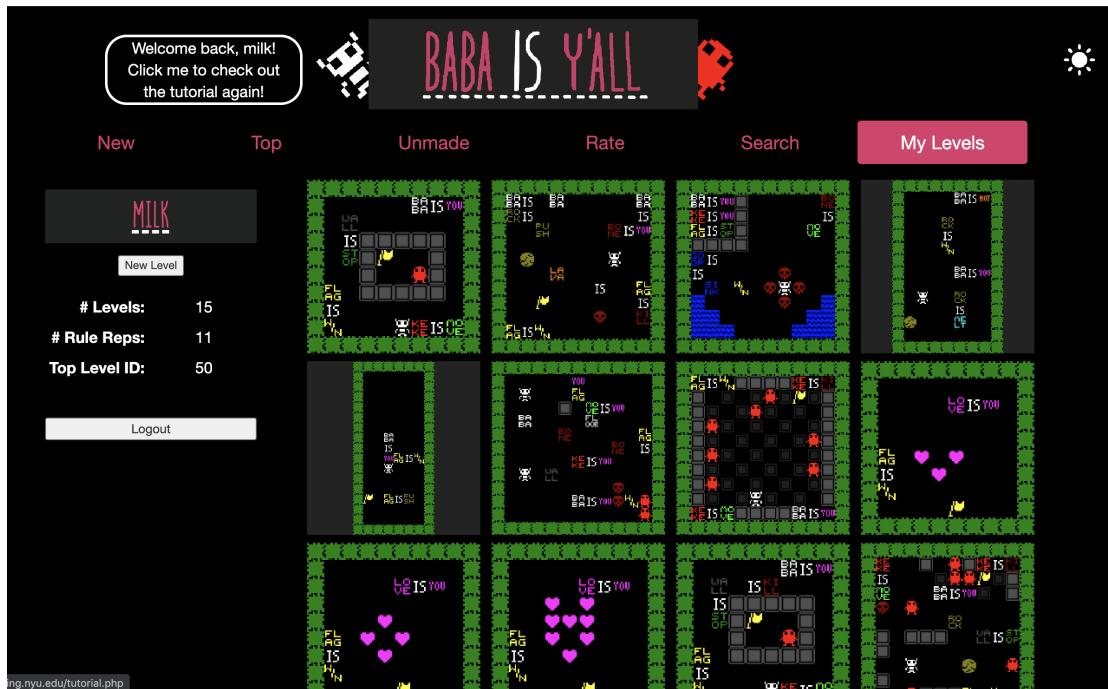


Figure 4.9: User profile tab on the main screen of the Baba is Y'all version 2 site

The Baba is Y'all website includes a registration system to allow users can claim authorship to their submitted levels. Users can also name their levels for added creative flexibility. When logged into their account, users can use their previously submitted maps as potential selections in the editor screen using the map module when designing new maps (see 4.2.4). Users can find their a list of all of their authored levels from their profile page on the main screen. This profile page also shows general stats about the user such as the number of levels they've created, the number of unique rule combinations they've designed with their levels, and their top-rated level. Figure 4.9 shows the user profile page listing the levels made by the author "milk".

4.3 Baba is Y'all version 1



Figure 4.10: Baba is Y'all Version 1 Main Screen (from April 2020)

The first version of Baba is Y'all (BiY v1) was released officially in March 29th, 2020, and promoted via Twitter. This version served as a prototype and proof-of-concept system for mixed-initiative AI-assisted game content collaboration specifically for designing levels in the game “Baba is You”.

BiY v1 (as shown in Figure 4.10) was a prototype example of a mixed-initiative collaborative level designing system. However, the site was limited by the steep learning curve required to interact with the system [22]. Features of the site were overwhelming to use and lack cohesion in navigating the site. A small observational study was run to examine the trends of submitted levels.

4.4 Baba is Y'all version 2

The second version of Baba is Y'all³ (BiY v2) was released on May 27th, 2021 and is designed to have a more user-friendly setup. Figure 4.11 shows the starting page of the home screen. It was similarly promoted via Twitter and on mailing lists. This version includes a cleaner, more compact, and more fluid user interface

³<http://equius.gil.engineering.nyu.edu/>



Figure 4.11: Baba is Y'all Version 2 Main Screen (as of September 2021)

for the entire website and consolidated many of the separate features from the BiY v1 site onto fewer pages for easier access.

In addition to updating the features and collecting more data about the levels created, we conducted a formal user study with 76 participants to gather information about which features they chose to use for their level creation process and their subjective opinion on using the site overall. This user study, as well as the general level statistics collected from the site’s database, showed that our new interface better facilitated the user-AI collaborative experience to create more diverse levels.

4.5 User Studies

4.5.1 BiY v1 Study

For BiY v1, an analysis on the submitted levels was conducted. After 32 hours online, the website collected a total of 58 levels – 12 user-only made levels, 11 AI-only made levels, and 35 user-AI mixed author levels. 38 cells of the possible 262144 levels in the MAP-Elites matrix contained a level. The ratio of mixed-

authorship levels to pure human or pure AI authorship levels was prompted us to continue studying the effect of human-AI collaborations as applied to level design.

4.5.2 BiY v2 Study

For BiY v2, a formal user study was conducted to study user motivation, design trends, and challenges with using the mixed-initiative level editor. Users were instructed to play a level already made on the site, create a new level using the level editor, test it, and finally submit it to the site. They were also given the option to go through the tutorial of the site if they were unfamiliar with the *Baba is You* game or needed assistance with interacting with the level editor tool. A total of 76 valid responses – validated by checking the level ID of the Google form response submitted and the level submitted to the site – were recorded for the user study.

Half of the participants reported playing video games for more than 10 hours a week with 80% of all participants playing at least 2 hours a week. However, 40.8% of participants had never designed levels for games. When asked if they preferred *making* puzzles vs. *solving* puzzles, 50% of participants preferred solving puzzles and 40% of users preferred both designing and solving puzzles. This difference in player preference vs. designer preference provided an insight into how users approach a level submission system – with many users most likely preferring to play levels and use content that is already made as opposed to creating it themselves. More than 90% of participants had played a Sokoban game, but only 30% of participants had played the game *Baba is You* before. The choice of domain for *Baba is You* may have also been a contributing factor to the learning curves and had given an incentive to use the assistive recommendation system. 75% of participants had never used AI-assisted level editing tools – potentially causing an even bigger learning curve for working with an AI collaborator for designing levels.

For the first task of the site interaction portion of the study, users were asked to solve a Baba is Y'all level and optionally go through the tutorial. Figure 4.12

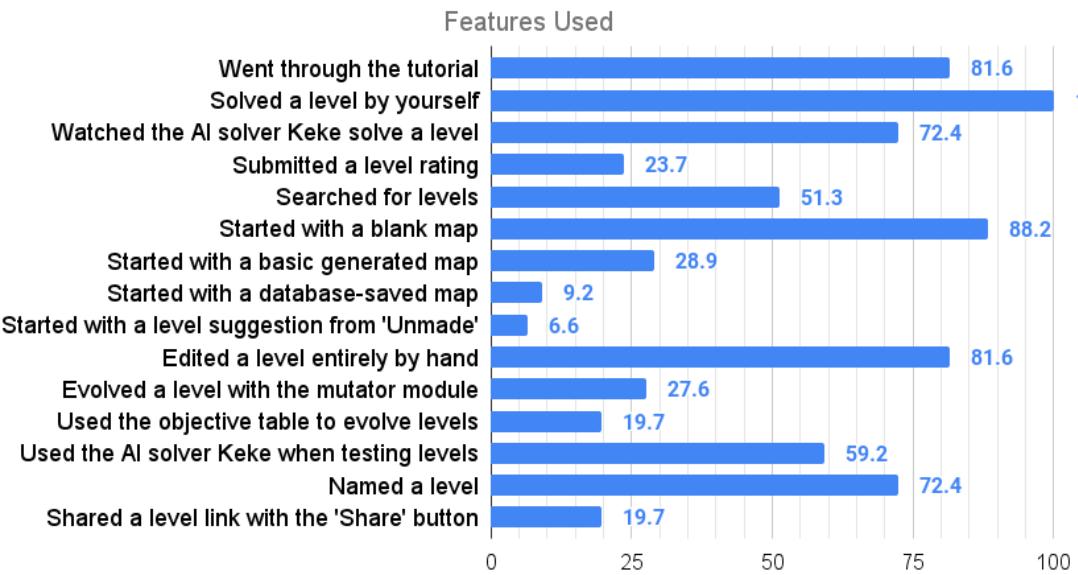


Figure 4.12: Survey results for users' reports on the features they used

shows a graph of the distribution of feature reporting. 100% of participants were able to solve a level on their own, but 74% choosing to watch the Keke AI solver complete the level as well. 81% of participants reported going through the tutorial section of the website. For the second task, users were asked to design a level with the Baba is Y'all editor system – some participants designed more than one level and thus had more interaction with the website's features. 88.2% of participants reported starting their level design process with a blank map and 6.6% of participants started with a level that was suggested by the Map Module to fill in the MAP-Elites cell with a missing rule combination behavior characteristic. Only 27.6% of participants reported editing a level with the mutator module and 19.7% of participants reported using the objective module. Conversely, more than 80% of participants created a map without any AI-assistance. It's possible that this low percentage is attributed the fact that a majority of participants were unfamiliar with AI-assisted systems or the *Baba is You* game overall. This - as well as the lack of selection for level comparison from the previously submitted levels in the database - made using the evolver tool for certain goals too steep of a task to accomplish and learn.

Statistic analysis of the levels submitted by the participants were done similar



Figure 4.13: Sample levels generated for the system. The left column is user generated levels, the middle column is evolver module levels, and the right column is mixed-initiative user and evolver levels

to that of BiY v1. Figure 4.13 shows some sample levels submitted by participants for each authorship type. Unlike the first version of the site, 70% of the 96 levels submitted by the participants were designed solely by the user with only 26% of the levels having mixed user-AI authorship. However, there were 57 unique rule combinations from the submitted levels with the mixed-authorship levels having the most diversity of rules combinations. In this sense, the AI-assisted system was successful in encouraging diversity with the level design process, but lacked in incentivizing site participants to interact or use it. Figure 4.14 shows the

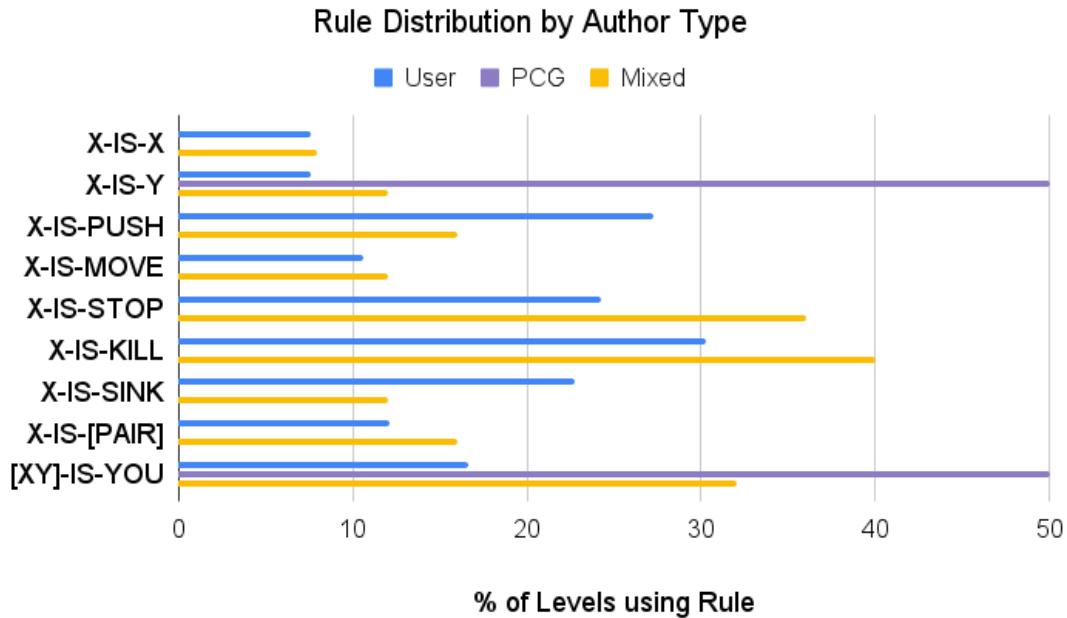


Figure 4.14: Distribution of rule types in the Baba is Y'all v2 system by authorship

distribution of rules by authorship.

While not required in the tasks given, we also asked participants about any extra site features they chose to explore. 23.7% of users reported submitting a level rating from the “Rate” page. 51.3% of users reported using the “Search” tool to search for specific levels (what their search criteria was we did not ask.) Finally, 19.7% of users reported using the “Share Level” feature to share a submitted level link with others online.

4.5.3 Demographic and Level Submission Correlation

	Baba is You?			Puzzle Level Preference			AI-Assisted Level Editors?		
	Yes	Maybe	No	Solve	Design	Both	Yes	Unsure	No
# Mixed Levels	9	6	7	12	2	8	3	0	19
# User Levels	14	10	28	25	5	22	11	4	37
# PCG Levels	0	1	1	1	0	1	1	0	1
Avg # Rules	1.96	1.53	1.67	1.63	0.57	2.10	1.53	3.75	1.63
Avg Sol. Len	30.87	30.65	28.47	32.08	25.14	27.77	24.00	43.50	30.21

Table 4.2: Level statistic between selected questionnaire responses and submitted levels

Table 4.2 shows the raw results from 3 selected questionnaire response groups

concerning their experience with *Baba is You*, AI-assisted systems, and the preference for designing vs. solving puzzle levels. Of the 36 users who had never played or heard of *Baba is You*, 77% chose to make a level that was exclusively user-authored. For the other two groups – those who had heard of the game but never played it and those who had played the game before – 62.5% and 56% of users respectively chose to make user-only levels. 41% of the mixed-authorship levels submitted from the participants of the study came from the group who had played *Baba is You* before. From this we can infer that those who were more familiar with the game environment were more likely to use the AI assisting tool to design their levels. Users who had played the game before also had the highest average for number of rule combinations in their levels at 1.96.

We also observed some correlations between level submissions and puzzle level preference. 60% of the participants who answered that they preferred to solve puzzle levels submitted user-only authored levels - however this group also held the majority for number of mixed-authorship levels submitted at 54.5%. This group had the highest solution length average of the 3 groups at an average of 32.08 steps. 28.5% of participants who answered that they prefer to design levels submitted mixed-authorship levels to the site - the highest percentage of the 3 groups.

An interesting correlation was found from users who never had experience with AI assisted level editors before. Of the 57 users from this group, 74% chose to submit a user-only authored level. However, 86.3% of the mixed-authored levels submitted from the study also came from this group. Only 20% of users from the group who had experience with AI-assisted level editing tools chose to submit mixed-authorship levels. From this we can infer that users with little experience in AI-assisted tools are more willing to experiment with them and try to design with them.

4.6 Summary

The first iteration of the Baba is Y'all website (BiY v1) served as a proof-of-concept prototype and was fairly successful in facilitating user-AI collaboration to create crowd-sourced levels [22]. The second iteration of the Baba is Y'all website (BiY v2) focused heavily on creating a user-friendly website with seamless transitions between editing levels manually as a user and procedurally with the AI interaction [20]. Additional features were added to the system to improve usability, such as a tutorial, a help menu, controls for the AI evolution, and more. A user study was also performed to gather more insight on user intention with designing puzzle levels collaboratively with AI in the system.

The results from the user study have demonstrated both the benefits and limitations of a crowd-sourced mixed-initiative collaborative AI system. Currently, users still prefer to edit most of the content themselves, with minimal AI input - due to the lack of submitted content and ratings for the AI to learn from. Pretraining the AI system before incorporating it into the full system would be recommended to create more intelligent systems that can effectively collaborate with their human partners for designing and editing content. This would lead to more helpful suggestions on the evolver's end as well as better designed levels overall. These projects and studies began the start of a much longer and bigger investigation into online AI tools and user-AI collaborative content generation.

4.7 Reflection

It is clear from both the submitted level statistics of the site and the self-reported user survey that mixed-authorship is not the preference for users when designing levels. For this domain, many users would still prefer to have total control over their level design process from start to finish. Users lacked direct creative control over the evolutionary process and attempting to use them in middle of creation might have ironically been more problematic. The AI-assistant could potentially

work against the intent of the human designer by destroying particularly placed elements.

The limitations of the AI back-end (both the evolver and solver) may be at fault for the lack of AI interaction. The mutator module was dependent on previously submitted levels and level ratings in order to “learn” how to effectively evolve levels towards high quality design. The fitness function defined for the mutator module was also inadequate for level designing from an aesthetic and creative intent perspective. It could produce a level that is deemed “optimal” in quality by its internal definition, but may actually be sub-par in quality for a human user. The Keke AI solver was also lacking in performance, as a few participants reported that the solver was unable to solve their prototype levels that they themselves could end up solving faster than the AI could.

While the results of the statistics on the levels submitted were disappointing for involvement of the AI assisting tool, we also asked users how likely they would continue using the site after the experiment. 38.2% of users said they would continue to use the site, while 55.3% said they would maybe use the site. Many users were optimistic and encouraging with the concept of incorporating AI and PCG technologies with level design - citing the project as a “cool project”, “a very unique experience”, a “lovely game and experiment”, and “very fun.” At the time of writing, a few users did return, as their “Keke” assigned usernames were shown as authors on the New page, long after the study was completed. Most notably, the participant with the assigned username Keke978 who took up the username “Jme7” and contributed 28 more levels to the site after the study was concluded. However, for a majority of the system’s lifetime, I was the main contributor of the website’s levels, ratings, and rule diversity.

Many users also provided us with constructive feedback for feature implementation, site usability, and suggestions for improvement with how to further incorporate the AI back-end interactivity. 70% of users who played with the system had never played the game “Baba is You” and 75% of people had never used an

AI-assisted level editor tool before this experiment. Based on this information and retainability of users to complete the survey and provide the constructive feedback, we can extrapolate 2 conclusions: 1) the game stands alone, independent of *Baba is You*, as an entertainment system; and 2) for people with even limited AI-gaming experience but some game experience, this experimental nature of the project with that involved the AI back-end systems is engaging to users.

I chose the Baba is You game as the domain for this system instead of standard domains in the game-AI research field such as Sokoban, Mario, and the GVG-AI framework for 2 reasons. The first was from miscommunication with my advisor in a meeting; he was recommending that I play the game Baba is You after he pitched the concept of this project. The second was because I was tired of seeing these same set of domains in research papers just with different methodologies applied to them. When I made this project, the Baba is You game didn't have a level editor, which I think might be another reason people chose to interact with the Baba is Y'all system.

I also wanted to make a system where anyone could use it, because I believed it would get the most interaction and content out of it. The reason why Wikipedia – an inspiration for this project – is so popular and utilized is because anyone can add content to it and anyone can read articles on it. The system isn't limited to just experts in the field, just like how Baba is Y'all isn't limited to game designers or AI scientists. With Baba is Y'all, anyone can open a browser and design a level with an AI – it doesn't require a specific operating system or extra libraries to install or supercomputers with multiple GPUs. This ideology remained a driving principle for the following projects in this thesis. To get the most interaction from a human-computer interaction project, it needs to be as accessible as possible; not just on a hardware level, but on an intuitive level.

This work was probably the most influential to the rest of the projects in my thesis. It showed many flaws and limitations with the AI and collaborative content generation systems. The isolated exploration of these problems inspired

the development of many projects such as Aesthetic Bot, the Keke AI Competition, and the Minecraft Interactive House Evolver. Users lack incentive to collaborate with AI, especially if they have their own creative goal or design plans in mind. The AI isn't strong enough to work as an effective collaborator with users yet, but feedback from the users is also needed to understand how to improve the AI to match their creative needs and intents.

4.8 Takeaways

- Online content generation systems engage users more than stand-alone interactive software
- Exploring a familiar domain that previously did not allow for content creation (i.e. level editors) also engages users
- This AI system, while accessible, did not encourage increased user interaction
- Intuitive interfaces help facilitate better human-AI interaction for those unfamiliar with collaborative AI systems
- If a PCG system does not meet user expectations with its output, then users will default to manually creating the content themselves
- A pre-set objective function for a PCG system may not be what a user actually wants out of a content generation system; one size does not fit all
- This recommendation AI system was not strong enough to encourage users to create novel levels with diverse mechanics for the game Baba is You
- Many users chose to use the site as a way to design Baba is You levels and post them online – similar to that of Super Mario Maker – disregarding many elements of the AI system
- Collaboration from multiple users for content generation and creation provides more diverse data than pure PCG or pure AI techniques

Chapter 5

Crowd-sourced Data for Game-Playing Agent Training

The ongoing submissions of puzzle levels to the Baba is Y'all database showed an increase in levels that proved to be diverse in both aesthetic design as well as solution complexity. This system provided an openly available resource of puzzle levels and their solutions to a game with dynamically changing mechanics and rule combinations. This domain and dataset provided an interesting challenge for AI solvers in the game AI research field.

From the Baba is Y'all version 2 user study, 59.2% of users reported using the Keke solver AI when testing their levels [20]. However, the solver within the system itself was lacking in performance and further limited the incentive for users to collaborate with the AI system for the development of their level editors. This also prohibited the system from being able to automatically generate *Baba is You* levels that were both complex in solution and diverse in mechanic space using only quality-diversity PCG methods.

As a result, the Keke AI Competition [25] was launched to search for a remedy to the limitations of the site's built-in solver. This competition also demonstrated the utility of how a diverse, crowd-sourced database of levels could be beneficial to game AI research – particularly with *Baba is You* game mechanics as a challenging

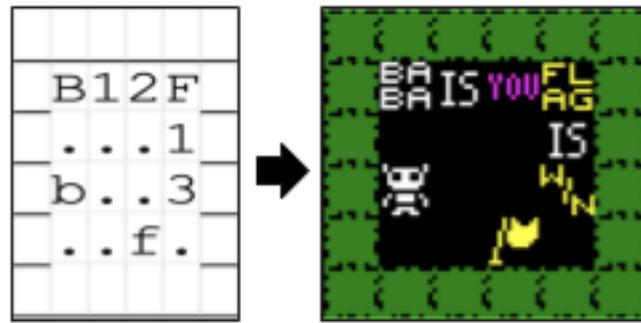


Figure 5.1: Example of the ASCII representation of the Baba is Y'all level map as interpreted by the framework engine

AI solver domain.

5.1 Competition Framework

The Keke AI Competition framework contains a simulator engine with state representation functions, a graphic interface evaluator that allows offline testing, and example baseline agents and level sets. The framework is written in JavaScript to work with the Baba is Y'all level editor web application.

5.1.1 Simulation Engine

The Keke AI framework uses the same game engine and ruleset mechanic definitions found from the Baba is Y'all website system (see Chapter 4.) The Keke AI Competition framework source code uses a state representation of the map similar to the Sokoban framework code by Khalifa [63]. The state itself contains an ASCII representation of the map (see Figure 5.1), a list of active rules, two lists of interactable and non-interactable objects, a list of keyword sprites in the map, and lists of object sprite assigned by attribute according to the currently active rules in the map. In other words, the states contain separated lists of active sprites that can be *PUSHed*, can *KILL* the player, can cause sprites to *SINK*, can *MOVE*, etc. Each state allows a look-ahead step that is passed one of 5 actions (up, down, left, right, or no movement) to allow for tree search agents to explore

future states. The framework also includes state modification functions, such as copying a state, setting a state to a passed ASCII map's setup, resetting a state to the starting game map's setup, and more to allow for potential deep learning training in future extensions of the framework.

5.1.2 Developing and Testing Agents

The Keke AI Competition framework contains six example agents that participants can use as a baseline to create their own agents. These agents include three tree search agents using BFS, DFS, and the current solver agent used for the Baba is Y'all web application — which uses a greedy best-first search algorithm, an agent that outputs a random sequence of 50 actions, and two example agents to function as a “DoNothing” evaluator and a broken agent that would not successfully pass the evaluation check upon submission – to give users an example of an invalid agent definition.

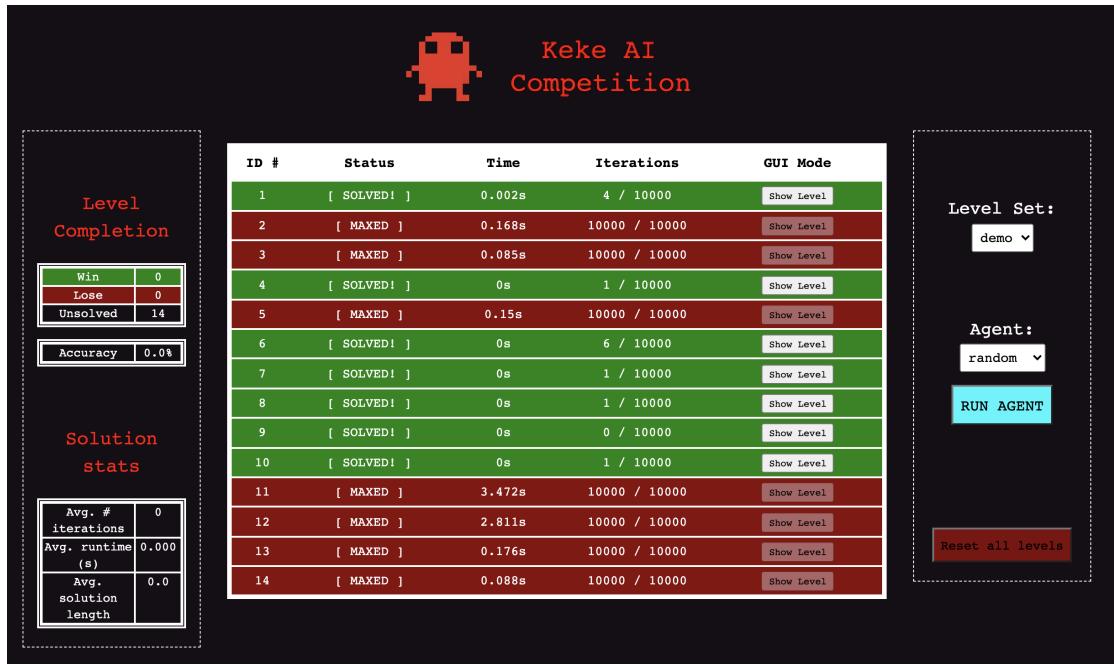


Figure 5.2: The evaluator screen of the offline Keke AI Competition offline interface.

5.1.3 Offline Evaluator Interface

Users can test the agents via the offline web interface provided by the framework. Here users can test and view the results of their agents' solutions on specific level sets. The interface also provides a GUI rendering of the level and simulates the agents' solutions on a particular level as if it were in the Baba is Y'all website. Figure 5.2 shows a screenshot of the offline interface with the indicators for solvability success, drop down menus for the agent and level set, and the statistics of the agent. Users can select the agent or level to test on from the dropdown menu at the right of the screen. To the left, stats about the agent's performance are updated as levels are solved. In the middle, users can view the solution length, time, and success of the agent on a particular level and see the solution if the agent was successful. This visual interface was implemented to provide more ease of access for the user and provide a lower barrier of entry to users who may not be as familiar with AI competitions in general. The interface also helps users identify quickly which levels were solved by the agent and how well they performed better than reading from a JSON report or a terminal output could.

5.1.4 Level Set Data

The training data level sets are JSON files containing ASCII levels created from the Baba is Y'all database. Users can create their own training sets for their agents by extracting levels directly from the Baba is Y'all website – creating a constantly updated, accessible, crowd-sourced dataset of training data. Users can extract levels from their own profiles, from a search result, or from the entire database. The level tester screen of the Baba is Y'all site also allows users to export singular levels in the JSON format - whether they are submitted to the database or not.

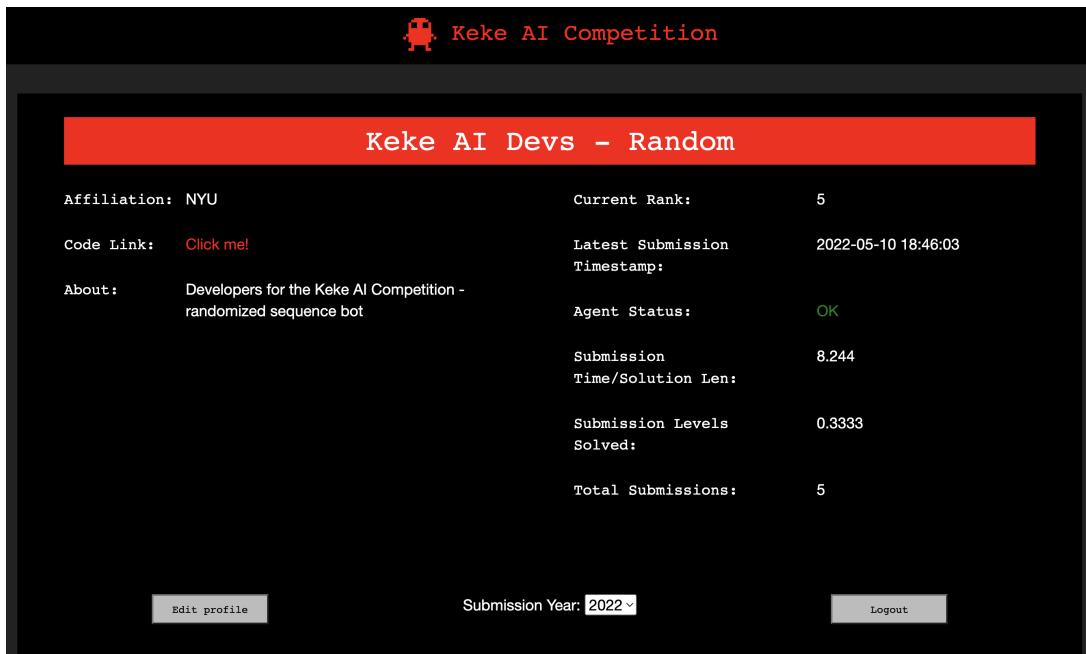


Figure 5.3: A screenshot of an example profile page on the Keke AI Competition website. Here, teams can view such as statuses such as rank, submission timestamp, submission count, and more.

5.2 Competition Website

Agents are submitted to the competition website¹ hosted on an Amazon Web Services Ubuntu server and evaluated automatically by back-end scripts monitoring the website. Participants are required to register a team under a email-password login system and could provide further details and information about their team on the profile page. After registering their team, participants can submit their JavaScript agents and the server would evaluate them on a set of test levels. Figure 5.3 shows an example profile page for the “Keke AI Devs” team and editable information fields. Users can navigate through the main screen of the website to find any necessary links they would need for the competition, including links to social media, the framework, the Baba is Y'all site, the wiki, and a walkthrough video for submitting an agent. The site also features an about page with rules for submission.

¹<http://keke-ai-competition.com/>

5.2.1 Competition Evaluator

The website has a readily available, constantly updated leaderboard that showed each team's agent performance on the evaluation levels. These levels are not included on the original Baba is Y'all website and cannot be downloaded or explicitly trained on by participants. Teams are ranked on agent performance that is evaluated in the following order:

1. *Code error* - agents that error out in the pre-processing step are automatically placed at the bottom
2. *Average time⁻¹ / solution length* (referred henceforth as timesteps) - agents that give shorter solutions solved in quicker time are ranked higher.
3. *Time submitted* - in the case of a tie in all other categories, the agent submitted earlier is ranked higher

Average time⁻¹ / solution length was designated as the main evaluation metric for the competition to encourage participants to create agents that could find solutions to the puzzles in the fewest amount of steps and in the shortest amount of time possible.

Evaluation of the agents is completely autonomous and handled by the website server. Once a team submits their agent, the JavaScript file is verified to have the required functions necessary to run the agent on the evaluation framework. After this check, the server will directly test the agent on the evaluation levels and save the results of the team's agent to the database to be passed to the leaderboard.

5.2.2 Evaluation Levels

The evaluation levels were designed to challenge specific solver styles, such as requiring precise agent movement, using word blocks as both rule modification elements and as physical blocks, a maze designed to exploit tree search agents, and levels with an overwhelming number of irrelevant rules designed to fool heuristic

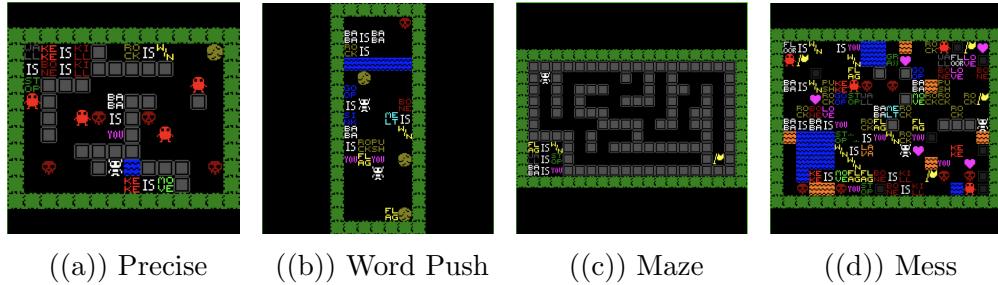


Figure 5.4: Sample levels from the evaluation set of the Keke AI Competition - each designed to exploit different agent search algorithms

based agents. The diversity of level design types is also present in the Baba is Y'all website itself via level submissions and is the reason why the original website's solver agent was lacking in performance. Therefore, it was crucial that the competition encouraged more complex agents that were capable of solving these dynamic level types.

5.3 Results

The leaderboard initially had the 4 baseline agents included the framework: the BFS-algorithm agent, the DFS-algorithm agent, the random sequence agent, and the default (Baba is Y'all solver) agent. Out of the 4 baseline agents the default agent (the original solver from the Baba is Y'all website) performed the best with 53.33% of evaluation levels solved and average of 34.123 timesteps. In second and third ranking were the tree search agents, the BFS and DFS agents. Lastly, the random agent performed the 4th best, solving 33.33% of levels - the same as the DFS agent - but had a lower average timestep of 8.244 and therefore was ranked lower. The competition received very few submissions overall, with only 4 total agents submitted from teams, none of them outperforming the baseline agents.

Future iterations of the competition will include a Python framework to allow for more accessibility and a more familiar and widely used programming language for participants. Since JavaScript was not as well known or widely used in the AI and Games community as compared to Python, this could create a larger learning curve and deter participants from trying to develop their own agents.

5.4 Summary

This work looked to solve the limitations of the built in Baba is You puzzle game solver and utilize the ongoing data collection from the Baba is Y'all level editor to start an AI solver competition focused on games with dynamic mechanic systems. From the competitive results of the competition, the agents submitted could then be directly placed back into the website to improve the human-AI designer collaboration in the Baba is Y'all website. Despite the low submission rate, this introduction of this competition allowed new and innovative agents to emerge from the Baba is You dynamic puzzle gameplay while demonstrating the utility and solution diversity of the crowd-sourced submitted levels on the Baba is Y'all website.

5.5 Reflection

I was disappointed with the lack of submissions and results from users – even with the video tutorial and wiki guide. It could have been because of the domain itself, the lack of publicity for the competition, or the ironic inaccessibility of having the agents and engine written in JavaScript instead of Python. I believe that giving participants the option to make their own training data from the levels on the live Baba is Y'all website – a system with constantly updating content and new challenges – makes this particular competition an open-ended problem for AI research. Years after the competition's inaugural release, people still ask if the competition will be hosted again or if a Python track can be made. I'd like to continue to work on this system, because it's still relevant and difficult for AI solvers and AI competitions in general. The level data authored by both humans and AI from this mixed-initiative collaborative process could be useful for the study of different types of AI solvers as applied to complex and dynamic problems in games. With this mixed authorship, novel and harder level types could be made that would prove to be even more challenging for AI solvers.

5.6 Takeaways

- Allowing users to make their own training data for an AI competition is beneficial for gameplaying agent development
- Automating the evaluation of agents for an AI competition saves time for the developers and organizers
- Human-AI content can be subjectively categorized
- Domains with dynamic mechanic spaces (i.e. Baba is You) make for challenging AI competitions
- Python is preferred for an AI competition
- Reference guides, tutorials, and example agents are beneficial for AI competitions
- The diversity of data from mixed-authorship content can create new AI challenges

Chapter 6

Crowd-sourcing Aesthetic Content Generation and Evaluation

The quality evaluation of Baba is Y'all levels relied exclusively on user feedback from the rating module. In both versions of the systems, users were not as incentivized to contribute evaluation ratings on the levels. As a result, the mutator and map modules suffered in quality and utility since their fitness recommendation and evolution algorithms depended on these ratings. The output of these two modules – their level recommendations and tile pattern changes – were often not beneficial to users, and thus not used by the designers. Two issues in procedural content generation were highlighted from the Baba is Y'all version 2 user study. First, there is a need for PCG systems that are capable of generating samples that are not based on practical evaluations such as playability, tile entropy, and/or sprite and game element composition but instead the subjective aesthetic design of the sample. Second, there is a need for a system that can allow for extended interaction and feedback from human users over a long period of time in a manner. In particular, a system that does not instigate user fatigue from continued and repetitive evaluations of generated samples - as present in other mixed-initiative and creative content research works.

The Aesthetic Bot system is a novel procedural content generation system that

focuses on improving only the visual design of a game’s map by learning general user preferences of aesthetic. To learn this preference, this system retrieves user feedback in a quick, minimal effort method of votes in an A vs. B comparison. The social media site Twitter was an ideal host for this system due to its large user base that frequently interacted with the platform. With this platform, Twitter users could quickly judge and vote directly on the bot’s generated maps or real, user-made maps within their regularly updated content timeline. After the polling, the system learns from the rating to generate new maps that are optimized to win future preference pollings and improve its overall ability to determine aesthetic quality in a map.

6.1 Twitter Bots and the Twitter API

Twitter bots are artificial profiles made on the social media platform that post content either unprompted with automated posts or in response to other users on the platform [133]. Many bots are made to produce content for mostly entertainment purposes - such as a bot that combines emojis together to form new ones (@EmojiMashupBot). Other bots exist to interact with users once prompted to do so - such as generating on-demand haikus (@haikookies) [99]. However, both types of bots only act as one-way posters - they do not learn from the a two-way communication between the user and the bot itself. Much in the same way Twitter’s human profiles can gain information by posing questions and receiving replies or posting polls to receive votes on a subject, so too can a bot learn from these posts. With the Twitter API, Twitter can be used as a means of retrieving feedback in the same way as the previous interactive evolution co-creative systems but in a system that is already familiar and regularly used by participants.



Figure 6.1: The tilesets used in the Aesthetic Bot system. The tilesets are referred to as such from left to right: Zelda, Pokemon, Among Us, Pacman, Dungeon

6.2 Aesthetic Bot System

The Aesthetic Bot system pipeline uses an evolutionary algorithm to generate new game maps using five custom designed pixel-art game tilesets – Zelda, Pokemon, Pacman, Among Us, and Dungeon. Each tileset contains 16 8x8 pixel tiles - some purposefully designed for inter-connectivity and to be placed in such a way to form a larger pattern of tiles (i.e. the tree tiles in Zelda.) We chose to use very small tileset visual dimensions so that users would ideally have a lower amount of visual aesthetic bias towards a particular art style or game due to higher resolutions or more complex graphics. The smaller resolutions are also much easier to view at a glance when the pixels are scaled up. Figure 6.1 shows all five of the tilesets used in the system.

Unlike other PCCG systems that use evolutionary algorithms to check for calculable metrics in the map’s level design such as tile entropy or solvability of the level, the fitness function of the system uses a convolutional neural network called the Aesthetic Prediction Model (APM.) The APM consists of an extremely small neural network with 3 convolutional layers, each followed by a batch normalization layer, a ReLU activation function and finally a MaxPooling layer (except at the last ReLU which has a flattened, dense layer.) Instead of passing the entire map to the network, the map is divided into one-hot encoded 4x4 tile windows with 16 channels (for each tile in the tileset). Due to the small input size, the model only has 15,185 parameters and output shapes of (2,2,64), (1,1,32), (1,1,16), (16), and (1).

The model is trained by passing the encoded 4x4 sliding subsection windows of

a map as its input and using the percentage of votes the input map received in the Twitter pairing as its output y value. Figure 6.2 shows this process of taking two maps from a polling and processing them for training the APM. We train the model on the percentage of votes instead of the votes directly because we want to encourage the model to learn how to estimate which maps will win in any given pairing rather than on how many votes it would receive.

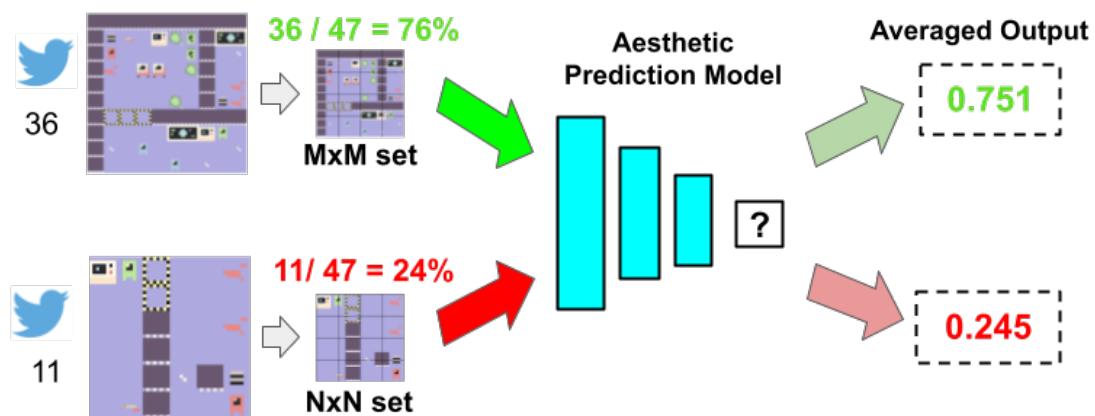


Figure 6.2: Diagram of the APM learning process on the sliding window tile patterns and vote percentages from Twitter.

6.2.1 Evolutionary Pipeline

To generate new maps, the pipeline implements a $\mu + \lambda$ evolutionary algorithm. The algorithm starts from a uniformly randomized population of maps of varying dimensions. The sizes of the maps can vary but are limited to a square range of 6-12 tiles in dimension. The APM – with the weights frozen – acts as the fitness evaluator for the evolutionary process, with the final fitness value for a generated map being the averaged output value of all sliding windows.

To mutate a map, random subsections of the map are replaced with a 2x2 subsection pattern from the real user maps saved to the system. We found that this method encourages the occurrence and retention of more “user”-made patterns with the direct replacement rather than randomly mutating singular tiles.

After a set number of iterations, the map with the highest fitness value (as determined by the APM) is selected from the evolutionary algorithm’s population.

This final map is saved to the database as the tileset’s generated map and labeled with the timestamp of the trained APM that helped with the evaluation to produce it. This is to keep track of which maps came as the results of a particular APM and monitor its improvements (or regression.) Figure 6.3 shows the entire learning pipeline generating and evaluating a new map that is sent to Twitter.

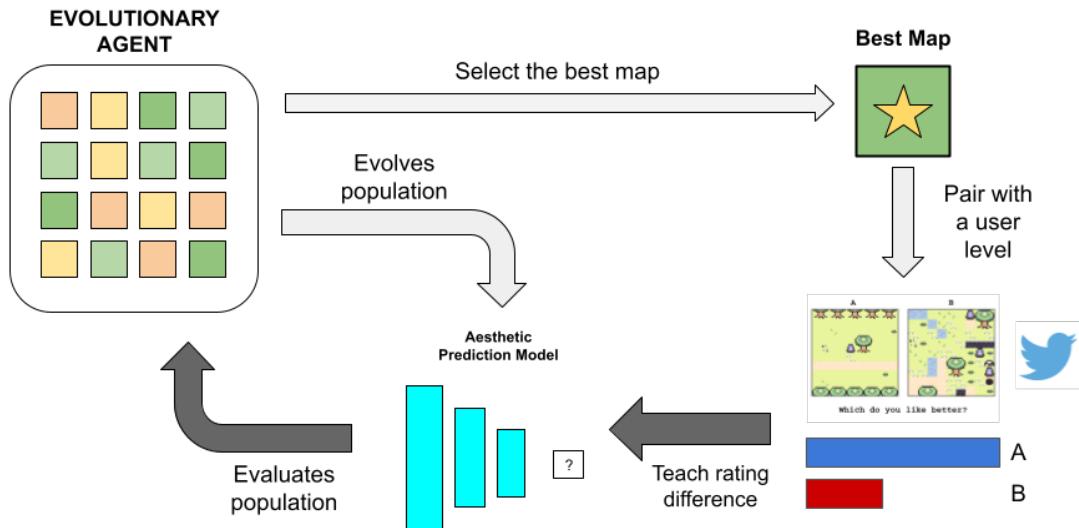


Figure 6.3: Evolution and learning pipeline of the Aesthetic Bot system

6.2.2 Pre-training Stage

Before pairing posts were made for the Aesthetic Bot Twitter account, the APMs for each tileset went through a pre-training stage to prevent completely random maps from being shown in the pairings and provide a “decent” looking map as comparison for users. A pseudo-Twitter evaluation function was used to emulate Twitter votes on a map and train the neural network. This function used a modification of Lucas and Volz’s Tile Pattern Kullback-Leibler Divergence (TPKL-Div) algorithm to evaluate maps [77]. The map’s Kullback-Leibler divergence score - or relative entropy - is used to compare against a training set’s tile patterns and determine the similarity in pattern compositions. This was to encourage more “user-made” looking maps while also encouraging novel design and structure. In addition to the TPKL-Div score, raw tile entropy probabilities were calculated (this

was to discourage maps with little variation in tiles or mostly empty maps) and multiplied together with the TPKL-Div score. The final value from this function was meant to simulate the number of votes a map received. Like the real Twitter polls, each map in a pairing received their votes (artificially calculated with the entropy and TPKL-Div) and the difference in votes was used as the output y value to train the neural network. By using an artificial function to simulate a base for user aesthetic, the neural network was able to train in a much shorter amount of time than the weeks or months it would take to train a randomly initialized network using live Twitter data.

Initially, the networks were pre-trained with a set of pairings of user-made and randomly generated maps so that the network would have a baseline output and learn to distinguish pure randomness before evaluating populations. Afterwards, the new maps were evolved in the evolutionary algorithm as normal with the pseudo-Twitter function acting as a last-stage evaluator to update the APM and decide which map was saved to be posted on the real Twitter. On completion of these iterations, the pre-trained APM was implemented into the actual pipeline and continued to update and learn using the real data polling from Twitter.

6.2.3 Twitter Interaction

A parallel, server-side Python script selects two maps of the same tileset – human-made or generated – and pairs them together. This pairing renders the two maps together, side-by-side, in an image with the label “Which do you like better?” underneath and saves the pairing information to another database table. The image is then posted to the Aesthetic Bot Twitter account with a poll as a threaded reply post using the Twitter API. The poll is held for 3 hours to allow as many users as possible to vote on the pairing. Upon completion of the poll, the votes are recorded to the database. The tileset’s associated APM is trained and updated with the results of the post’s pairing. Figure 6.4 shows a sample post and poll from the Aesthetic Bot on the Twitter page.



Figure 6.4: Sample Twitter post of the Aesthetic Bot system in with a user-made and artificially generated map being compared in a poll

A link to the Aesthetic Bot map editor is included in the post to encourage participants to upload their own map to the database to be used in a future pairing. Users could design a map using one of the five tilesets and provide their Twitter handle to be tagged in the post featuring their submitted map. The map editor interface - as shown in figure 6.5 - was designed to be mobile friendly as well so that users on the Twitter app could seamlessly submit a map. This design used the same codebase as the level editor from the Baba is Y'all version 2 website and featured many of the same tools such as the select, erase, undo, and redo features [20].

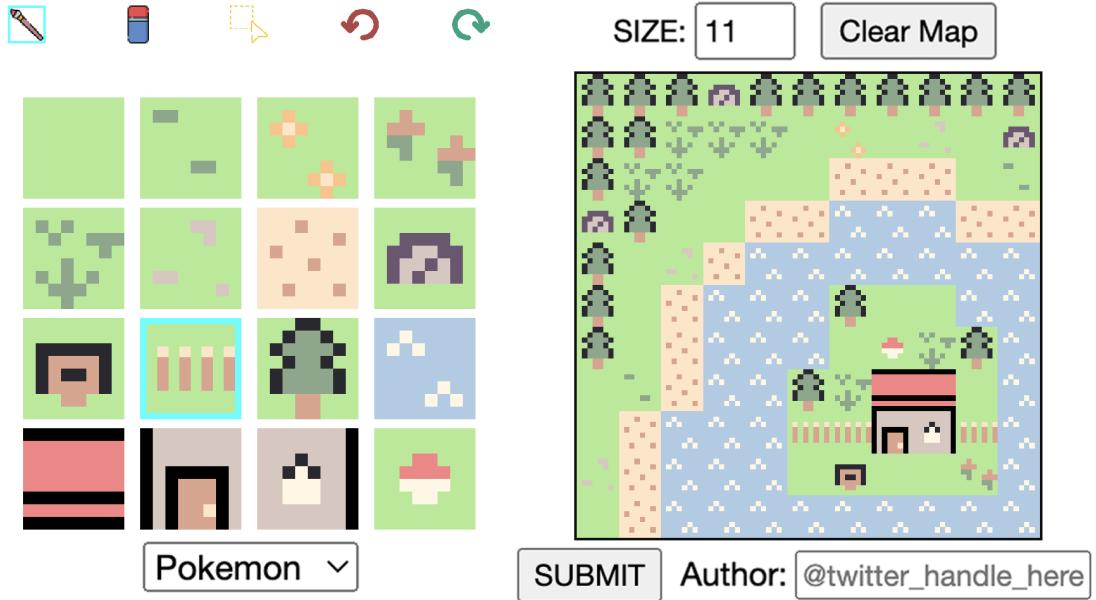


Figure 6.5: Level editor for the Aesthetic Bot system where users can design and submit their own maps to be used in future polls.

6.3 Results

By May 30th, 2022, there were a total of 399 user maps submitted with an average map size of 9.92 tiles squared. The distribution of maps included 101 user maps made for the Zelda tileset, 108 Pokemon maps, 31 Among Us maps, 69 Pacman maps, and 90 Dungeon maps. The popularity of Zelda and Pokemon maps may be due to the generic-ness of their art designs (a forest, adventure art style and a town-based, RPG style.) Meanwhile, the Among Us tileset had nearly a third of the number of maps of both tilesets, most likely because the actual game *Among Us* is the most unfamiliar of the 5 tilesets; being the most recently made and 20 years younger than all of the other games. The authors with the most submitted maps (excluding anonymously authored maps that made up 1/3 of the user maps) were @Nifflas, @MasterMilkX, dginev, @3phen, @AndresZarta, and @charphinB.

Out of the hybrid pairings, the average number of total votes per poll were 27.5 votes. Of the 220 total polls, the user-made maps won 74% of pairings, while the generated maps won 24%. 2% resulted in ties between the 2 maps. However,

percentage of votes the user maps received slowly decreased over time - as shown in Figure 6.6. The trend line (in red) shows the user maps slowly decreasing in vote majority. This implies that the generated maps are starting to receive more user votes over time and rival user-made maps in aesthetic quality.

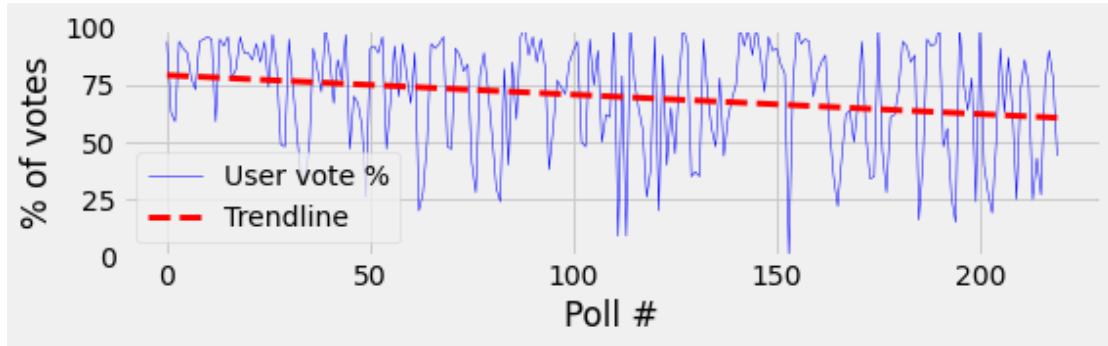


Figure 6.6: Percentage of votes received for the user maps.

A validation test was done on the APMs to test how well it could distinguish between “good” and “bad” maps over time. 3 different models at various stages of training: the pre-trained model before release to Twitter, the model 2 weeks after release, and the model 6 weeks after release. These were evaluated on 2 sets of maps: randomly generated (unevolved) maps and unseen user maps hand chosen and predicted to have a high aesthetic. Every tileset model - including those trained on Twitter were able to successfully predict the random maps at a lower winning rate (less than 30% of votes) - except the 2 week Pacman model. The models were also able to predict high vote rate (more than 60% of votes) for the unseen user maps - except the Among Us tileset, most likely because of the considerable lack of trained pairings. Thus, it can be inferred that most of the APMs were still able retain a high accuracy for predicting how likely a particular map would win a polling even after receiving new data from real polls. Table 6.1 shows the results of the averaged predictions from each model type on each of the tilesets.

By August 4th, over 3 months since the release of the Aesthetic Bot, further data was collected from the polling results. The total number of maps nearly doubled from 399 to 733. Understandably, the average number of votes per poll

Tilesset	Map type	U	P	2W	6W
Zelda	random	0.50	0.27	0.23	0.23
	user	0.50	0.52	0.67	0.66
Pokemon	random	0.50	0.22	0.15	0.22
	user	0.50	0.56	0.68	0.62
Among Us	random	0.48	0.30	0.33	0.31
	user	0.48	0.41	0.38	0.48
Pacman	random	0.50	0.13	0.49	0.35
	user	0.49	0.52	0.67	0.74
Dungeon	random	0.49	0.33	0.24	0.28
	user	0.49	0.50	0.71	0.72

Table 6.1: Average prediction values for each model trained on the pairing data ([U]ntrained, [P]retrained, [2 W]eeks, [6 W]eeks) for each tilesset. The values indicate the percentage of votes the maps are expected to receive.

decreased over time from 27.5 to 18.4. The user win percentage has increased to 78% as well, most likely due to the increase in quality from the user-made maps. The author @Nifflas has uploaded 135 maps out of the 733 submitted maps that have a poll win rate of 94% against generated maps and other user maps.

By the end of the study, nearly a year after release, a total of 935 user maps were submitted to the system with a total of 1973 map pairing polls recorded to the database. This data was made open-source in a Github repo¹.

6.3.1 Local vs. Global Evaluation Experiment

Initially, the APM architecture was designed with the sliding window inputs to account for the varying sizes of the maps. However, because there was substantially more training data since the initial experiments and release of the system, we decided to validate the original APM architecture against a convolutional neural network with a fixed input size.

The internal architecture of the network remained the same as the original APM, however instead of taking 4x4 windows of the input map and averaging the output value of each window for a final evaluation measure of the map, the fixed input convolutional neural network (referred henceforth as the POST network)

¹<https://github.com/MasterMilkX/aesthetic-bot-public/tree/main/db>

takes in the entire map as input and outputs a single value as the evaluation score. For this experiment we calculate the percent error from the predicted rating of the original APM network architecture and the POST network architecture against the real pollings from Twitter. The poll data was randomly split into a train/test set (90% / 10%) to train and evaluate both networks as if they were deployed to Twitter. We trained the networks on 2 map different sizes: 6 (the smallest map size) and 10 (the most common map size.)

Maps that were not exactly 6x6 or 10x10 tiles large were not evaluated by either network. Each network was also pretrained with real-user levels with a vote of 100% and randomly generated levels with a vote of 0% to establish a baseline before being fed the real poll data. Table 6.2 shows the results of this experiment averaged over 5 trials. The APM has a slightly lower relative error for map prediction than the POST network for all tilesets for both map sizes. Not only is the original APM model more flexible to accommodate for varying map sizes, but it also has a better prediction rate than a fixed-size CNN model.

Map Size		6		10	
Network		APM	POST	APM	POST
Tileset	Zelda	0.1363	0.1564	0.1061	0.1492
	Pokemon	0.1596	0.2011	0.1157	0.1319
	Among Us	0.2136	0.2272	0.1150	0.1159
	Pacman	0.1309	0.1488	0.1419	0.1616
	Dungeon	0.1327	0.1761	0.1545	0.1657

Table 6.2: Average percent error on the test dataset for the original APM network architecture and the POST network architecture. The APM slightly outperforms the POST network for poll prediction accuracy.

6.4 Discussion

Many interesting trends with the aesthetics of both the user-made maps and the generated maps arose after the Aesthetic Bot went live on Twitter. Both the users and the Aesthetic Bot were capable of producing different styles of aesthetic, thus the system tried to emulate different aesthetic styles in the generated map designs



Figure 6.7: Different groups of user made maps that were submitted to the database. From top to bottom are “level”-like maps, “hacked” maps, and “non-ludic” maps.

– sometimes combining multiple styles at once. Some examples of these aesthetics include minimalism, symmetry, abstraction, and sprite arrangement. Users also made different styles of maps for each tileset which we categorized as the following: *level-like*, maps that could look like standalone levels from their tileset’s games; *hacked*, maps that involved complex structures and patterns to make overlapping sprites or clusters of objects; and *ludic*, maps that took advantage of the tilesets art design to look like images of other objects (e.g. faces) or words (e.g. ‘Yo’ or ‘Hi.’) Figure 6.7 shows examples of the level-like, hacked, and ludic user submitted maps.

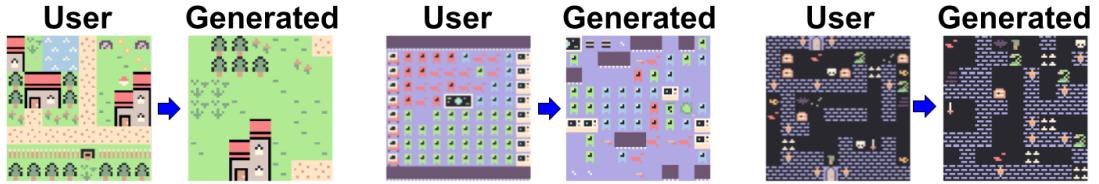


Figure 6.8: Some examples of user maps influencing future map generations. The bot tries to recreate the more complex styles and designs in the user maps.



Figure 6.9: Generated maps for the Pokemon tileset over time. The sprite composition and tile patterns shifts towards different "types" of maps.

The APM attempted to take user-designed map patterns and incorporate them into the generated maps in order to increase aesthetic ratings. Figure 6.8 shows some examples of generated maps that most likely have been influenced by previously evaluated user maps; the Pokemon model tries to recreate the protruding structure of the house, the Among Us model tries to arrange groups of characters by color, and the Dungeon model tries to make maze-like paths through the map. The APM also tried to emulate different aesthetically pleasing pattern trends that shifted over time. This was influenced by the art direction pattern of the user levels submitted to the database in a given period. Figure 6.9 shows an example of generated maps for the Pokemon tileset at different months of the bot's lifetime where the maps change from containing many houses, to having more flowers, to water-focused areas, then path-focused areas, and finally to heavy tree / forest designs.

6.5 Summary

Aesthetic Bot was an automated Twitter account that posts images of small game maps that are either user-made or generated from an evolutionary system.

The bot then prompted users to vote via a poll posted in the image's thread for the most aesthetically pleasing map. This creates a rating system that allows for direct interaction with the bot in a way that is integrated seamlessly into a user's regularly updated Twitter content feed. Upon conclusion of each voting round, the bot learns from the distribution of votes for each map to emulate user preferences for design and visual aesthetic in order to generate maps that would win future vote pairings. Over time, the Aesthetic Bot system learned to compete on a level with user-made maps, with the generated maps occasionally winning the polls for aesthetic preference. This system also demonstrated how different types of aesthetics can be made with such simple design palettes and how the aesthetic AI can learn design trends and patterns from the user-made maps and the polling results over time. The Aesthetic Bot system was successful in retaining user feedback for maps as well as user interaction and map submissions to help the model learn, however learning global user aesthetic preferences still proved to be a challenge for this PCG system.

6.6 Reflection

This was probably my most successful project in the thesis work. Had it not been for Elon Musk's takeover of Twitter and the Twitter API costing money to use, I probably would have continued to use Twitter as a means of user evaluation for generated content. Until the API usage became premium, when the Twitter userbase started to dwindle, users continued to submit new map and vote on the map pairings – about a year after the public launch of the account. I noticed not just with the Aesthetic Bot system that other Twitter Bots also lost their presence on Twitter after the API changes. My feed became boring and I started using Twitter less and less. I have yet to find another medium or platform that can allow for this much accessibility while engaging with such a large userbase.

The users that interacted with the Aesthetic Bot also originally started from

those who were following my Twitter account: other games and AI researchers and game developers. How would have the Aesthetic Bot changed and learned if another demographic group was interacting with it – such as artists, or technology enthusiasts who didn't have a background in AI? Game AI researchers were more inclined to interact with Aesthetic Bot because it was a direct application and contribution to the field of PCG. The anonymous interaction prevented collection of demographic information about the people who made maps or voted on Aesthetic Bot (not including looking at the Twitter handles associated with the map authorships as users could put any account not including their own.)

The motivation for this work was to try to design a model capable of emulating discernment for aesthetic quality in content generation. The model would then be applied to other content generators for grid-world levels. However, based on the user maps submitted, it's clear that there exist different kinds of aesthetic and a single model cannot account or emulate consistently every type of aesthetic. Even during the preliminary experiments, I tried so many different content generation methods that couldn't compare to the output of the evolutionary model; GANs, VAEs, reinforcement learning, and I even tried evolving the maps with a hard-coded fitness value like the Mech-Elites work. The final model pipeline gave the best and most dynamic estimation of "aesthetic" but it's flexibility also made it prone to learning bad patterns. Still, the APMs outperformed making random maps and created arguably more aesthetically pleasing maps than maps generated based on a quantitative metric or preset aesthetic.

This project showed me that making AI systems accessible while also reducing the effort needed from users to interact with it is the best way to design and improve content generation systems. The different level types of level-like, hacked, and ludic demonstrated that users are capable of a range of creative outputs. Collecting data from a large userbase also allows for the most diversity in the content produced and can lead to insights or user trends that a designer may not have predicted on creation of the tool.

6.7 Takeaways

- The demographic of people that interact with a collaborative online system will shape how the system evolves and grows over time and reflect the userbase
- Users have differing preferences and tastes for something that looks “good”
- A single model cannot accurately reflect or address every user’s needs collectively
- The less effort needed on a user’s part for evaluation – the more likely they are to give feedback
- Creating a system for open content submission can create a diversity of content not originally seen upon launch
- A large userbase is needed for the success of a human-AI collaboration system
- Open-ended problems such as aesthetic can be explored more thoroughly via multi-user interaction

Chapter 7

Accessible Text-to-Image Generative Models from Crowd-sourced Data

Around the time of the Aesthetic Bot release, there was a growing trend in generative AI models – models that could create high quality content nearly indistinguishable from user made content that were generated from text-prompt inputs. However, the popular generative models (i.e. DALL-E, Midjourney, StableDiffusion 2) were extremely large, required hundreds of thousands of labeled content for training, and relied on the heavy compute power of GPUs and TPUs to be effective. While these models were versatile, they lacked in practical integration with smaller systems, such as for game development, as they were too large or required premium API calls for usage. Therefore, there was a critical need for accessible, small-scale generative models that could be directly integrated into a game design environment or provide practical utility for creative developers.

This chapter describes the \$5 model – a lightweight text-to-image generative architecture that generates low dimensional images from an encoded text prompt [83]. This model uses the Aesthetic Bot map dataset that contained copious amounts of aesthetically pleasing, user-evaluated data which was submitted to the database.

It also used small-scale images with limited color palettes – similar to that of game sprites. Compared to the extremely large text-to-image generators such as DALL-E, StableDiffusion 2, and Midjourney, the \$5 model provides an alternative, small-scale content generation tool for game developers. This system is capable of generating small pixel-size game content such as levels and sprites without having to use large compute and training data. The model provides quick and aesthetically valuable samples while maintaining a relatively small model size (around 1 million parameters total) and requiring relatively little amount of training data and training time compared to the larger generators. This model was evaluated on three different labeled datasets: 100 PICO-8 sprites – 8x8 pixel images with a maximum of 16 colors; 882 10x10 tile-based maps from the Aesthetic Bot user-made submissions (from the Pokemon tileset); and 663 16x16 pixel down-scaled Apple-made RGB emojis.

7.1 Model Description

The \$5 Model¹ uses a relatively simple neural network architecture and training setup in order to compete with larger generative models while retaining quick training time on less data. The following subsections detail the components of the model.

7.1.1 Architecture

The model takes in as input a embedding vector representation of the text prompt converted using a pre-trained sentence transformer and a random noise vector for additional output variations. The two inputs are concatenated and passed to a dense layer and 2 convolutional layers with batch normalization followed by upsampled residual blocks to output a final image with dimensions of 8x8x16 (for the sprite dataset), 10x10x16 (for the map dataset), or 16x16x3

¹A model so simple, we bet \$5 it wouldn't work

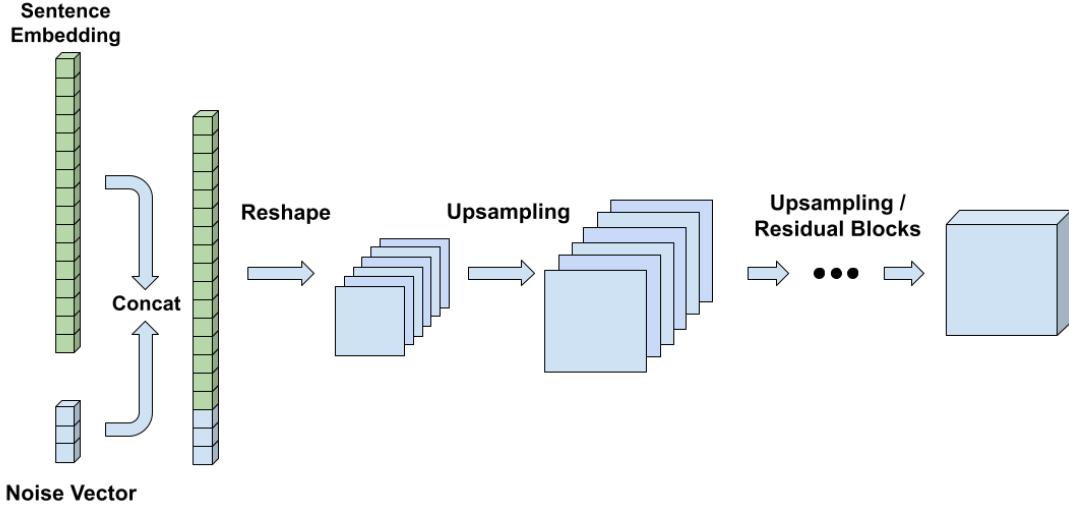


Figure 7.1: Diagram of the \$5 Model architecture

(for the emoji dataset.) This simple concatenation style generation produced the best output compared to other preliminary experiments that involved using other generative methods such as Conditional Instance Normalization [39] and Feature-wise Linear Modulation (FiLM) [97]. Other methods were also attempted for the sprite generator model in particular. These methods included conditional Path of Destruction [111], use of attention networks [132], variational autoencoders [104], and Siamese networks evaluated with CLIP [101]. However, the simple convolutional to batch normalization model architecture still proved to be the fastest and most general across domains. The final model architecture used is shown in Figure 7.1.

7.1.2 Training Data

The model was trained with paired text labels and images in the form of multi-channel integer arrays. Unlike the emoji dataset, which came from the Kaggle website², the Aesthetic Bot dataset and the PICO-8 datasets were not pre-labeled. Labels were manually created for the data via online crowd-sourcing. The largest dataset (Aesthetic Bot maps) contained 882 samples and the smallest containing 100 samples. The small resolutions of the dataset images allowed us to demonstrate the range of output and utility as a small-scale text-prompt generative

²<https://www.kaggle.com/datasets/subinium/emojiimage-dataset>

model. These datasets were also chosen because of the nature of game assets and content that is limited by color space (using 16 colors) or array-based tilesets (with small selection of possible tiles for a map.) Each dataset was labeled with general descriptions, for example, the sprite that looked like Mario from the Super Mario series was labeled as “a man with a mustache dressed in red” instead of given the name “Mario.”

The datasets were expanded by using GPT-4 to generate alternative labels, adding multiplicative Gaussian noise to the random vector concatenation in the model, and using MixUp[141] to interpolate between samples. Only the alternative labels and Gaussian noise showed improvements to training.

7.2 Evaluation

The image output of the \$5 model was passed through a CLIP model to evaluate the measure of text-to-image alignment without human review. Although the model was trained on higher dimensional inputs, it was able to differentiate between the random, fake data input and the structured, real data for all domains except the map dataset. This may be due to the higher pixel dimensionality of the tiles (i.e. trees, flowers, and houses) making up the maps themselves compared to the pixelized images of the sprites and emojis.

Subjectively, the model performed well when given extremely descriptive input prompts. In terms of generalization, the game map dataset performed the best and was able to identify and generate specific areas based on keywords such as ‘rocks’ and ‘island’, the emoji dataset performed the worst in terms of generating novel images and generalizing for seen prompts. The model also performed particularly well on unseen prompts for the map dataset. Figure 7.2 shows examples of the model’s output for text prompts not found in the dataset. The sprite dataset model struggled the most with overfitting, most likely because it had the smallest number of samples in its dataset. Vector arithmetic experiments were also performed (as

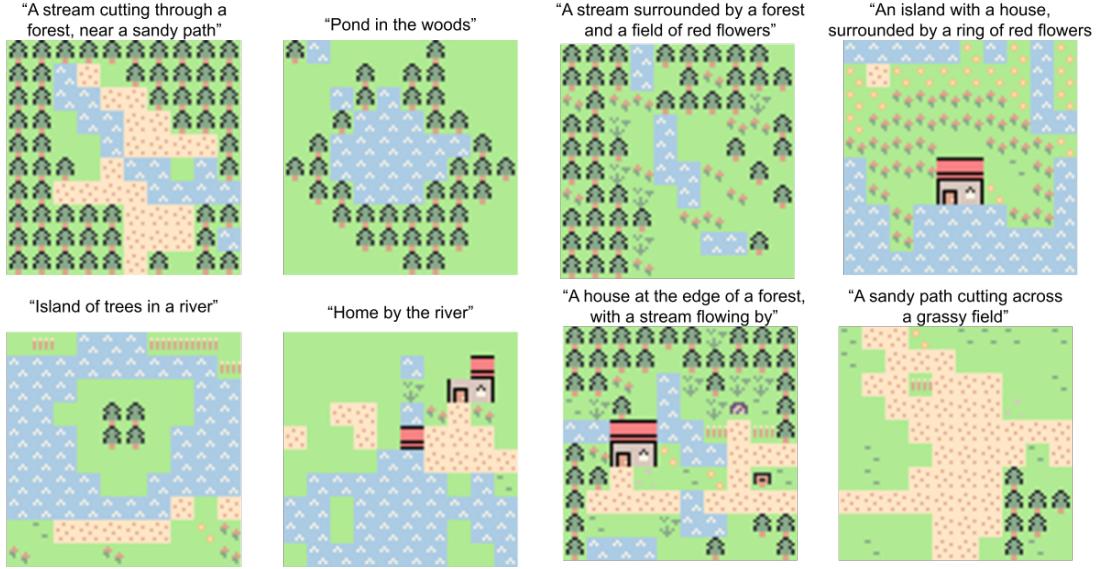


Figure 7.2: Pokemon-tileset maps generated from unseen text prompts using the \$5 model

shown in Figure 7.3) to determine if the model was capable of interpolating between images based on the text encoding of the input prompt and the model was shown to be able to distinguish identifying components and design areas of the sprites as they were associated with the words (i.e. changing the color of the character’s clothes or the body type from duck to a man.)

7.3 Application

To test the effectiveness of the \$5 model in real-world game systems, a demo game was built in the Unity3D engine. Players can generate new sections of a top-down 3D map by moving to the north, south, west, or east of the map and typing a description of the map they would like to see next. Based on the text, the \$5 model would generate the next section of the map that matched the player’s input prompt. The generation for the map is near instantaneous and was fairly accurate. Figure 7.4 shows two screenshots from the demo game showing the text input and the output map screen.

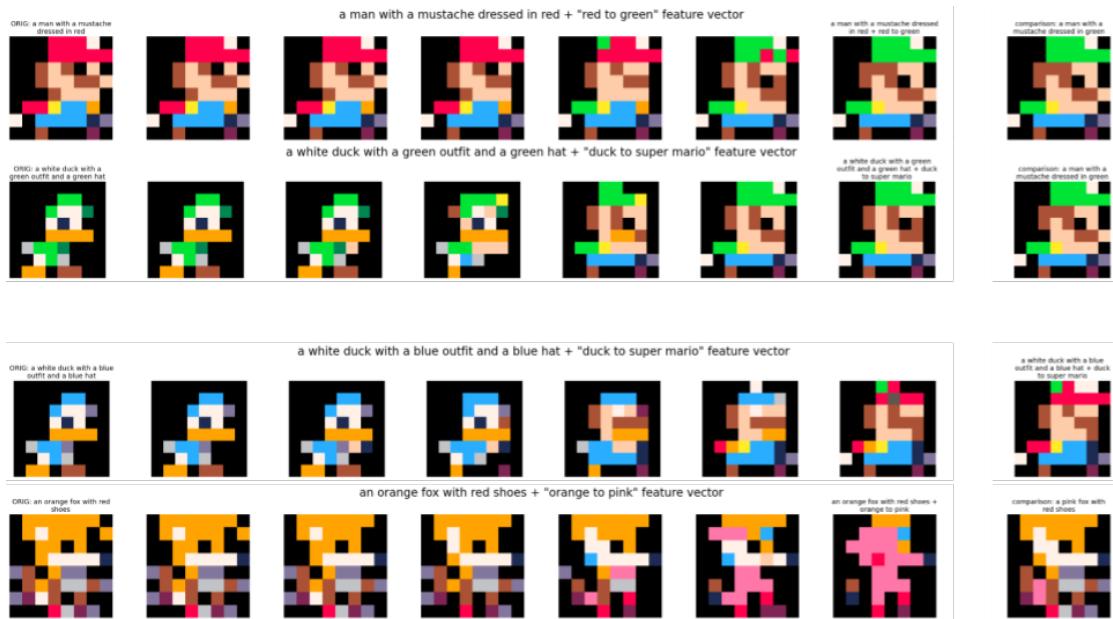


Figure 7.3: Model output vector interpolation between sprites

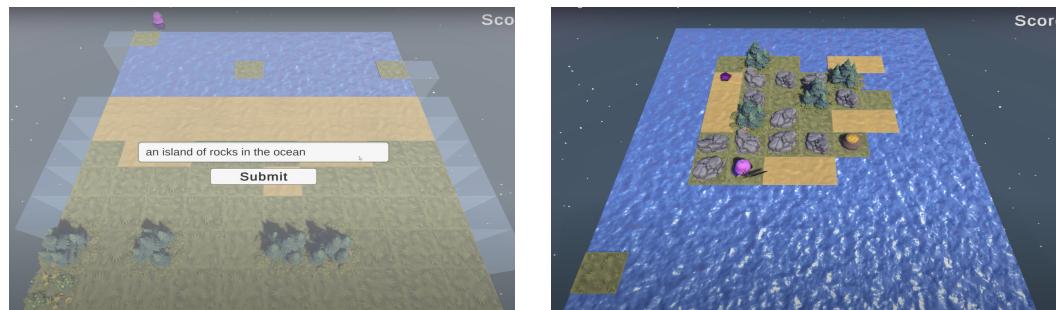


Figure 7.4: Screenshots of the demo game utilizing the \$5 dollar model. On the left, the user inputs the text for the next map to be shown. On the right, the map is generated from the description showing “An island of rocks in the ocean”

7.4 Summary

Despite the visual limitations and lack of proper comparison, the \$5 model proved to be a more accessible, lightweight, and easily trainable text-to-image generator that could be used for game asset specific content generation for multiple domains. This system was intended to act as an alternative to large-scale models that required big-data and large-compute training with the intention of creating small-scale hyper-specific content output. It also made use of the relatively less amount of user-AI generated and crowd-sourced labelling data that was gathered from the previous systems. This system shares a methodology to that of the large scale-generative AI with the text-to-image process in a way that is lightweight, easy-to-train, and accessible for game designers.

7.5 Reflection

The \$5 Model originally started as 2 different projects for text-to-content generation meant for direct use in game development. Each stage of the project was difficult, from gathering the data, training, modifying the model, and writing up the project. The sprites were taken from a Twitter user that recreated pop culture characters using the PICO-8 palette. The emoji dataset was taken from a Kaggle dataset and modified to be smaller and use fewer colors. The Aesthetic Bot maps were the most accessible at the moment and required the least amount of preprocessing. For each of the datasets, we had to retrieve labels either through crowd-sourcing or doing it ourselves.

Finding the right model methodology was also difficult. Originally we tried attention networks, path-of-destruction[111], and finally ended on the \$5 model of a direct text condition to output similar to that of a GAN. We bet \$5 this model wouldn't work because it was so simple. Even trying to add more layers or different augmentations didn't improve the output any drastically. Overall, the system is still very limited and doesn't have the best output. Many of the samples shown in

the paper and in the demo were cherry picked from hundreds of different prompts we tested.

More data would definitely improve the model. Making the model larger, with more layers, more weights, or training for longer could also improve the output. But compared to the amount of data, training, and sizes of the SOA text-to-image generators, this model is more niche and appropriate to what an indie game designer or an AI scientist or student with limited resources could recreate. Considering that these SOA models also take from multiple resources online via scraping – often without the creators permissions – the \$5 model offers an alternative. Designers utilizing this model architecture could train on their own custom dataset and bootstrap the content generation process. The text input format of the model also makes it more accessible to create new content as close as possible to their aesthetic or design needs compared to designing an evolutionary algorithm from scratch.

7.6 Takeaways

- Training small specific models provides more direct utility and application for content generation than large generator models
- Text interfacing for content generation is quicker than traditional PCG methods such as evolution
- Text-to-content generation allows for more controllability from the user
- There is a need for models trained on small amounts of data to bootstrap more content
- There is a need for small and portable content generation models – particularly text-input models
- Making interactable demos for AI systems garners more interest from users

- Voluntary crowd-sourced data labeling and content creation can create diverse training datasets for AI research

Chapter 8

User-Limited Collaboration for Aesthetic Content Evolution

The Baba is Y'all version 2 user study highlighted designers' general preference for manually designing the content themselves over AI-suggested and generated content. The familiarity of the domain also played a key factor in how a user interacted with AI tools. If a user had an specific output in mind for their design, would they find AI suggestions helpful in a way that could speed up their design process? Or would they find the AI suggestions a hindrance and detrimental to their creative intent? Thus, another study was conducted where the roles were reversed – AI would be the sole content designer while users guided the output.

The Minecraft Interactive Evolution system looked to procedurally generate 3D voxel structures reminiscent of the houses of the survival game Minecraft using interactive latent vector evolution [82]. A web application (shown as a screenshot in Figure 8.1) was created to interact with the generator back-end which would evolve and render a population of 3D houses that the user could select from to create new structures. This work was inspired by other mixed-initiative systems that focused on constrained user-AI control and limited user input only to selection of outputs [10, 106].

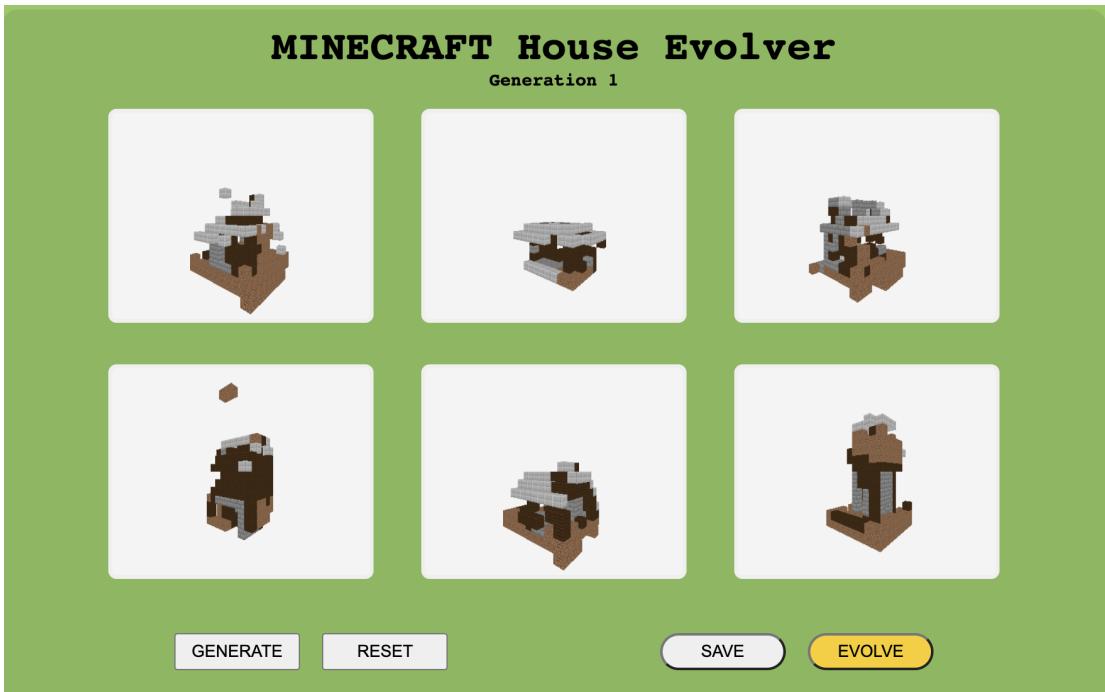


Figure 8.1: Screenshot of the interactive evolutionary Minecraft house generator web application with 6 structure options the user can select to evolve.

8.1 System Description

The Minecraft Interactive House Evolver system had many components to make up the structure generation and evolution process. Unlike other 3D generators, the system was made up of two separate neural networks: one network for generating the structure of the house (the Builder model) and one network for applying textures to the voxel blocks of the house (the Painter model.) A client-side render system was also developed for this system to allow the structures to be rendered outside of the Minecraft environment. These components are described in the following subsections.

8.1.1 Dataset

The models were trained on 128 in-game NPC villages and 219 structures from the online database, CraftAssist [47]. The pre-made Minecraft NPC villages contained a variety of structure designs due to the variance in biomes – which gave us a “typical” structure design for the generator model as they were the most

“house”-looking. Conversely, the CraftAssist structures were player-made submitted structures where users were allotted 30 minutes to build any structure they could – giving a more open-ended creative design to the structures. Both datasets were pre-processed to use only 5 different block types – ‘Air’, ‘Stone’, ‘Dirt’, ‘Wood’, and ‘Slab’ – in order to reduce the block output dimensionality from over 1000 possible block types. The size of the structures was also scaled down to a size of 16^3 cubic squares to reduce the computational load for the generator models. This dataset was augmented by shifting the placement of the structures on the X-Z axis and rotating the structures on the Y axis.

8.1.2 Generator Models

The first model in the house generation process – called the Builder model – used a Wasserstein Generative Adversarial Network (WGAN) architecture similar to that of Wu et. al [137]. This model focused on generating the structure of the house and was trained on binary 3D representations of the training data structures with 0s representing air blocks and 1s representing physical blocks (building the house.)

The second model in the process – called the Painter model – used a variational autoencoder (VAE) model and was trained on the fully textured training data. It took in as input the binary model from the builder and output a 3d structure retaining all if not more of the structure in addition to categorized block textures (painting the house.) The painted version of the structure was what was shown to the user in the web interface. The Painter model contained an unintended action of “fixing” parts of the Builder model’s output structure by replacing some air tiles with blocks.

These models were separated by functionality to improve the aesthetic output of the structures created. Because the aesthetic of the system depended on both the structure of the houses *and* the types and placements of the block textures, having two networks allowed for more explainability with the system’s faults and

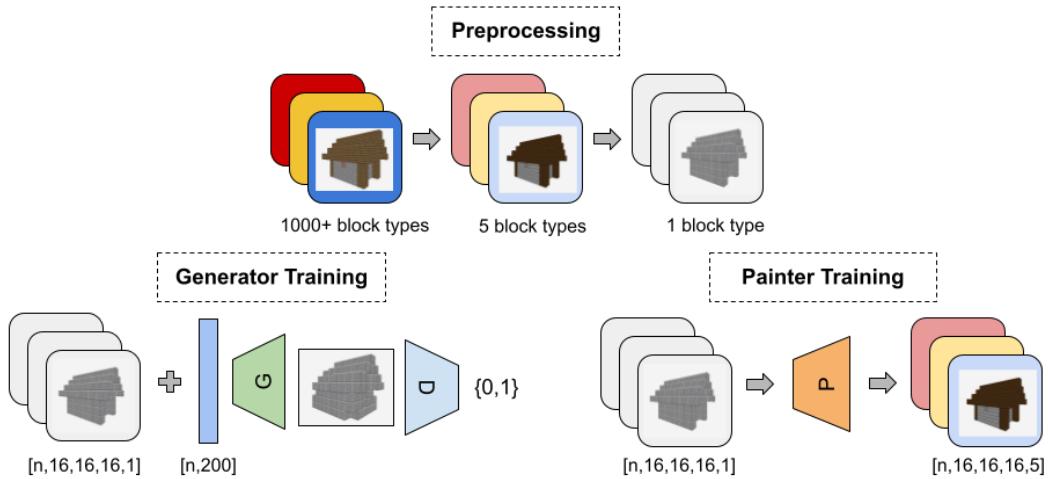


Figure 8.2: A diagram of the preprocessing and training steps for the system

allowed for easier preliminary feedback during training. One aspect of the output could be altered without drastically affecting the other. Figure 8.2 shows a diagram of data pre-processing, and training the generative process for the Builder and Painter models.

8.1.3 Online Evolver

An interactive web interface¹ was designed to allow users to directly select and evolve the rendered Minecraft structures. A custom-made Javascript voxel renderer² was developed using Three.js and NodeJS libraries to render images and GIFs of the Minecraft structures without the need for the Minecraft game engine. This was to increase speed in structure generation and rendering and to provide more accessibility to the system. On the web application, users can select from 6 possible structures, of which the structures' latent vectors will be evolved using a genetic algorithm involving mutation and crossover. These vectors are passed into the generator model to create a new population of Minecraft structures that will be shown to the user – similar to that of Bontrager's deep interactive evolution work [13]. Figure 8.3 show a diagram of the interactive evolution process from web interface to generation.

¹<http://www.minecraft-house-evolver.xyz/>

²<https://github.com/MasterMilkX/VoxWorldJS/>

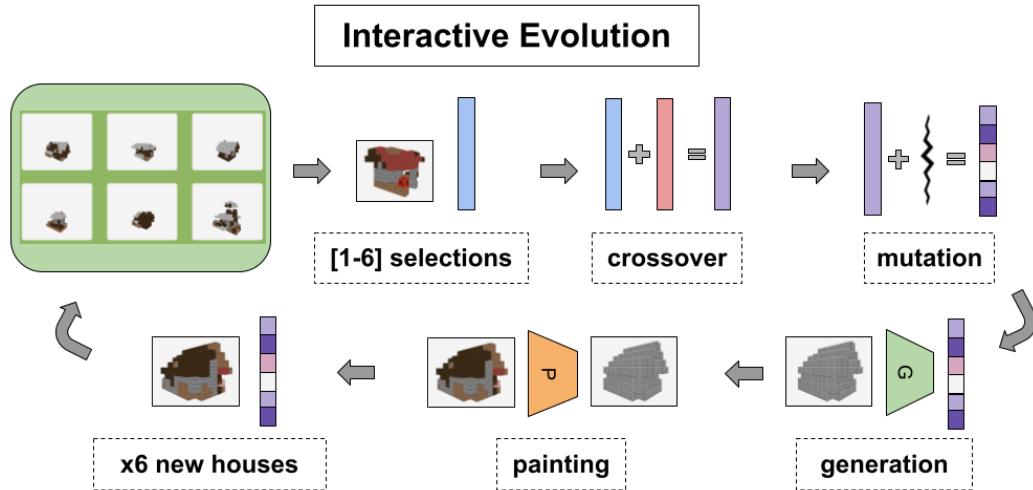


Figure 8.3: A diagram of the evolutionary process for the system

8.2 User Study

A user study was performed to collect information about user design choices, and evaluate the efficacy of the web application and the interactive evolver. This study was modeled after the interactive latent variable evolution experiment performed by Bontrager et. al. [12] and the Minecraft structure generation experiment by Grbic et. al. [48].

A demographic survey was collected first to gather insight into each user's general design style, to gauge familiarity with the Minecraft game and aesthetics, and to see if participants had experience with AI-assisted tools.

Participants were asked to use the system to evolve one of three types of structure - a house with a door and 2 windows, a tower, or two separated houses that were made in the same output sample. A second open-ended task allowed users to also generate any structure of their choosing. For both tasks, participants were asked to report the number of iterations needed to reach a satisfying structure and to paste the raw 3D array data of the structures the generated (a feature included in the web application.)

Finally, participants were asked to respond to 12 Likert scale questions based on their experience with the system. Table 8.1 shows the list of questions provided to users, separated by theme and intention. Q1-Q4 evaluate the usability and

intuitiveness of the IE system. Q5-Q9 evaluate the output of the generator pipeline and gauge user satisfaction with evolution tasks. Q10-12 evaluate long-term satisfaction of the IE system and other AI-assisted design tools.

8.3 Results

8.3.1 Demographic

We received a total of 25 responses over the course of 5 days. The self-reported ages of the users were between 14-35 years of age, with a strong median of users of aged 26. 40% of participants reported preference for goal-oriented tasks (i.e. solving puzzles) over open-ended tasks (i.e. sandbox exploration), and 52% of users said that they enjoyed both tasks equally. Many users (60%) had used an AI-assisted generation tool (i.e. DreamFusion, ChatGPT, Picbreeder) before the experiment and were familiar with online AI generators. We also asked participant to identify their possible personas – inspired by Bartle’s Taxonomy of Player Types[9]. The personas for this study were contextualized for the game Minecraft: Builders - players that enjoy building and decorating; Explorers - players that enjoy exploring the generated game world; Fighters - players that enjoy engaging in combat with NPC or player enemies; and Achievers - players that enjoy achievements, such as beating the final boss. Half of the participants in this subgroup identified most with the ‘Builders’ persona.

8.3.2 House Generation

72% of participants chose to evolve a house with a door and window, 24% of participants chose to design towers, and a single respondent chose to evolve two separate structures. The majority of participants reported successfully achieving the design goals described in their selected prompt with many participants able to reach their goal in under 10 generations (64% for task 1, 72% for task 2). For the open-ended task, many users still chose to evolve a structure similar to the ones

prompted from the previous task (houses and towers.) However, a few participants were extremely creative, either in description or ending structure by trying to make the most “chaotic” structures possible or by giving very thorough descriptions of their structures such as ‘A small house with an outer porch and a roof on top of the porch’ and “A fortress with stone foundation and roof”. Figure 8.4 shows some samples of the houses generated by participants for the open-ended generation task.

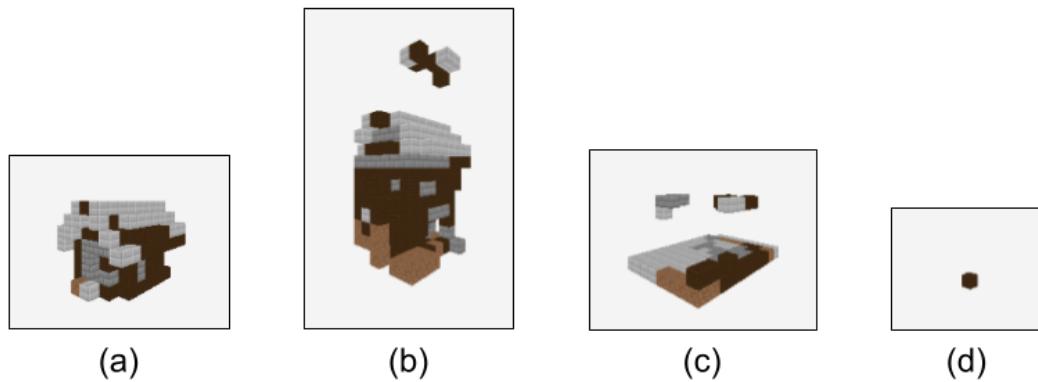


Figure 8.4: Open-ended houses evolved by participants labeled a) ‘a more expensive house’ b) ‘a tower with a cloud idk’ c) ‘A Mario game level’, and d) ‘A single block’

8.3.3 User Experience

Figure 8.5 shows the responses of participants to the Likert scale questionnaire. Participants were in agreement with most statements concerning the aspects of the system. Most participants were ambivalent in satisfaction to the generator’s structure output (Q5), the painter’s texture output (Q6), and how their selections were reflected in the output of the next generation (Q7.) Only 36% of users reported that they would like to use the system again (Q10.), though many agreed that generating the structures using this PCG method was faster than designing a house from scratch in Minecraft (Q8), and that the generated houses were novel and interesting (Q9.) Participants were especially satisfied with the GIF rotation images for their house selections, with 84% reporting ‘Agree’ or ‘Strongly Agree’ to Q3. They also reported especially positive experiences for the long-term AI assisted

#	Statement
Q1	I found the system intuitive and easy to use
Q2	The number of houses shown is satisfactory (not too many and not too few)
Q3	The GIF rotations of the houses were helpful in my selection process
Q4	The time it takes the model to generate new houses is reasonable
Q5	I am satisfied with the structural output of the generated houses
Q6	I am satisfied with the texture designs of the generated houses
Q7	I am satisfied with how my selections were reflected in the output of the next generation
Q8	Evolving the set of 6 different houses is faster than designing and constructing 6 houses from scratch in a 3d environment such as Minecraft
Q9	The generated houses are interesting and novel
Q10	I would use this system again in the future
Q11	I am interested in creating procedurally generated 3d objects.
Q12	I am interested in using more interactive evolution tools like this in the future for building designs either in video games or in other creative mediums.

Table 8.1: Table of statements used in the Likert-scale experience reporting of the survey.

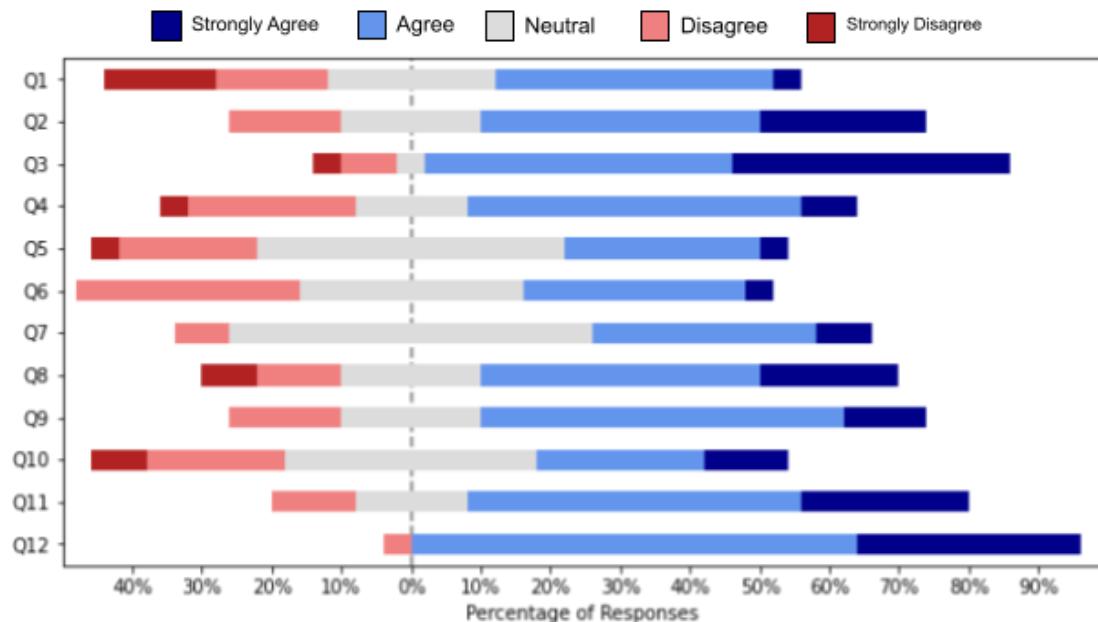


Figure 8.5: Results from the Likert-scale user experience questions.

question. 72% of participants were interested in 'creating procedurally generated 3d objects' (Q11) and almost 100% of participants reported 'Agree' or 'Strongly Agree' to being interested in using more interactive evolution tools for creative tasks (Q12.) Overall, users were enthusiastic to use more AI-assisted tools in the

future for design and construction focused tasks — especially in a 3-dimensional domain.

Participants had the option of providing comments and criticisms they had about the system. As indicated by the ratings of the survey, a few users had difficulty understanding the UI of the system — however this was slightly alleviated with the ‘Help’ menu that demonstrated features of the system with GIFs. Other users mentioned their disappointment with some of the generated outputs — mostly concerning the painter model’s texture placements. Nonetheless, users reported an overall interest in the generation process and would like to see ‘more variant structure types’ and others were interested in ‘[creating] really weird structures.’

8.4 Summary

This study used the 3D voxel game Minecraft as a domain for open-ended aesthetic evaluation and content generation of structures. This system enforced an AI-constrained evolution process in order to examine user experience when faced with creative limitation. While many users agreed that procedurally generating the houses via evolution was a faster process than building the houses from scratch, a majority of participants were ambivalently satisfied with the aesthetic of the system’s structure output overall and how their selections were reflected in the next population. Despite this, a strong majority of the participants were interested in the idea of using AI-assisted design tools and using interactive evolution as a means for creative content generation tasks. This system allowed for rapid 3D content generation based on aesthetic evaluation that could potentially be implemented back into the game environment.

8.5 Reflection

The development timeline for this project was long but overall fulfilling. This project started originally as a class research project for NYU’s AI for Games class.

Timothy Merino, the first author of the paper, proposed a project to automatically generate Minecraft structures using a GAN. The result was an amorphous mound of randomized blocks placed in-game.

We refined the model and tried many different methodologies including changing the size of the GAN layers, using VAEs, updating the training data, and training the model for longer. As a sanity check, we made the training data binary to see if the model could generate structures that resembled “houses”; this in turn led to the development of the double model pipeline for the generator-painter models.

I was also frustrated with having to re-render the generated structures using the EvoCraft library in Minecraft every time also limited the aesthetic evaluation process. As a result, I developed the VoxWorldJS framework for use with the project to quickly render the structures but allow for 3D viewing from any angle. I intended for the framework to be used for future 3D PCG projects and made a markdown usage guide to go with it. So from this experiment, 2 projects emerged that could be considered as a contribution to the academic PCG field. I could see the VoxWorldJS tool being used for more online mixed-initiative content generation projects since it made interfacing with 3D structures so much easier and versatile with the texture selections.

The user study had a surprising number of participants considering the time needed to complete the survey and evolve the houses. It was also insightful in the same way as the Baba is Y'all user study. With the demographic and usage information, I could examine how people interact with a mixed-initiative content creation tool depending on their background and design preferences (i.e. designer vs. player.) Would a designer lean more towards a generative assisting tool vs. a player? Does limiting the user’s creative input frustrate the designer?

The study also allowed me to experiment with accessibility. Since users didn’t have to launch the Minecraft game on their own system in order to generate new houses, I believe this made people more likely to engage with the survey and complete the study. I would’ve liked to have received more responses from users

who had never played Minecraft – but the study was promoted with focus on generating specifically Minecraft houses.

8.6 Takeaways

- Collecting demographic information is key for a user study on interactive systems, content generation systems, and systems with user interfaces
- When given an open-ended task, users will be more incentivized to engage with a content creation tool
- User study questionnaires should be designed to have consistency in positive answers (i.e. framing the question as a positive for all questions vs. having some questions be negative)
- Pre-processing is more required for custom datasets or datasets that are obtained manually
- Interactive evolution is an interesting form of content generation for game assets
- Quick evaluation and accessible interaction is key for aesthetic content systems
- Sometimes an entire framework needs to be designed to make a system more accessible
- Accessibility for designers and developers is also key for continued use of a system (i.e. making guides, READMEs, and wikis for software)
- Adding more automated rendering features (i.e. GIF rotations in 3D space) helps for aesthetic evaluation
- The more questions in a user study, the more possible insights, but could also lead to user fatigue and less user engagement overall

Chapter 9

Recommendation and Generation of Abstract Content

While there is no shortage of game content generators for assets such as levels, sprites, terrain, and other game objects, there is a comparable lack of generators to aid developers with more abstract content generation such as features to add to a game. As the previous user studies highlighted in the Baba is Y'all version 2 and Minecraft Interactive House Evolution systems, AI assisting tools can sometimes hinder a designer's creative intent by overwriting work with generated content. Even when the system offers suggestions or recommendations for content instead of directly replacing the current content, users still prefer to manually design and edit their work. This work looks to study the effect and utility of a system that doesn't create tangible content for developers to add directly into their game. This system recommends and generates conceptual game features intended to aid designers at the brainstorming stage of game development – the earliest step in the creative design process. This system was designed to provide ideas and inspiration for designers before they start to create their game, similar to a way a friend or collaborator would [19].

9.1 System Description

The pipeline for the system is made up of a recommendation model for identifying which games and game features are most thematically similar to the input prompt and a second model for generating new game features based on the thematic similarity of the recommended games. The following subsections describe the components of the system and training process.

9.1.1 Dataset

Summary descriptions and genre tags of over 400 thousand games were retrieved from the Steam Web API ¹ and LaunchBox Games Database ² metadata. The SpaCy³ library’s part-of-speech tokenizer was used to extract nouns from the description text to use as theme-relevant entities (i.e. ‘sword’, ‘gun’, ‘farm’)

Mullich defines “game features” more broadly - which represent descriptive aspects of a game’s design, art, or technical capabilities to explain the overall game experience [86]. These features are typically found in the game’s summary text and use higher-level language to communicate the most interesting aspects of the game without giving away too many gameplay details. As Mullich stated, game features are often found in the game’s descriptive summary text, typically starting with a verb (i.e. ‘jump’, ‘build’, or ‘attack’) and ending with a noun (i.e. ‘platform’, ‘tower’, ‘enemy’.) Part-of-speech grammar expressions defined as “VERB / ARTICLE? / NOUN” was used to find and extract phrases from the description text to retrieve “feature”-like phrases. Figure 9.1 shows a sample of features extracted from the description of the “Castle Crashers” game.

Games that did not contain any features found using the grammar were removed from the training set. This left us with nearly 60k games that contained a list of entity-based theme-relevant nouns, pre-defined labeled genre tags, and grammar-extracted feature phrases to be used as the training data.

¹<https://steamcommunity.com/dev/apiterms>

²<http://gamesdb.launchbox-app.com/Metadata.zip>

³<https://spacy.io/>

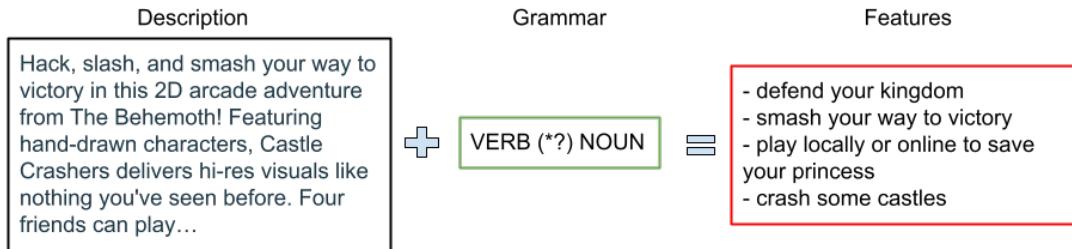


Figure 9.1: Sample game features from the Castle Crashers game description extracted using a part-of-speech grammar

9.1.2 Recommendation Model

An input prompt in this system consists of only a short sentence describing high-level ideas of the user’s game. However, this input description alone could not provide nearly enough information to match it to the games in the training data. Therefore, we created a game recommendation system to find the game or set of games that are closest in theme, gameplay, and semantic relation to the input prompt and augment the data when prompted with more information that the system could use for new feature generation.

Similar to extracting the thematic tags and entities, we used SpaCy to tokenize the user’s input prompt and extract any relevant noun entities and game tags. After that, the recommendation system uses the 400k token, 50-dimension GloVe word embedding model⁴ to encode each noun entity in the input prompt and the training data. This gives us embedding vectors (50 dimensions each) for every noun entity. These vectors are used to calculate a similarity matrix between the input prompt vectors and the vectors of each game in the training data using the cosine-similarity metric. To calculate the final score for each game, we compute the maximum similarity score between the input vectors and the game vectors. Table 9.1 shows an example matrix of the game entities for ‘Nidhogg’ compared to a hypothetical action game about knights.

Finally, these scores are added together using a weighted sum. The weights for the input nouns are calculated using the term frequency inverse-document

⁴<https://nlp.stanford.edu/projects/glove/>

	fighting	jump	opponent	swords	MAX
knights	1.081	0.344	0.473	2.217	2.217
castle	0.679	-0.062	0.219	1.026	1.026
assassins	1.82	0.151	0.918	0.168	1.82
					1.504

Table 9.1: Similarity matrix for the terms ‘knight’, ‘castle’, and ‘assassins’ compared to some game entities from ‘Nidhogg’.

frequency (TF-IDF) [102]. This is intended to exploit the uniqueness of particular words that may be seen less often in the game descriptions. For example, “shooter” would be seen more often across the dataset of games compared to a more niche word such as “alligator” and therefore “shooter” would have a lower TF-IDF score. This score would decrease the likelihood of generating generic features and retrieve more niche features for a unique game concept. The tags and noun entities from this subset of recommended games are added to the input prompt’s extracted noun entities to create an augmented input dataset that can be fed into the generator models. Figure 9.2 demonstrates the input to the recommendation pipeline of this system.

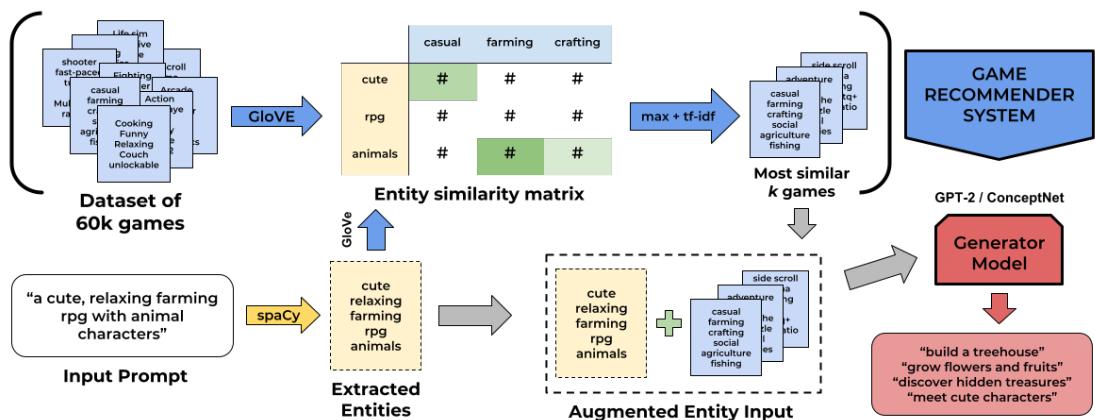


Figure 9.2: Diagram of the conceptual feature recommendation process feeding into the generator model for training

9.1.3 Generator Models

9.1.3.1 GPT-2 Model

For this experiment we used 2 different machine-learning models to generate new game features based on an input prompt. For the first generator - referred to as the Transformer or the GPT-2 model - we fine-tuned a medium GPT-2 model for 5 epochs (due to compute constraints and to avoid overfitting) using the tag set, extracted entities, and description features of the training dataset described in subsection ???. After some preliminary experiments with hyperparameter values and informal evaluation of the generated output, we decided to use the following hyperparameter values for the model during generation: $temperature = 0.95$, $top_k = 100$, $top_p = 0.8$, and $repetition_penalty = 0.95$.

Example 1 shows an example of the formatted training data extracted from the methodology described in subsection 9.1.2 passed to the transformer. For new feature generation, the output from the recommendation system is preprocessed and passed to the model in the same format (a list of tags and entity nouns followed by 'REC FEATURES' to facilitate text generation of new features). The generator often produced a large bullet point list of features of varying length - occasionally repeating some features and phrases. We manually selected 5 features for each prompt from the output of the model to be used in the user study. This manual selection emulates how a user might pick and choose ideal text output from a transformer model recommending them new features - with the user preferring some features that fit their prompt better than others or ignoring repeated generated features.

Algorithm 1 A sample data point (from 'Donut County') from GPT-2 fine-tuning data. Game tags and entities are bolded.

```
3d puzzle story rich cute funny relaxing family-friendly female protagonist stylized comedy great
character feet friend friends game ground hole holes homes physics prizes puzzle raccoon
REC FEATURES:
- Meet cute characters
- Steal their trash
- Throw them in a hole
- Earn idiotic prizes
```

9.1.3.2 ConceptNet Model

The second feature generator model - referred to as the ConceptNet model - is a search-based semantic model that uses the ConceptNet API [117] to generate features. The ConceptNet model uses the list of noun entities from a prompt (extracted using spaCy as well - just like the recommendation system and the Transformer model) to request the semantic information of each word from the ConceptNet API. Unlike the GPT-2 model, which required fine-tuning and training prior to the experiment, this model directly uses the ConceptNet API to generate features.

pirate is capable of...	bear is capable of...	paper is used for...
<ul style="list-style-type: none"> [en] board a ship → [en] sail a ship → [en] fence contraband → [en] find treasure → 	<ul style="list-style-type: none"> [en] fish for salmon → [en] fish with its paws → [en] hunt a rabbit → [en] winter over in his den → 	<ul style="list-style-type: none"> [en] drawing → [en] make paper airplanes → [en] print something on → [en] write a letter →

Figure 9.3: Sample data retrieved from the ‘CapableOf’ and ‘UsedFor’ lists of the 3 words ‘pirate’, ‘bear’, and ‘paper’

To find the features from the ConceptNet data, we find all the semantic connections for our input nouns that are labeled with ‘*CapableOf*’ and ‘*UsedFor*’. We use the output directly from these lists (after stemming the verb to its base form) as they have a similar grammar format (VERB / ARTICLE? / NOUN) to the original training data’s features. Figure 9.3 demonstrates sample nodes returned as “features” for the words ‘pirate’, ‘bear’, and ‘paper’. We manually selected five features from the output of ConceptNet for the user study for the same reason as the GPT-2 model. For this model, none of the additional game tags suggested by the recommendation system were used as this exponentially increased the execution time of the generator and the number of returned features from the user’s prompt were more than sufficient for the experiment. It was also possible to extract related words from each noun’s ConceptNet data object to extract their “feature” relations by stepping through the network. However, this process also greatly increased the execution time to return features from the model - regardless

Game #	Prompt	Human Features	GPT-2 Features	ConceptNet Features
1	<i>a class based multiplayer online shooter with capture the flag, death match, and deliver the payload</i>	shoot gun to heal allies double jump in the air drink energy drinks throw suspicious jars backstab enemy targets attack enemy turrets buy power ups type to communicate with enemy team coordinate tactics with allies protect allied mobs	use a bomb help them survive the assault earn experience change the route use an offline password fight against each other in an arena advance through the game control enemies start with a single character become the champion	secure doors pick a leader construct buildings blow something up kill someone secure valuables hold a large conference find the treasure pick a leader plan a road trip
2	<i>a 5v5 game where you protect your base while destroying enemy bases using an array of different abilities and items</i>			
3	<i>an open-world exploration game in a post-apocalyptic fantasy world where you can climb anything and destroy everything</i>	explore the wilds battle towering enemies gather ingredients bundle up with warmer clothes think quickly and develop the right strategies meet new animal friends explore the ruined castle identify new flora and fauna discover new alchemy recipes find the antidote to your curse	Travel to a variety of forests and field make weapons learn new combat become the new king create new outfits grow your own plants make your own unique potions sell your creations explore a huge world use your imagination	shred into pieces of confetti provide comfort prevent starvation house a king find shells on a shoreline sometimes buy and sell things discover the world make wood pulp reach towards the sky make foods
4	<i>an RPG about a princess who collects swords and flowers to turn into potions and is secretly a frog</i>			
5	<i>a collaborative cooking game where you make and sell onigiri in your college dorm room</i>	make onigiri on the weekends pay off your tuition decorate your dorm room buy new meats and veggies hire friends and roommates part-time hack the corps suits	Sharpen your knives communicate and coordinate NPC actions engage in hard boiled head become a master chef battle your crew	cook a gourmet meal cut things cure a headache cry when cutting an onion furnish your home
6	<i>a retro-futuristic cyberpunk skateboarding game where you hack corporations, the robot police, and the street gangs</i>	vandalize properties with graffiti find secret hideouts skate the streets of the city upgrade your bioware tech	experience the iconic grind master the deep combo system choose their personality complete secret missions Show off your style	win a war scare people play science fiction games organize data give money to each other

Table 9.2: Table of hand-authored and model-generated features for each game prompt.

of whether the original prompt entities or the augmented prompt entities were used. Thus we only extracted the first level of data from the prompt’s entities exclusively.

9.2 Experiment

A user study was conducted to evaluate the generative quality of the system when given a text prompt describing a hypothetical game premise. The first half of the prompts described existing games with common genres and themes found in triple-A titles: “Team Fortress 2” - an FPS multiplayer game, “DOTA 2” - a multiplayer online battle arena game, and “The Legend of Zelda: Breath of the Wild” - a single-player open-world action game. The second half of the prompts described games that had not been made and with less commonly found genres and more niche themes: “Alchemy Princess” - a role-playing game with crafting mechanics and magic systems, “Onigiri Dorm” - an economy-based cooking game set in a college dorm room, and “Hoverdeck 209X” - a cyberpunk, action skateboarding game. For both the real and fake games, we hand-authored 5 game features to describe the games. Naturally, the fake game prompts were hand-authored since they did not exist yet. However, despite human-authored features

existing for the real games (extracted from the game’s summary description as described in subsection 9.1.1,) the GPT-2 transformer model was fine-tuned on these same features and was therefore likely to generate the exact same feature set when prompted. As such, a new set of features had to be created for the human feature set for each of the real game prompts.

The two generator models described in sub-subsections 9.1.3.1 and 9.1.3.2 took as input and processed the single sentence description prompts for each game (both real and fake) to generate new game features to create sets of 5 features each. Table 9.2 shows the set of features selected for the experiment for each prompt: the human authored features and the manually selected features from the fine-tuned GPT-2 transformer model and the ConceptNet model.

For this small user study, participants voted on which set of features of the 3 groups they preferred when provided the one-sentence game description prompts. For each prompt, the 3 sets were anonymized and randomly given the label A, B, or C. A Google Form poll was created for each game prompt asking participants ‘Which set of game features do you prefer?’ Each of the 6 prompts were separated into their own form so as to not cause user fatigue with participants. The authorship of the features was not disclosed to participants and they were not told whether the features they were shown were artificially generated. Figure 9.4 shows the image shown to users on the form for prompt #5.

Participants were also given the option of submitting their own suggestions for new features not listed in any of the sets. This was to potentially gain more information about how a human might offer game feature suggestions to a game designer if they were under the impression that they were trying to brainstorm ideas for a new games. This information could be used to train the generators to give more natural and “human-like” suggestions that users would find beneficial for their own game design process. The polls were circulated both online via Twitter and Discord and physically as small printout flyers in the lobby of a classroom floor for game design students.

Game Description

 a collaborative cooking game where you make and sell onigiri in your college dorm room

Which set of game features do you prefer? *

A	B	C
<ul style="list-style-type: none">- make onigiri on the weekends- pay off your tuition- decorate your dorm room- buy new meats and veggies- hire friends and roommates part-time	<ul style="list-style-type: none">- cook a gourmet meal- cut things- cure a headache- cry when cutting an onion- furnish your home	<ul style="list-style-type: none">- Sharpen your knives- communicate and coordinate NPC actions- engage in hard boiled head- become a master chef- battle your crew

A

B

C

Figure 9.4: A screenshot of the Google Form shown to users for prompt #5. Each feature set is anonymized with A, B, or C.

Unlike the previous studies with the Baba is Y'all and Minecraft House Evolver projects, no demographic information was collected for this informal study (i.e. name, age, email, whether they were game designers, or how many game jams they had participated in.) Only the votes for the set of features (values of A, B, or C) and any additional game feature suggestions the users volunteered to suggest. After answering one poll (any of the 6 provided) participants were redirected to a central "hub" website containing all 6 of the prompt polls and were given the option to submit a response to any of the other 5 polls. Since no identifying information was collected, the number of unique users that participated in the study cannot be determined. Therefore, following results are under the assumption that each vote is completely independent from each other and the other polls.

9.3 Results

Over 140 responses was collected across the 6 polls - with an average of 23.6 responses per poll. Figure 9.5 shows the voting results of each set of features based on the game prompts. The human-authored feature sets had a total majority of votes across all 6 polls with 52% of total votes. For the real games prompts, the number of votes is nearly equal for the transformer and the human-authored features; with the GPT-2 transformer model's features having a slight majority of votes (28) over the human-authored features (27) across the 3 games. Conversely, with the fake games, the human features had the strong majority of votes (47) over the transformer (15.) The ConceptNet model underperformed in polls except for prompt #1 - outvoting the transformer model.

This stark difference in votes could be because of the frequency of these particular game genres and common features found in the massive amount of real game data the transformer model was trained on. For game #3, for example, the transformer model could have generated features similar to those found in other open-world fantasy games, such as 'Elden Ring' and 'The Witcher' in addition to

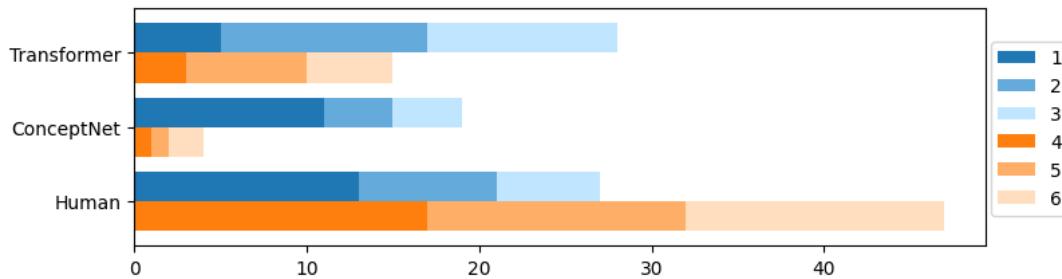


Figure 9.5: Results from the 6 voting polls - blue segments for the real games and orange segments for the fake games.

'Breath of Wild.' The hand-authored features for the real games could be considered "less fun" compared to the original features of these games that the GPT-2 model trained on. Conversely, the fake prompts were more niche in theme and did not exist conceptually in the training dataset, therefore the artificial models may have struggled to generate features that were related to these genres and themes. For both groups, the ConceptNet model's features consistently had the least amount of votes except for game #1's prompt. Compared to GPT-2's suggestions, the ConceptNet model had more "non-sequitur" and detailed suggestions which may have pushed away the participants to select other ones.

While there were relatively few added human suggestions, some participants (who voted for the human set) did note that they liked certain features from the AI-generated sets - and even liked 2 of the sets equally. One participant suggested aesthetic visual designs for the game and specific game elements that could be added. A few participants suggested to 'add sex' as a feature in the optional form input for games #5 and #6 which was - coincidentally - also a feature suggested by the ConceptNet model for those games but not included in the final set. We found that the format of some of the user input game feature suggestions matched the same VERB / ARTICLE? / NOUN grammar format we used to extract the training features from the summary descriptions, while other user input suggestions erred more on the conversational side - often providing their reasoning behind why a certain feature could improve the gameplay. This conversational format could be beneficial to add to a recommendation system to give the AI a sense of

explainability to the user requesting game feature suggestions.

9.3.1 Comparison to ChatGPT

An argument can be made to that models with more parameters and versatility in output – such as the ChatGPT and the GPT-4 model – would outperform the fine-tuned GPT-2 model for game feature recommendation. However, the size of the models, compute costs, costs for API requests, and challenge to fine tune these large models make these impractical to incorporate into AI-assisted tools for game designers. To further argue on the usage of smaller fine-tuned GPT-2 models over the general ChatGPT models - we provide excerpts from a ChatGPT interaction requesting game features in the same format given in the user study experiments. We used the input prompt “Can you recommend me 5 different potential game features that are 4-7 words each in bullet point format for the following one sentence game description:” followed by the one-sentence prompt used in the experiments. Table 9.3 shows the output responses given by the ChatGPT interaction.

While it is impressive how well the model can adapt and stay on theme with the prompt for its output generations, we find that the features given by ChatGPT are overall subpar in comparison. Many of the features suggested are too generic or non-descriptive (i.e. ‘Intense team-based combat’ and ‘Fast-paced gameplay with varied objectives’ in prompt #1,) simply reiterate on the description of the game given in the prompt (i.e. ‘Sword collecting and upgrading system’ for prompt #4 and ‘Dynamic environmental destruction and interaction’ for prompt #2) or repeat features across different prompts (i.e. ‘Competitive matchmaking and ranking system’ in both prompts #1 and #2.) Very little is offered from the transformer model in terms of specific and innovative gameplay mechanics and elements. While prompt engineering and refinement could be performed to try to extract more niche and varied feature suggestions, the simplest, and less computationally and financially expensive, GPT-2 model is still preferred for its practicality and contextual application to the problem at hand.

Game #	ChatGPT response
1	Specialized classes with unique abilities Fast-paced gameplay with varied objectives Intense team-based combat Customizable weapons and loadouts Competitive matchmaking and ranking system
2	Dynamic hero abilities and item builds Base building and defense mechanics Resource management and strategy Intense team-based battles Competitive ranking and matchmaking system
3	Dynamic environmental destruction and interaction Fantasy creatures and unique enemies Character customization and progression Immersive exploration and discovery Non-linear story and quest system
4	Sword collecting and upgrading system Alchemy-based potion crafting mechanics Frog transformation with unique abilities Immersive fantasy world and characters Non-linear story with multiple endings
5	Cooperative gameplay with friends Customizable dorm room and kitchen Onigiri recipe book with unique ingredients Customer feedback and rating system Time and resource management mechanics
6	Dynamic skateboarding mechanics and stunts Hacking mini-games and puzzles Open-world cyberpunk city to explore Factions and reputation system Story-driven campaign with multiple endings

Table 9.3: Table of ChatGPT responses given the same prompts used in the user study

9.4 Summary

This chapter discussed the preliminary work of a high-level game feature recommendation system and generator. Based on the results of the small, informal user study, and given more time fine-tuning and more feature data - possibly extracted from real game reviews along with their summary texts - it could be argued that the transformer could compete on the same level as the human-suggested features and outperforms more commonly used AI-based text generators such as ChatGPT. With more training and fine-tuning, this system could be incorporated into a mixed-initiative game design tool to help designers brainstorm game features and provide game content at a conceptual stage rather than as a directly influential asset such as level design or sprite design.

9.5 Reflection

Originally, this project was the start of a much larger interactive chatbot system called BMO. BMO was a robot side-character from my favorite TV show, Adventure Time, and he often made new games for the main characters Finn and Jake to play. But he was also a good friend to them. I was inspired to make a system depicting the relationship I had with my friends while I designed video games. They didn't have much game development experience, but they helped me come up with new game feature ideas, debug my code, and playtest my games. This system, BMO, was designed with this intention in mind – to help a game designer in a similar to my friends. Recommending game features was the first step of this large project. But around the end of this first project's development, ChatGPT became very prominent, versatile, and accessible, and it was clear that I wouldn't be able to develop a chatbot that was capable of generating content of similar caliber.

Still, the outputting game feature recommendations and suggestions of this preliminary BMO system was better than the more generic feature suggestions of

ChatGPT. The search process for finding this specific data and parsing it for text patterns also gave it more specificity and color than what ChatGPT could output for the same game concept. If the text pattern parsing had been tweaked more, or if more text could be extracted from the summary, then maybe the generator would output better feature recommendations for the niche, made-up games. Given the time and resources, I would have extended this work and conducted another study tasking indie game designers to implement 1 of the 3 groups of game features (human, GPT-2, and the ConceptNet) into a small game and asking participants to anonymously evaluate their preference of the games. However, this work was done in a similar fashion later with ChatGPT generated features as described in Chapter 10.

I'm also still very interested in an integrated chatbot system for game development that isn't as heavy or as generic as ChatGPT. I'd like to make a system that could be implemented directly into a small game engine and help with more specific tasks for game development such as level design, sprite creation, bug fixing, and playtesting – tasks that have already been solved in the game AI research community, but not all at once by a single, consistent, and fully integrated system. It'd be like going onto Discord to ask your friends in a group chat for help designing all of these things for your game. My experiences may be entirely personal, but I believe there exist game designers who could benefit from a personalized content creation tool that addresses many aspects of game design.

9.6 Takeaways

- Regular user feedback at multiple development stages is needed to evaluate the quality of content generation
- Sometimes more out-of-the-box and weird content is preferred by users; particularly for generating abstract content such as ideas
- Demographic and background information on game designers and the type

of games they design can influence their decisions and picks on the type of content they prefer

- Simple techniques for data processing (i.e. TF-IDF, regex matching) can be very effective and often quicker than machine learning processes – particularly in natural language processing
- Implementing a specific pattern can possibly exclude more niche and interesting artefacts from a dataset that will negatively influence the outputting content generation
- Data augmentation provides more interesting and versatile content output
- ChatGPT is powerful, but not good at everything
- Break up user studies into sections to help with user fatigue
- Short user studies will garner more participants, but have less contextual information

Chapter 10

Large Language Models as Game Design Partners

The abstract content recommendations from the previous work gave many insights into how a user might turn to a generative model for advice on conceptual game elements. While the large-language model ChatGPT had limitations and lack of specificity to the feature recommendations for niche games premises, it still had the potential to offer more content variability for other aspects such as game feel or more specific but behind the scenes content such as code. A novice user that needs assistance during multiple stages of the game design process including brainstorming, programming feature implementations, or adding elements of game feel [123] could benefit more from a large language model’s assistance. This chapter explores a study on how large language models (LLMs) can act as creative collaborators for multiple areas of game design as applied to three game genres: space shooter games, platformer games, and roguelike games. These games were blindly evaluated based on quality by players in a formal user study.

10.1 Interaction Methodology

To test the suggestive game design capabilities of ChatGPT, a series of 2D games for three game genres —arcade space shooter, platformer, and turn-based

roguelike—were designed in the Unity game engine. These genres were chosen to test ChatGPT’s capabilities for general game domain knowledge and how well it could apply context-relevant design suggestions. Each genre had three games: a baseline game ; a purely human-designed game; and a game made from ChatGPT suggestions (referred to as ‘ChatGPT-designed game’.) The games were designed with minimalist graphics, where players and NPCs were made up of simple shapes and colors, to prevent any visual bias across genres.

The AI trials mimicked a situation where a developer had a basic gameplay loop in mind but little game design experience. This designer would then turn to ChatGPT for guidance on how to implement their idea in addition to receiving recommendations on mechanics or abilities that could be added to improve gameplay. For this study, the authors performed the role of the novice designers and directly implemented ChatGPT’s responses (without regard to actual skill level or personal opinion of the output) into the games. ChatGPT was prompted for high level subjective recommendations (e.g. “What abilities should I add to my game?”,) low level code implementations (e.g. “Can you provide code and instructions on how to apply this in Unity?”,) and suggestions for features and improvements to the game feel of each genre.

In addition, improvements to game feel and general bug fixes were limited to suggestions exclusively from ChatGPT. A number of limitations were placed on prompting ChatGPT to ensure the prompter’s bias or influence was kept to a minimum. The prompter was only allowed to request code implementation for up to 5 different features for each game. If a bug arose from any of the generated code, up to 3 attempts could be made to ask the program how to fix the error, after which its failure to do so would be noted. The prompter could request 1 general explanation of the provided code and could ask once which game feel elements should be associated with the feature. Figure 10.1 shows a diagram example of prompting ChatGPT for implementing a freeze ability in the space shooter genre game.

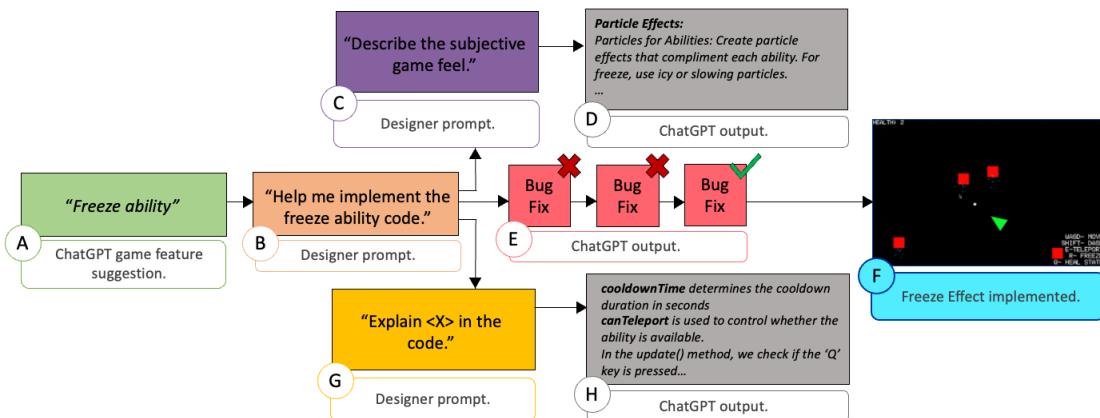


Figure 10.1: Example of the freeze ability being implemented for the space shooter genre game with guidance from ChatGPT. The freeze ability is (A) first suggested by ChatGPT from an initial prompt. (B) The user then asks ChatGPT for help with implementing the code in Unity engine in C#, (E) the output code from ChatGPT is attempted to be implemented 3 times by the user until it is successfully integrated into the game without error. The user also asks for (G) explanations about the generated code and (C) further suggestions to add game feel

The following subsections describe the distinguishing features and creative development process from implementing the 9 games.

10.2 Space Shooter Design

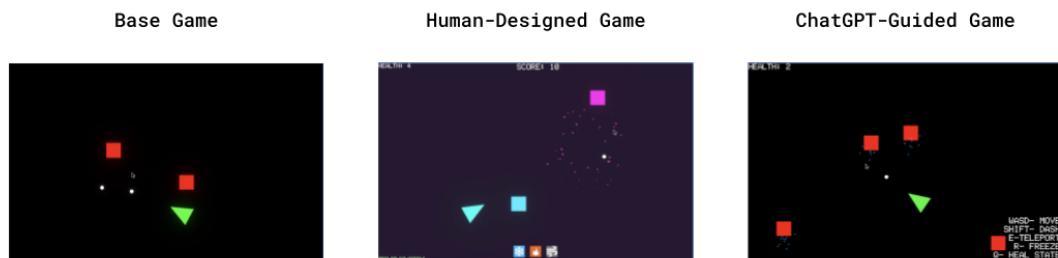


Figure 10.2: Screen-captures of the 3 shooter games created for the experiment.

The prompt for the first game was the following: *"Create a fast paced, top-down shooter with 3 special abilities. Each ability should have a short cooldown. The goal is to survive for as long as possible while enemies randomly spawn into the scene and chase the player"*. Figure 10.2 shows screenshots from the base game, human-designed game, and ChatGPT designed game for the space shooter genre.

10.2.1 Base Game

The base space shooter game included standard top-down movement as well as a short ranged omni-directional dash on a brief cooldown. The player would be oriented towards the mouse position on screen, and a button press would fire a bullet in the same direction. Enemies would spawn into the edge of the scene at set intervals, from any one of 9 evenly spaced spawn points. Enemies would follow the player's position and were destroyed when they either collided with the player, in which case the player's health would drop, or shot by a bullet. The game featured a black background and simple geometric shapes for the player, bullets, and enemies, with no elements of game feel (i.e. particle effects.) The player's health was shown as a text in the top of the screen.

10.2.2 Human-Designed Game

The human-designed space shooter game added elements of game feel on top of the base game's mechanics. Screen shake was added for when an enemy or player was hit. An additional "glitch" visual effect for when the player was hit – changing the colors of the player and enemies. Sound effects for shooting enemies and enemies being destroyed were also added along with an explosion particle effect. Additional elemental-themed mechanics (with game feel) were implemented in the game for added game complexity:

- **Freeze Effect:** Cause enemies to turn light blue in color and stop moving.
- **Fire Attack:** Created an orange arc around the player to attack enemies
- **Wind Attack:** Push enemies away from the player with white lines moving outward

Cooldowns and visual UI indicators for the new attacks were incorporated for better cohesion and readability.

10.2.3 ChatGPT-Designed Game

Initially, ChatGPT was asked to recreate the base game from scratch to test whether it was capable of understanding designer intent from the initial stages of game development. The model was able to accurately rebuild most the game from scratch, with detailed descriptions of how to add objects, code, and components such as line renders in Unity. While the process was fairly harmonious, proper execution required a more in-depth understanding of Unity and the general game making process.

ChatGPT was then asked to suggest a list of potential abilities for a "top-down, spaceship shooter game". It provided a list of 20 options, including:

- **Teleportation:** Instantly jump to a different location on the screen, useful for escaping dangerous situations or repositioning strategically.
- **Nanobot Repair:** Activate a healing ability that slowly restores your ship's health over time.
- **Time Manipulation:** Slow down or temporarily freeze time to dodge bullets, aim precisely, or gain a tactical advantage.

When ChatGPT was asked for code, a number of issues and bugs arose in the process. For instance, teleportation did not achieve the desired effect that ChatGPT itself outlined in its description, as it would often teleport the player directly to an enemy or to the same position they already were. A later issue with the same ability would cause the player to disappear every time it was activated. Despite multiple attempts to solve this problem, ChatGPT was not able to resolve the issue, and the problem was fixed manually by recognizing the player's location on the z-axis was being changed during teleportation.

The model struggled most with incorporating game feel elements for the abilities. It often gave vague instructions for adding particle and sound effects. Basic suggestions such as having the teleport look like "a burst of stars or particles that

appear and disappear quickly” were provided, but without detailed instructions on how to achieve them. As a result, the final game contained a significantly lower count of additional game feel elements than the human-designed counterpart. The ChatGPT log for the space shooter genre game can be accessed here¹.

10.3 Platformer Design

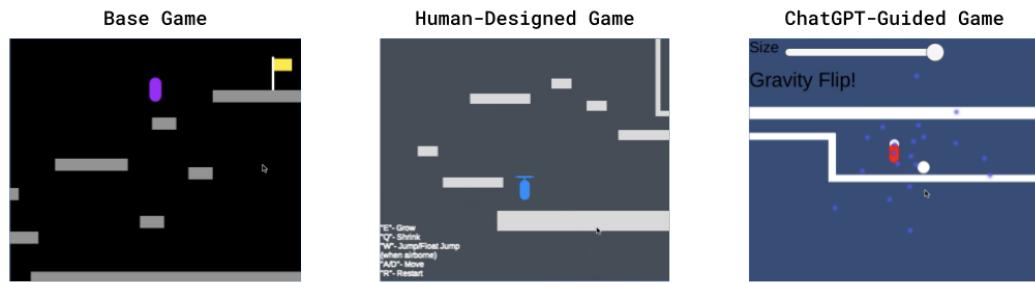


Figure 10.3: Screen-captures of the 3 platformer games created for the experiment.

The prompt for the second game was “*Create a simple platformer where you need to reach the flag to win. Develop 3 transformative mechanics to the player which change their platforming. Design a short level that takes these abilities into account*”. Figure 10.3 shows screenshots from the base game, human-designed game, and ChatGPT designed game for the platformer genre.

10.3.1 Base Game

The base platformer game contains minimalist controls and visual design with only horizontal movement and a jumping mechanic to allow the player to traverse floating platforms found in the level. The player must reach the flag object found in the level to complete the game. The level is designed similar to that of a pyramid, where platforms are placed vertically and horizontally throughout the space. If the player falls off the level, they are respawned to the original starting point. Like the base space shooter game, the base platformer game contains no particle effects,

¹<https://chat.openai.com/share/f807763a-6a25-4fd3-85de-adb5f1de8159>

sound effects, or any other elements of noticeable game feel. The only UI is a transparent screen that tells the player when they have won the game.

10.3.2 Human-Designed Game

Like the space shooter, the design process began with implementing a layer of general game feel elements to the base game. “*Coyote-time*” jumping was added to make the platformer feel more responsive to the player [115]. Two transforming mechanics were added to the game:

- **Shrinking:** The player can traverse thin crevices, but suffer from a much smaller jump
- **Growing:** The player is unable to pass through certain environments, but is able to jump proportionally higher and farther
- **Float:** When in midair, players extend their jump and descend at a slower rate

Game feel elements such as particle effects and physical property changes in the player sprite (such as a hang-glider at the top of the character for the float ability) were also implemented.

10.3.3 ChatGPT-Designed Game

While ChatGPT was once again able to recreate the base game eventually, its poor understanding of the game’s context became an even bigger issue for the platformer. It would often provide code for a 3D project- despite multiple reiterations that the game was in 2D- and was incapable of producing a reliable method of following the player with the camera. It also listed a time manipulation ability which would pause all enemies on screen for a brief period of time as one of its 3 suggestions for “transformative mechanics”. However, the game not featuring any enemies. As a result, only 2 abilities were implemented in this final game.

ChatGPT also recommended implementation of transformative size mechanics such as growing and shrinking. A gravity flip mechanic was also recommended, however the code provided by the model only worked to flip the player in one direction. Unique to the platformer game, ChatGPT also gave suggestions for level design given the transformative mechanics at play, such as advising that the game have a section which was too narrow for the player to fit through if they were in the largest transformation size state. Game feel suggestions were once again minimal, apart from a UI element which displayed the player's size through a slider at the top left of the screen. The ChatGPT log for the platformer genre game can be accessed here².

10.4 Roguelike Design



Figure 10.4: Screen-captures of the 3 roguelike games created for the experiment.

The roguelike game had the following prompt: *"Create a turn based stealth roguelike where the player must collect a key and then head to an exit to escape, all while avoiding patrolling enemies. The player can collect or recharge up to 3 abilities, but they must be stealth related. The player can also engage in direct combat with enemies"*. Figure 10.4 shows screenshots from the base game, human-designed game, and ChatGPT designed game for the roguelike genre.

²<https://chat.openai.com/share/e1813d47-d568-475e-9f57-5975cc42fce0>

10.4.1 Base Game

The base roguelike game uses mechanic designs from classic dungeon-crawler games. Rooms of a random size are procedurally generated and connected with corridors using a binary-space partition algorithm in a fixed-grid-world space. The player movement and combat are turn-based, where for each step, the player enacts the next turn of the game. The player can move in any directional space (up,down,left,right) so long as there is a floor tile. The player can also skip a turn and remain on the same tile. Enemies are randomly placed in the rooms, move in a random direction every 3 turns, and follow the player when they are within 5 spaces of them. To win the game, the player must find a randomly placed key, survive combat from enemies, and then use the key to unlock a randomly placed door. If the player loses too many health points, they lose the game.

10.4.2 Human-Designed Game

The roguelike separated itself from the previous two genres largely due to its turn-based nature, which impacted the approach to adding features the game. Three stealth-focused mechanics with time-based cooldowns were implemented in the game

- **Invisibility:** The player would be invisible for a number of turns and invulnerable to enemies. Includes a vanishing cloud particle effect and sound effect and the player sprite turns translucent.
- **Smoke bomb:** A cloud particle effect is shown and all enemies in a certain radius of the player stop moving
- **Shuriken:** A projectile attack to attack multiple enemies in a line (shown as a dashed line for aiming)

Blood splatter particle effects were added for when enemies were killed and cooldown timers were added to the user interface. To incentivize combat, killing enemies shortened all the cooldowns of abilities times by a sizeable amount.

10.4.3 ChatGPT-Designed Game

Unlike the other 2 genres of games, ChatGPT was unable to recreate the original design of the base game for the roguelike genre. It could not understand the ‘turn-based’ nature of the game and apply it to a real-time Unity game engine context. The LLM was also unable to correctly implement the binary-space partition algorithm for the rooms and the turn-based enemy AI movement. This could be due to the lack of context for the input prompts sent to the LLM, the Unity game engine, or the miscommunication between user intent and what ChatGPT was capable of creating. Therefore, the original base game was used as a starting point for implementing the feature suggestions of the LLM.

Like the human-designed game, ChatGPT suggested to implement an invisibility feature. It also suggested to implement an energy system, where players would need a certain amount of energy points in order to engage in combat and turn invisible. This energy could be restored by waiting turns. In addition to the random chance to regain health from defeating an enemy, ChatGPT suggested that the player should have a chance to increase their energy restoration rate, decrease their energy consumption, or increase the invisibility time duration as an alternate reward.

Finally, ChatGPT suggested to add game feel elements that included character animations (implemented in the form of slight idle movement on the minimalist character shapes), a UI indicator for when the player became invisible, color changing effects on the enemy for both when they detected the player and on the player for when they were low on health, and proximity-based sound effects when enemies moved on the map. The ChatGPT log for the roguelike genre game can be accessed here³.

³<https://chat.openai.com/share/32afedb3-6ef5-4578-81e7-a81874acdacf>

10.5 User Study

10.5.1 Website

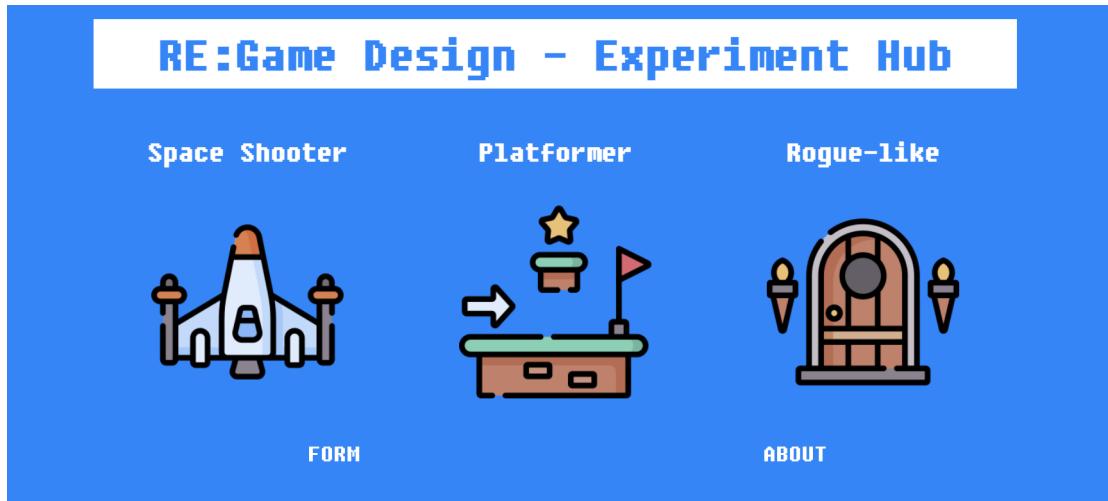


Figure 10.5: Screen-capture of the website where users could play each of the 9 games and provide feedback.

A website was hosted on Github Pages to serve as a hub for users to play the games. On the main screen, shown in Figure 10.5, was a link to all 3 genres and the user study form. The games were anonymized and the ordering of the base, human-designed, and ChatGPT-suggested games were shuffled so they could be evaluated by participants for quality, feel, and gameplay. Users would select a genre of game and then be given the anonymized games (labeled A-C) to play in order. This experiment setup was inspired by Mazzone’s art and AI collaborative output experiments, which asked participants to evaluate AI generated art in a Turing Test style experiment [81]. The games’ pages included instructions for how to play and control them below the game window as well as arrows to navigate between games. Figure 10.6 shows the base game for the platformer genre (Game C). The participants could play the games for as long as they liked. After 2 minutes, a button would flash orange on screen—either the ‘next’ button for the user to play the next game in the genre or the button to the user study form—to encourage users to evaluate all of the games in the study.

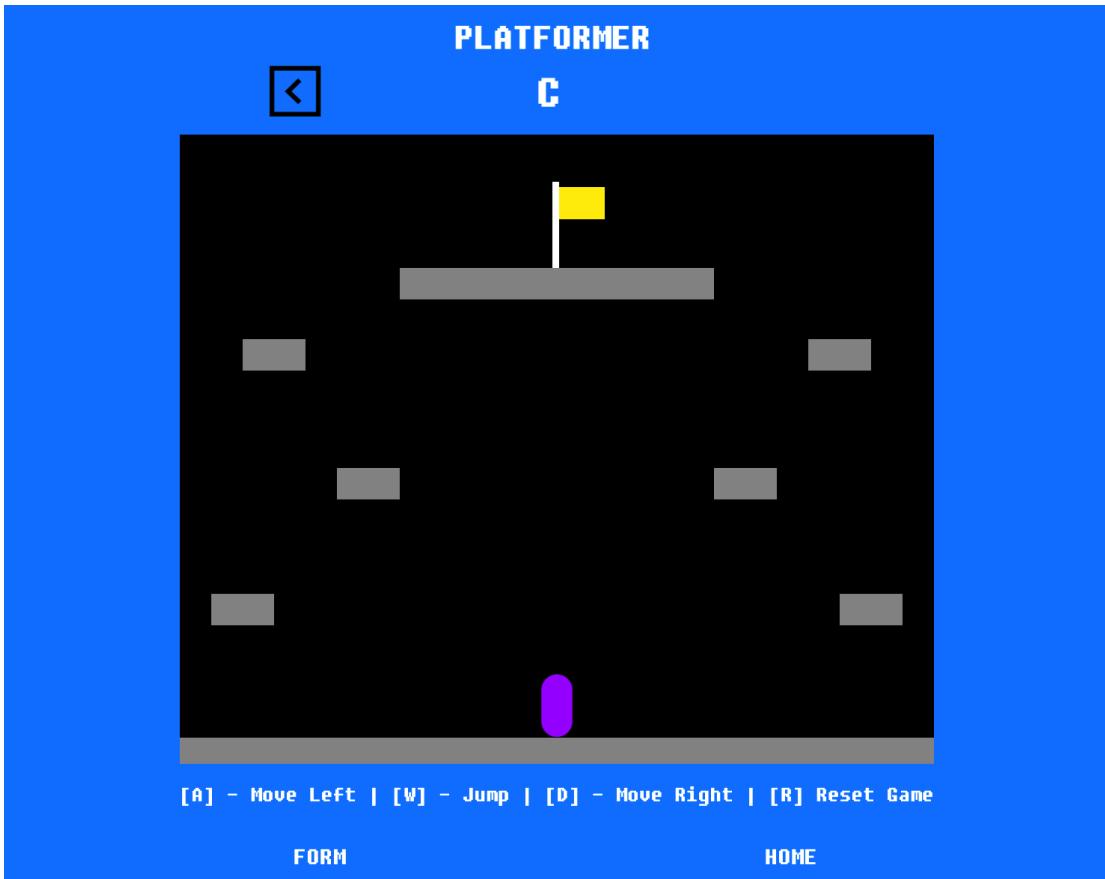


Figure 10.6: Page of the anonymized base platformer game shown to participants with instructions for how to play the game below.

10.5.2 Survey

The study was sent as a Google Form survey via Twitter posts, flyers (seen in Figure 10.7 as a co-designed ChatGPT-human creation), and Discord servers. For the first section of the form, we asked general demographic questions (listed in Table 10.1) concerning the gaming habits, their game design experience, and experience with AI. The following sections of the survey were divided based on the three genres of games (space shooter, platformer, and roguelike). The participants did not have to play all 3 genres, but were required to play all 3 games in the genre of their choice for proper comparison. After they played the games on the hub website, participants were asked to rank the games (described as they were on the website as game A, B, and C for anonymity) based on their aesthetic and qualitative evaluations. Finally, the participants were asked to give optional feedback on the games they played. The study received a total of 45 responses.

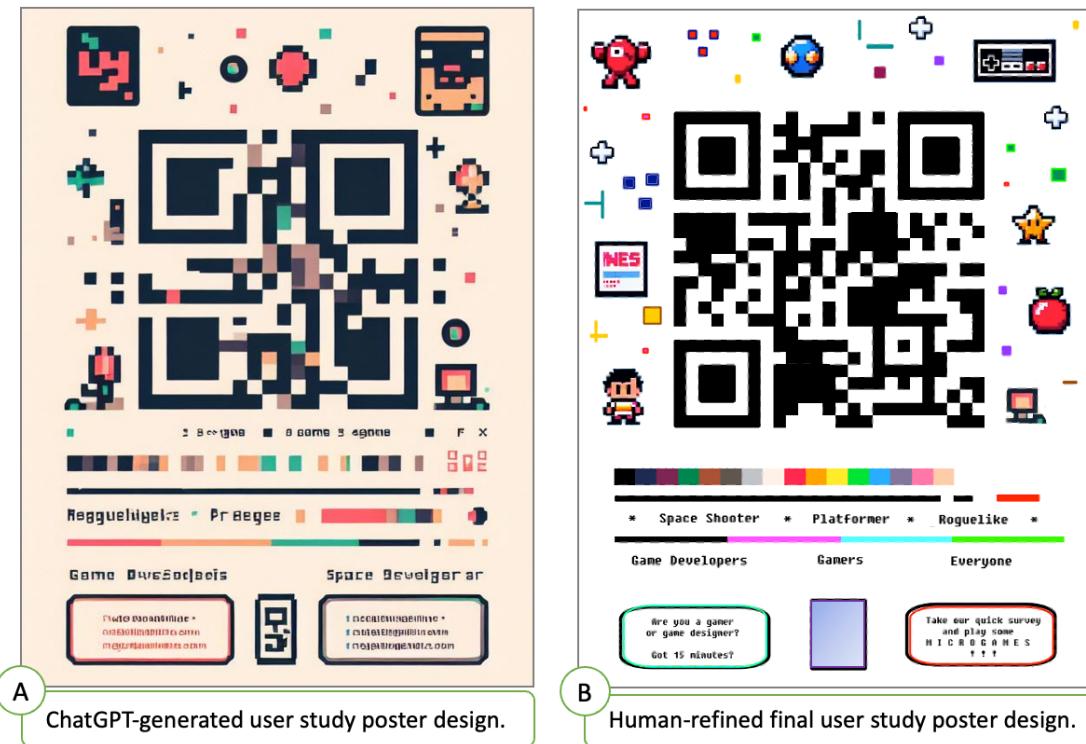


Figure 10.7: A poster used to recruit participants for this study. The left image shows the AI generated inspiration and the right shows the final human-edited version that was released.

10.6 Results

10.6.1 Demographic Information

All of the participants in the study played video games (2 or more hours a week) with 48% of participants playing 8 or more hours on average a week. We asked participants about the types of games they played in the context of publishers: independent (i.e. games published on user-based platforms such as Newgrounds or Itch.io and released by small indie studios such as Supergiant, Innersloth, Developer Digital, etc.) and/or triple-A games (i.e. games published largely on console systems and released by large game studios such as Nintendo, FromSoftware, Riot, etc.) 57% of participants reported to equally play both independent and triple-A titles and 31% played independent games more frequently. 53% of the participants had never designed a game, while 31% had between 1-11+ years of experience making games. 42% of those who had experience with game

Q#	Question
Q1	On average, how often do you play video games?
Q2	What are some of your favorite genres of video games to play?
Q3	What type of games do you play more? Independently made games or triple-A games?
Q4	What is your level of game design experience?
Q5	What kinds of games do you design?
Q6	Have you ever submitted a game or prototype to a game jam (either leisurely or for an assignment)?
Q7	Have you ever used an AI-assisted content generation tool?
Q8	On average, how often do you interact with a large language model?

Table 10.1: Demographic survey questions asked in the user study separated by general topic.

design had submitted a game or prototype to a game jam, while 24% reported an attempt to submit to a jam but had never finished a game for it. 64% of participants had experience using and interacting with an AI-assisted content generation tool (e.g. text-to-image generators, LLMs). 62% of participants reported interacting with a large language model between 1 to 4 hours a week.

We asked participants to list the genres of games they like to play and, if they designed games, to list the genres they like to design. Platformer games were tied with puzzle games for the most commonly designed game genre and the 5th most played genre. Roguelikes were the 4th most frequently designed genre and the 11th most played genre. While arcade games were not explicitly said as a genre, we consider the space shooter game an action game. Action games were the 2nd most played game and the 2nd least designed game. From this, we can gather that players were most familiar with action games, while game designers were more experienced with designing platforming games. Roguelikes were not as familiar to player or designer participants.

10.6.2 Game Ranking

For each game genre, we asked participants to rank the three anonymized games from “Best”, “Mid”, and “Worst” based on the following 6 categories of criteria:

- **Overall Preference** - “Which game did you like best? Which did you think was the worst?”
- **Game Feel** - “How good or ”game-like” a game feels as a player to play”
- **Innovation** - “How unique, different, or interesting the features are relative to the general genre of a game and to each other. Did the features feel like they fit the gameplay? Did they seem connected by a core idea? Did the player’s abilities fit together?”
- **Thematic Cohesion** - “How relevant the features were given the genre of the game. Did the features feel like they fit the gameplay?”
- **Most Interesting Abilities** - “Which player abilities did you find the most interesting given the genre”
- **Visual Presentation** - “Which game did you prefer in terms of visual presentation and aesthetics”

For example, for the “innovation” category a participant can rank the games as “Best” = A, “Mid” = C, and “Worst” = B for a given genre. We re-associated the games to their labels and analyzed the rankings for each category for each genre. Participants were not required to play all 3 genres of games, and some participants chose not to answer every question, therefore each question has a varying number of responses. Our experimental hypothesis was that the human-designed game would outrank the ChatGPT-guided game while both would outrank the baseline game.

To examine the difference in votes across the 3 games for each category, we created a scoring system based on the following equation $S_G = 3V_B + 2V_M + V_W$ where V_B is the number of votes the game received for being “Best” in the category, V_M is the number of votes the game received for being “Mid” in the category, and V_W is the number of votes the game received for being the “Worst” of the category. The “Best”, “Mid”, and “Worst” votes are weighted with values of 3,2,1

respectively and summed to equal the final score S_G for a given game G where $G \in [\text{Human game}, \text{ChatGPT game}, \text{Base game}]$. Table 10.2 shows the scores received in the category out of all of the votes (each category received different numbers of total votes). The platformer genre had the most variance in ranking, with 2 categories (game feel and visual presentation) going against the hypothesis. From this, there were potentially some evaluative disagreement and mixed opinion for comparatively evaluating the quality of the platformer games. The space shooter genre only had one category deviate (innovation). The roguelike category contained no scores that differed from the hypothesis.

	Space Shooter			Platformer			Roguelike (%)		
	Human	AI	Base	Human	AI	Base	Human	AI	Base
Overall	110	88	48	97	77	57	87	77	64
Feel	108	85	48	86	73	75	92	71	65
Innov.	93	95	49	104	76	48	93	65	52
Cohesion	91	80	63	98	66	58	79	66	65
Interest.	101	92	47	103	85	46	96	65	55
Present.	107	83	50	98	56	68	83	75	58

Table 10.2: **Scores for each game and genre using the formula S_G and category criteria** Models with the highest score are colored in green, second highest in yellow, and lowest in red. (AI = ChatGPT)

10.6.2.1 Statistic Analysis

In addition to the scoring system, we performed the Kruskal-Wallis and Friedman Tests on the rankings of all of the participants for each genre of game in each category. Both tests used an alpha value of 0.05. For all genres and all categories with an exception to the platformer “game feel” category, the null hypothesis was rejected for each test and there was a statistical significance. Tables 10.3, 10.4, and 10.5 shows the results of both tests for each genre on all participants.

Additional Kruskal-Wallis and Friedman tests were performed on the two groups of participants who reported not having any game design experience (non-designers) and that had experience (designers) again with an alpha value of 0.05 across all genres and categories. For the designer group, the space shooter’s

	Space Shooter (All Participants)					
	overall	feel	innov.	cohesion	interest.	present.
KW	70.50	78.32	54.076	24.02	71.00	88.34
P-val	4.89e-16	9.81e-18	1.80e-12	6.07e-06	3.81e-16	6.53e-20
F	47.4	52.66	36.37	16.15	47.74	59.4211
P-val	5.09e-11	3.66e-12	1.26e-08	0.0003	4.29e-11	1.24e-13

Table 10.3: Statistic Tests on the Space Shooter Genre games based on all participant responses (KW = Kruskal-Wallis Test Statistic; F = Friedman)

	Platformer (All Participants)					
	overall	feel	innov.	cohesion	interest.	present.
KW	21.36	4.46	66.75	24.19	62.31	13.58
P-val	2.30e-05	0.1075	3.20e-15	5.60e-06	2.95e-14	1.13e-03
F	14.37	3.00	44.89	16.27	41.90	9.14
P-val	7.58e-04	0.22313	1.78e-10	2.93e-04	7.98e-10	1.04e-02

Table 10.4: Statistic Tests on the Platformer Genre games based on all participant responses (KW = Kruskal-Wallis Test Statistic; F = Friedman)

	Roguelike (All Participants)					
	overall	feel	innov.	cohesion	interest.	present.
KW	20.03	21.94	51.85	12.48	52.59	27.49
P-val	4.47e-05	1.72e-05	5.51e-12	1.95e-03	3.80e-12	1.07e-06
F	13.47	14.76	34.89	8.40	35.39	18.50
P-val	1.19e-03	6.25e-04	2.65e-08	1.50e-02	2.07e-08	9.61e-05

Table 10.5: Statistic Tests on the Roguelike Genre games based on all participant responses (KW = Kruskal-Wallis Test Statistic; F = Friedman)

category of “cohesion” with the Friedman test and the platformer’s “game feel” and “presentation” categories for all tests did not have any statistical significance for the rankings and rejected the null hypothesis. For the non-designers, the platformer’s game feel category rejected the null hypothesis for the Friedman test only. Most notably, the roguelike categories of overall preference, feel, and cohesion all rejected the null hypothesis as well and did not show significant difference in both tests. Tables 10.6, 10.7, 10.8 shows the results of the tests for each genre for the designer group and tables 10.9, 10.10, 10.11 shows shows the results of the tests for each genre for the non-designer group.

As we noticed, roguelikes were not among the top 10 most played games, however, they were the 4th most made game genre. This correlation can lead us

to believe that designers would have a stronger opinion or aesthetic preference to the roguelike genre overall than the non-designers. Overall, participants almost consistently rated the human-designed games higher than the ChatGPT-suggested games, and both games were better than the control baseline game with no added game features.

	Space Shooter (Designer Participants)					
	overall	feel	innov.	cohesion	interest.	present.
KW	20.03	21.94	51.85	12.48	52.59	27.49
P-val	4.47e-05	1.72e-05	5.51e-12	1.95e-03	3.80e-12	1.07e-06
F	13.47	14.76	34.89	8.40	35.39	18.50
P-val	1.19e-03	6.25e-04	2.65e-08	1.50e-02	2.07e-08	9.61e-05

Table 10.6: Statistic Tests on the Space Shooter Genre games based on the designer group participant responses (KW = Kruskal-Wallis Test Statistic; F = Friedman)

	Platformer (Designer Participants)					
	overall	feel	innov.	cohesion	interest.	present.
KW	11.57	0.18	33.60	10.17	28.64	3.13
P-val	3.08e-03	9.12e-01	5.06e-08	6.19e-03	6.04e-07	2.09e-01
F	7.88	0.13	22.88	6.93	19.50	2.13
P-val	1.95e-02	9.39e-01	1.08e-05	3.12e-02	5.83e-05	3.44e-01

Table 10.7: Statistic Tests on the Platformer Genre games based on the designer group participant responses (KW = Kruskal-Wallis Test Statistic; F = Friedman)

	Roguelike (Designer Participants)					
	overall	feel	innov.	cohesion	interest.	present.
KW	23.01	27.16	41.70	11.75	41.70	22.21
P-val	1.01e-05	1.26e-06	8.83e-10	2.81e-03	8.83e-10	1.50e-05
F	15.65	18.47	28.35	8.00	28.35	15.13
P-val	4.00e-04	9.75e-05	6.97e-07	1.83e-02	6.97e-07	5.20e-04

Table 10.8: Statistic Tests on the Roguelike Genre games based on the designer group participant responses (KW = Kruskal-Wallis Test Statistic; F = Friedman)

10.6.3 Participant Feedback and Comments

While optional, we received a total of 22, 20, and 19 comments for the space shooter, platformer, and roguelike games respectively. 55% of the comments left were from participants that reported to have some experience designing games. We

performed simple word frequency analysis on the comments using SpaCy⁴ to look for commonality in themes and criticisms concerning the game. Some critiques were directed towards individual games while others critiqued the games at a general level. For readability, we refer to the games the users critiqued as their true labels for the following paragraphs.

For the space shooter games, 91% comments left by participants concerned the player “ability”, while 70% of the comments pertained to the “feel” of the game. Participants commented positively on the “cool animations” and “interesting”-ness of the abilities of the game. Many participants even offered suggestions for improvement on the abilities such as adding more enemies and changing the rate of fire. However, some participants had issues with the controls and accessibility of the game. Other found some of the sound effects “harsh” and the overall gameplay “easy” or “boring” and the abilities “unnecessary”. Some participants had positive comments about the ChatGPT-suggested game.

For the platformer games, all of the comments concerned the “feel” of the game and half concerned the “level”. Unfortunately, a majority of the comments were negative. Some participants stated the controls were too “confusing” and “horrendous” – particularly in the ChatGPT-suggested and the human-designed platformer games. Although they found the abilities and mechanics “interesting” and “floaty”, many people stated that the games were “not polished”. Some participants said that the base game, while many participants stated had better

⁴<https://spacy.io/>

	Space Shooter (Non-Designer Participants)					
	overall	feel	innov.	cohesion	interest.	present.
KW	33.17	46.06	36.41	22.70	40.42	48.22
P-val	6.26e-08	9.94e-11	1.24e-08	1.18e-05	1.67e-09	3.38e-11
F	22.45	31.18	24.67	15.36	27.36	32.67
P-val	1.33e-05	1.69e-07	4.40e-06	4.61e-04	1.14e-06	8.06e-08

Table 10.9: Statistic Tests on the Space Shooter Genre games based on the non-designer group participant responses (KW = Kruskal-Wallis Test Statistic; F = Friedman)

	Platformer (Non-Design Participants)					
	overall	feel	innov.	cohesion	interest.	present.
KW	13.64	6.05	30.51	11.81	30.75	15.75
P-val	1.09e-03	4.87e-02	2.37e-07	2.73e-03	2.10e-07	3.81e-04
F	9.24	4.10	20.67	8.00	20.82	10.67
P-val	9.86e-03	1.29e-01	3.25e-05	1.83e-02	3.02e-05	4.83e-03

Table 10.10: Statistic Tests on the Platformer Genre games based on the non-designer group participant responses (KW = Kruskal-Wallis Test Statistic; F = Friedman)

	Roguelike (Non-Design Participants)					
	overall	feel	innov.	cohesion	interest.	present.
KW	1.86	1.86	13.15	5.36	14.89	12.57
P-val	3.94e-01	3.94e-01	1.40e-03	6.84e-02	5.86e-04	1.87e-03
F	1.26	1.26	8.94	3.65	10.11	8.53
P-val	5.32e-01	5.32e-01	1.14e-02	1.61e-01	6.37e-03	1.41e-02

Table 10.11: Statistic Tests on the Roguelike Genre games based on the non-designer group participant responses (KW = Kruskal-Wallis Test Statistic; F = Friedman)

controls, had “boring” level design. One participant even suspected that “the games may have been by an AI” but could not identify which.

Lastly, the roguelike games were more varied in responses and had only 26% comments pertaining to the feel of the game and 21% related to the “time” of the game. Many of the comments related to the “feedback” given by the game to players and the general “combat.” However, there were some complaints about “broken levels” in the games, a possible generation error from the PCG levels. Many participants explicitly stated that they most enjoyed the free-movement of the shuriken ability from the human-designed game. Many participants stated that the movement of the players and enemies felt “stiff” and commented their “confusion” for the gameplay and saying the game was “unclear”.

10.7 Discussion

10.7.1 Human-ChatGPT Game Genre and Game Engine Knowledge Base

Since we, the authors, were the game developers and designers for the study, the overall quality of the human-designed games were limited by our skill levels. ChatGPT, as a general large-language model, can only imitate “reasonable” game design decisions and regurgitate relevant code snippets. The Unity3D engine has a large breadth of online resources available to developers through forum sites, in addition to online tutorials and open source code projects that provide code examples. With this information, ChatGPT can suggest code that is relevant to the task at hand and contextual enough for the Unity engine. Had this experiment been done in an engine with fewer online resources or one which does not rely on text-based programming such as Scratch, ChatGPT’s suggestions may not have been as helpful.

Understanding of common tropes, important gameplay elements, and basic problem solving was also an important factor in the human-designed games which separated them from their AI counterparts. In situations where problems would arise in the code implementation, ChatGPT would often struggle with suggesting fixes without pointed prompting from the designer, which required a deeper understanding of programming and certain aspects of Unity – something the model was not capable of comprehending and resolving, but an experienced game designer could.

ChatGPT performed best when it was prompted for abstract, high level game feature suggestions. These feature suggestions were genre relevant and specific, occasionally matching the features implemented for the human-designed games. Conversely, ChatGPT struggled with context, particularly when suggesting code snippets to implement into the game (i.e. the turn-based movement of the roguelike genre or camera follow feature for the platformer.) Therefore, we would recommend

ChatGPT as a tool for conceptual game ideas or for writing pseudocode over direct code implementation in a game engine.

10.7.2 Design Approach

Each of the human-designed games followed a sense of theme and cohesion – showing clear intent with the design process and reasoning for adding certain elements and mechanics to the games. The LLM, by contrast, would remain within the confines of what was asked of it, and could not replicate any semblance of the same vision. However, while a human designer may be more capable of retaining a thematic cohesion with elements of their game, this may close them off to external concepts that may deviate from the central theme. A designer can become narrow-minded in their creative process and less likely to “think outside the box” in terms of innovative and novel concepts that can be added to their game. Like the previous work with the abstract game feature recommendation and generation system, having an occasional suggestion that doesn’t follow the central theme of the work or goes against the designer’s intent can be the source of new inspiration. The LLM offered a much wider range of suggestions that went outside the themes established for the genres later during the development process that could have led to much more interesting and innovative game mechanics that the human designer had not considered (i.e. the gravy flip mechanic for the platformer and the energy resource consumption mechanic for the roguelike.)

10.8 Summary

This chapter describes a study on the large language model ChatGPT as a collaborative game designer. 9 games across 3 genres were made by human designers and ChatGPT output code and feature suggestions and evaluated blindly in a Turing-Test style evaluation. While ChatGPT was able to produce fully satisfactory and structurally sound games in a collaborative and leading role, it had no direct

ability to replace a human designer's creative goals, understanding of important context, and ability to identify what "felt" good to a player. Participants of the study found the pure human-designed games to overall be better, however the games produced by the ChatGPT suggestions were found to be of higher quality than the baseline games which contained no features whatsoever.

These findings reject the notion that AI could act as a sufficient replacement for humans in game design, despite their ability to be highly effective tools in that process. However, certain observations from each game – including their ability to innovate and employ interesting gameplay features – indicated that the AI games had enough appeal for players to want to see them explored in full. These results, compounded by the human designer's own acknowledgment of their shortcomings in ideation, varying skill levels, and the expanded outlook they gained through exploring ChatGPT's suggestions indicate that AI can have an effective role in the creative process of game making without undercutting the work of game developers.

10.9 Reflection

This project demonstrated the creative and collaborative capabilities of ChatGPT. Honestly, I was surprised that it was able to effectively give such helpful game feature ideas and useful code snippets. It sometimes had a difficult time adjusting to context for the base games at times (i.e. designing in the PCG dungeon of the roguelike game,) and occasionally lacked in suggesting or implementing key game feel elements (i.e. making the player move more fluidly in the platformer genre.) Nevertheless, for a novice game designer, ChatGPT and possibly other LLM or text-interfacing systems could be very helpful and improve their game development skills. Even for expert designers, ChatGPT can offer some interesting or novel game feature suggestions at a high level of design that the expert can implement into their game later.

In the Game Design class I taught during the semester following the development

of this project, I ran a survey asking the students what kind of generative AI they incorporated into their game development for one of the prompts. Many of the students said they used ChatGPT, either for generating code snippets, generating thematic dialogue, for coming up with game feature ideas, or even giving the premise for the entire game with the themes and genre. The accessibility and ease of use for communicating with ChatGPT compared to other content generators that are more directly integrated into game design, can make for a lower barrier of entry for novice game designers like many of the students in my class. Even outside of the prompt, many students continued to use ChatGPT for assistance with their game design projects, citing that it often sped up their development time. I think with more iterations of ChatGPT and the better we as a field get at prompt engineering, using LLMs in game development will become more mainstream and more versatile to our needs.

10.10 Takeaways

- ChatGPT is good for abstract, game feature recommendations if you're a novice game designer or have no starting ideas
- ChatGPT is limited for the context of the game you are trying to design (i.e. engine, genre)
- Collaboration and creative intent, regardless of whether with another human or an AI, is difficult to communicate
- Prompt engineering for LLMs is key to better quality and more specific content output
- When possible, statistic analysis on user study responses provide more insights into user perception of a system
- Splitting responses based on demographic data shows more information on user evaluation

- If possible, try to get as diverse of a demographic group for a user study – even outside of the intended audience
- Participants that also design users studies (i.e. researchers) will be more likely to participate in another user study, but also bring their own biases to their responses
- Game designers will be more likely to provide feedback than participants that don't regularly or have never designed games

Chapter 11

Evolving Abstract Multi-Agent AI Environments

The works described in the earlier chapters use content generation in game domains that have well defined mechanic spaces, such as the grid-space worlds of *Baba is You*, *Minecraft*, and the GVG-AI framework. Content generation has also been applied to abstract game design elements such as game feel and game feature ideation. However, there is a lack of work in both procedural content generation and mixed-initiative content generation for designing entire environments and novel domains; particularly those that are abstract and open-ended.

Open-ended environments allow for a range of interesting and unrestrained artificial agent interactions to occur. Depending on the limitations and rules of the environment, agents are capable of interacting in ways that the developers of the system would be unable to foresee or design manually. However, if a specific interaction needs to be demonstrated from an open-ended system, it can be difficult to rediscover the emergent and potentially accidental behavior. For example, many player experiences in simulation games such as the Sims or Dwarf Fortress are unique to the player and difficult to recreate exactly. Having the ability to recreate a similar emergent behaviors would allow us to find starting points for new experiences that would otherwise be nearly impossible to achieve without a complex

and specific sequence of events. Automatically generating these environments to facilitate these emergent behaviors can also open AI research to new challenges in multi-agent spaces and define new mechanics and interactions that could not have been pre-programmed.

This chapter looks to explore more abstract and open-ended forms of content generation by evolving and collaboratively designing multi-agent simulation environments. The Amorphous Fortress is an open-ended simulation framework for evolving artificial life with abstract and emergent behaviors. This system allows us to observe the interactions that emerge in multi-agent simulations and how innovative and interesting behaviors can be generated from a small pre-defined set of primitive actions and conditions in a confined space. Two evolutionary experiments are conducted on the Python version of the framework that look to generate both the agent definitions[23] and entire environments[41] while maintaining abstraction with their emergent behavior and interactions.

11.1 Amorphous Fortress Engine

The Amorphous Fortress Python framework¹ is an artificial life simulation that shows simple agents in a contained grid-world similar to that of classic terminal-based dungeon-crawler roguelike games such as Rogue, NetHack, and Dwarf Fortress. Figure 11.1 shows an example generated environment in the Amorphous Fortress with many different agents (represented by ASCII symbols.) The design of the Amorphous Fortress framework was inspired by the VGDL system for the GVG-AI framework as mentioned in Chapter 3. Both the VGDL and Amorphous Fortress frameworks are designed around simple and explainable game mechanics to form multitudes of games [62]. The system is made up of a hierarchy of 3 components: entities (the agent class of the system) the fortress object (the environment class of the system) and the engine (the “manager” and main loop of the simulation). A configuration file can be provided upon initialization of the system to define

¹<https://github.com/dipikarajesh18/amorphous-fortress>



Figure 11.1: Screenshot of a generated environment in the Amorphous Fortress framework containing multiple instances of autonomous agents

a particular range of interactions and allow for simulation reproducibility in the framework. The following subsections describe each component of the framework.

11.1.1 Entities

Entities in the Amorphous Fortress framework are made up of finite state machines (FSMs.) The simplicity of the FSM’s structure was chosen as the basis for the agent definitions in order to explore variance with the behaviors that could emerge from such a limited design. Each “entity” of the Amorphous Fortress is defined by a singular ASCII character, a unique 4-bit identification hex number, a list of nodes, and a set of edges.

The ASCII character-rendering was directly inspired by classic terminal-based rogue-likes such as Rogue and Dwarf Fortress and also allowed for more anthropomorphism with the objects and classes without explicit definitions of their identities. These characters can be any symbol that can be represented on a terminal but lack

the color variance found in text-based games. This definition was also inspired by the versatile semantic and functional assignments of the sprite objects in the Baba is You game and Baba is Y'all system (see Chapter 4.)

Because the system uses a finite-state machine as the basis of interaction and behavior, each node in the FSM graph represents a potential action state the entity object can be in. These actions are pre-programmed and define how an entity interacts with the environment. The possible action nodes for the Python framework are defined in Table 11.1.

Function	Param.	Description
idle	-	the entity remains stationary at the same position
move	-	the entity moves in a random direction (north, south, east, or west)
die	-	the entity is deleted from the fortress
clone	-	the entity creates another instance of its own class
push	char	the entity will attempt to move in a random direction and will push an entity of the specified target character into the next space over (if possible)
take	char	the entity removes the nearest entity of the specified target character
chase	char	the entity will move towards the position of the nearest entity of the specified target character
add	char	the entity creates another instance from the class of the specified target character
transform	char	the entity will change to a different entity class - thus altering its FSM definition entirely
move_wall	char	the entity will attempt to move in a random direction unless there is an entity of the specified class at that position - otherwise it will remain idle

Table 11.1: List of action nodes to define the agents

Each agent is initialized with the “idle” node as the starting base node, then a random subset of the possible action nodes is chosen from the simulation’s provided configuration file to construct the rest of the graph. Each node can only be selected and added to the graph once to avoid redundancy and prevent over-complicated graph structures.

Edges of the agent’s FSM definition are conditions that can occur within the simulation to allow node transitions of the agent from state to state. Like the

Condition	Param.	Definition
none	-	no condition is required to transition states
step	int	every x number of simulation ticks the edge is activated and the node transitions
within	char, int	checks whether the entity is within a number of spaces from an instance of another entity with the target character
nextTo	char	checks whether the entity is within one space (north, south, east, or west) of another entity of the target character
touch	char	checks whether the entity is in the same space as another entity of the target character

Table 11.2: Entity FSM conditional edge definitions (ordered by least to greatest priority)

nodes, these conditions are also pre-programmed. In the case that a node has multiple edges, the conditions are ordered by priority within the system and the edge with the highest conditional priority transitions to its endpoint state. For this system, the possible condition edges to connect the action-state nodes of the graphs are shown in Table 11.2 in the order of least to greatest priority.

The edges and their conditions are directional in relation to the nodes (i.e. an edge from node 0 to node 1 may have a different condition and definition than an edge from node 1 to node 0.) The entity can have from 0 to $n \times (n - 1)$ edges. Like the node generation of the FSM, the edge connection and conditions are randomly generated from the subset of possible conditions provided in the simulation's configuration file.

At any time during the simulation, the entity is always in a state at one of the set nodes. At each timestep—or update within the fortress environment—each connection is evaluated to move to the connecting node state based on whether the conditions are met, in order of priority defined internally. The agent then performs the action at its new current node on the next timestep. Figure 11.2 shows an example entity and its associated FSM definition and active node in the simulation.

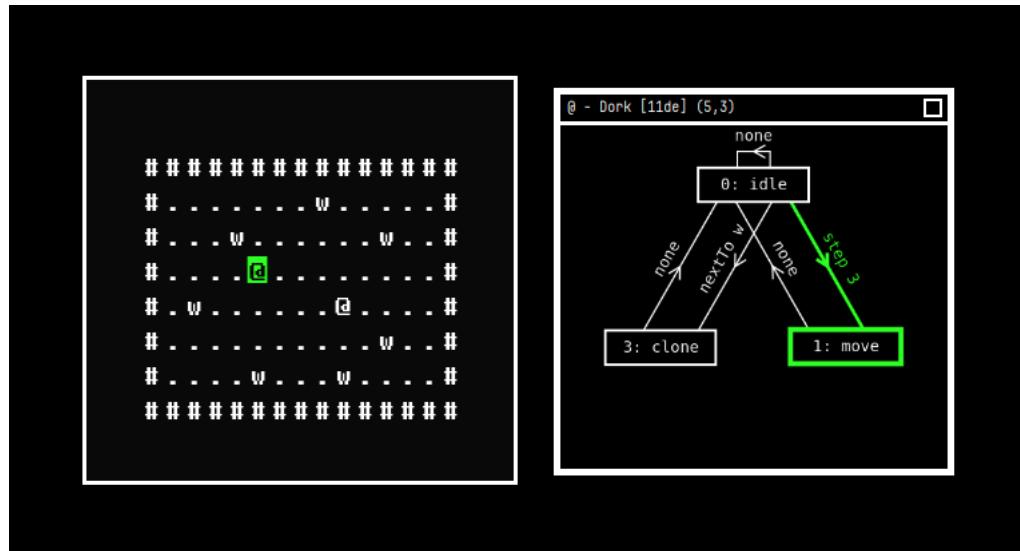


Figure 11.2: The ‘dork’ entity (highlighted in green on the map on the left) and its FSM definition (right window) with the active node (move) and most recently transitioned conditional edge (step 3) outlined in green.

11.1.2 Fortress

The “fortress” class of the Amorphous Fortress contains the environment where the simulation takes place and stores general information accessible to all of the entities in the fortress. The borders and size of the fortress are defined initially with a set character width w and height h to enclose the entities. On initialization, the fortress generates each entity class FSM for each character defined in the passed configuration file. This global dictionary of entity classes allows any instance of an entity to add or transform different entity instances even if none currently exist on the map. The fortress maintains a list of currently active entity instances in the simulation and adds or removes them if called by ID value from an entity instance. The fortress also maintains positional data about each entity to return for conditional checks (i.e. whether a particular position has an entity located there.) A log of actions is maintained and keeps track of each action taken by each entity and the timesteps at which they occurred during simulation. This log is exported upon the termination of the simulation by the engine. The fortress has 3 termination functions that are checked by the engine defined as follows:

- *extinction*: occurs when there are no entities at all left in the simulation

- *overpopulation*: occurs when there exist more than $2(wh)$ entities in the simulation
- *inactivity*: occurs if an action has not been taken by an entity in the last x timesteps

The cause for termination is added to the end of the log, along with every entity class definition's FSM tree, and labeled by the seed used to generate the simulation for reproducibility.

11.1.3 Engine and Main Loop

The engine serves as the entry point to the system and maintains the entire simulation as well as the update loop for the entities. The engine also imports the configuration file to pass to the fortress and to the entities - including the seed value for reproducible randomization. Once the entities and their FSMs are populated in the Fortress, the engine starts and updates the simulation. Each entity is evaluated at its current node and conditions are checked from the edges to move from that node onto a connecting node if possible. The engine continues updating the simulation until any of the termination conditions have been reached.

11.1.4 Configuration File

A configuration file that defines the action space and conditions can be provided to the system during initialization for the purpose of reproducibility. The parameters that can be specified in the configuration file are defined in Table 11.3.

11.1.5 Example Scenario

To illustrate the potential of the system to generate plausible simulation dynamics, the FSM entity class definitions are used to manually design two simple example environments. These environments, inspired by the recent franchise entry *The Legend of Zelda: Tears of the Kingdom*, contain three entity class definitions:

Setting	Input	Description
seed	<i>str</i>	the seed value to specify a particular seed for reproducing that particular simulation or ‘any’ to create a new simulation
character set	<i>array (str)</i>	the possible set of ASCII characters to represent the entities
action space	<i>array (str)</i>	the possible set of action nodes an entity can be initialized with for its FSM graph
edge conditions	<i>array (str)</i>	the possible set of conditions for a node to transition into another node for its FSM graph
step range	<i>(int, int)</i>	the range of timesteps to randomly initialize the step edge condition
prox range	<i>(int, int)</i>	the range of spaces to randomly initialize the within edge condition
save log	<i>boolean</i>	‘True’ if the logs for the simulation should be saved, ‘False’ if not
min log	<i>int</i>	the minimum number of steps for the simulation to have to save the log
log file	<i>str</i>	the path to the file where the logs can be saved
inactive limit	<i>int</i>	the number of seconds to let the simulation run before stopping due to inactivity
pop percent	<i>float</i>	the chance for a copy of the entity’s class to be added to the population

Table 11.3: List of configuration parameters that can be set for the Amorphous Fortress Python framework

‘Link’, ‘Bokoblin’ and ‘Korok’. In the Zelda franchise, Koroks are characters normally revealed after the player, controlling the character Link, completes a puzzle or interacts with a unique hidden object, at which point the Korok gifts the player a seed that can be used as in-game currency. Bokoblin characters in the game also interact with the character Link by chasing him when he comes into range and trying to attack him. Figure 11.3 shows a screenshot of both scenarios in game and recreated in the framework.

For scenario A, a single Link entity L and a single Korok entity k are initialized in the fortress with grass entities w placed as decoration. Link’s FSM is defined such that the character will *move* randomly on the map every 2 *steps*. When the Korok (k) is *next to* Link (L), it *transforms* into a seed, ‘\$’. When Link *touches* the seed, he *takes* it, removing the ‘\$’ character from the map. For scenario B, a

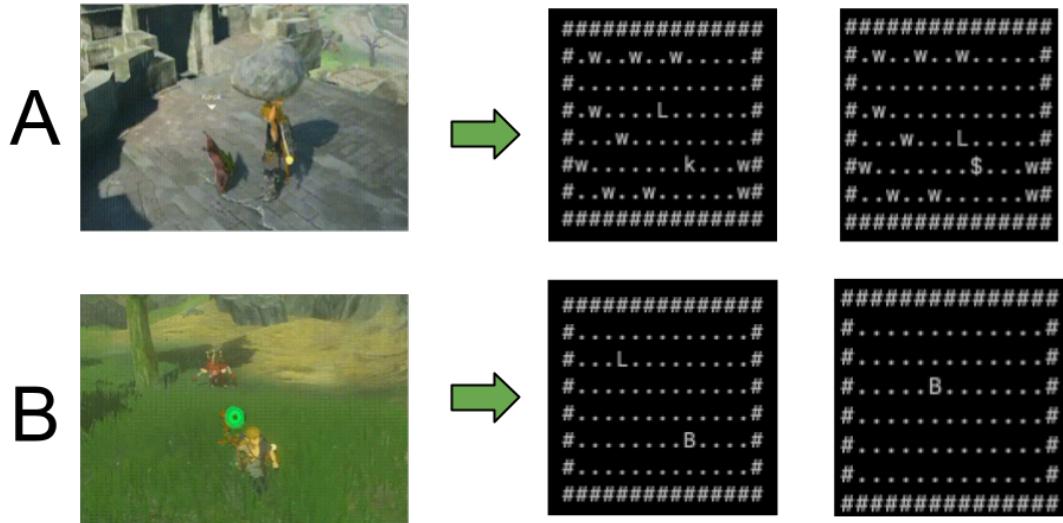


Figure 11.3: Two example agent interactions from *Zelda: Tears of the Kingdom* remade in the Amorphous Fortress framework

Bokoblin entity B will *move* randomly in the fortress. If Link is *within* 5 spaces of the Bokoblin, the Bokoblin will *chase* him. If the Bokoblin is *touching* Link, Link will *die*.

11.2 Hillclimber Experiment

To explore the potential depths of the generated finite state machine behaviors, we implement a hillclimbing evolutionary algorithm to evolve the agents and fortress towards having the largest, but also most traversed FSMs.

11.2.1 Experiment Setup

Three different mutation algorithms were implemented for evolution. The first involved mutating the nodes of randomly selected entity classes. This was done by either 1) deleting random nodes (so long as there were more than one node in the graph) 2) adding more nodes (if there were any that have not been added already) or 3) replacing nodes with another action. Similarly, the second mutation modified the edges by either 1) deleting random edges (so long as there was more than one edge in the graph), 2) adding edges between nodes, or 3) overwriting the

conditional check on an edge. The third mutation modified the number of instances in the fortress by adding or removing instances. After mutation, the modified FSM was pruned up for any unconnected edges and orphaned nodes after mutation of the finite state machines. The mutation was done by coin-flip chance, given some pre-set probability for both the node level, edge level, and instance level mutation. Algorithm 2 demonstrates the mutation process for the evolutionary system of the fortress.

The mutations of this algorithm were applied at the entity's class level definition - such that each future instance of an entity class would all have the same modified FSM tree definition. The number of instances initialized in the fortress were also mutated from generation to generation. We chose to use the hillclimber evolutionary algorithm for this study as we were only concerned with exploring how the fortress and simulation could gradually evolve toward an objective as a whole. The hillclimber algorithm for this experiment also allowed for the simplest open-ended emergence for novel agent behaviors.

In this experiment, we wanted to encourage more diverse and deep behaviors from the generated FSMs of the fortress. Therefore, our fitness function was based on if a node or edge of an FSM was traversed during the entire course of the simulation. The fitness function, f , for evolving the fortresses using the hill-climbing algorithm was defined as the following equation: $f = \frac{v}{u+1} \times t$ where v is the sum of the number of traversed (or visited) nodes and edges of every FSM in the fortress, u is the combined number of unused (or unvisited) nodes and edges of each FSM, and t as the total number of nodes and edges in the entity class FSMs. The visitation of a node or edge in an FSM was determined by the combined behaviors of every instance of a particular entity. Because some instances of an FSM may have different experiences and interactions based on initial placement, we used the combined behaviors of all instances of an entity class presented in the fortress for the fitness function. For example, if a single instance of the '@' class completes an action that is part of the FSM's tree due to a

Algorithm 2 Mutation function for the Fortress

Input: *node_prob*, *edge_prob*, *instance_prob*

node_r = random()

edge_r = random()

instance_r = random()

```

/* Mutate random entity class nodes                                */
1 while node_r < node_prob do
2   | i = random(0,2) e = random(fortress.ent_def) if i == 0 then
3   |   | fortress._delete_node(e)
4   | else if i == 1 then
5   |   | fortress._add_node(e)
6   | else if i == 2 then
7   |   | fortress._alter_node(e)
8   | node_r = random()

/* Mutate random entity class edges                                */
9 while edge_r < edge_prob do
10  | i = random(0,2) e = random(fortress.ent_def) if i == 0 then
11  |   | fortress._delete_edge(e)
12  | else if i == 1 then
13  |   | fortress._add_edge(e)
14  | else if i == 2 then
15  |   | fortress._alter_edge(e)
16  | edge_r = random()

/* Mutate random entity instances in the fortress                */
17 while instance_r < instance_prob do
18   | i = random(0,1) e = random(fortress.entities) if i == 0 then
19   |   | fortress._remove_entity(e)
20   | else if i == 1 then
21   |   | x,y = random(fortress.pos) fortress._add_entity(e,x,y)
22   | instance_r = random()

```

conditional check successfully occurring, both the node it traveled to and the edge condition it traveled by would be considered 'visited' for the entire entity class of '@'. This fitness function was also defined to encourage both the generation of large but thoroughly explored FSMs for the fortress overall and not just by a single entity class.

For this study, we examine the evolution of 5 fortresses each starting from randomly generated seeds. The engine randomly selects from the character class definition to create new instances to add the fortress at random positions. We evolved the fortress for a total of 1000 generations – comparing the fitness of one new

mutated fortress to the best-generated fortress. If the generated fortress achieved a higher fitness score, the best fortress would be replaced by the current instance, and so on. The fortress was simulated for 20 timesteps before being evaluated for entity tree traversals. This was to limit the computational calculations of each instance in the fortress while still allowing for depth and interaction with the agent behaviors. This study used a 50% independent probability for all mutation chances described before – a 50% chance to mutate a randomly selected entity class’s FSM nodes, edges, and/or to add or remove an instance from the fortress. A total of 15 unique ASCII characters were used for the experiments, and all edge conditions and node action conditions were allowed for selection.

11.2.2 Results

We observed a number of relatively interesting behaviors from the evolutionary experiment concerning the final entity FSM definitions, the fortress construction, and the trends of the fortress evolution. Figure 11.4 shows the fitness score composition over the 1000 generations over the 5 trials, which steadily improves over time. The following subsections describe these phenomena in more detail along with our insights and hypothesis for these emergent behaviors.

11.2.2.1 FSM Objective Coverage

A large majority of the entity classes in the final best fortresses had high graph traversal coverage overall. The calculated average across all 5 fortresses for every entity class had an average of 72% node coverage and 58% edge coverage. The function was designed with the intention to reward large, well-traversed finite state machine graphs - while giving less incentive to generate large graphs with less traversal and small graphs with fuller traversal. While many generated entity classes had large graphs - defined as having 7 or 8 nodes and between a range of 15-30 edges - and strong node/edge coverage, there was also a small number of entities that had significantly smaller graphs - with 2 or fewer nodes and less than

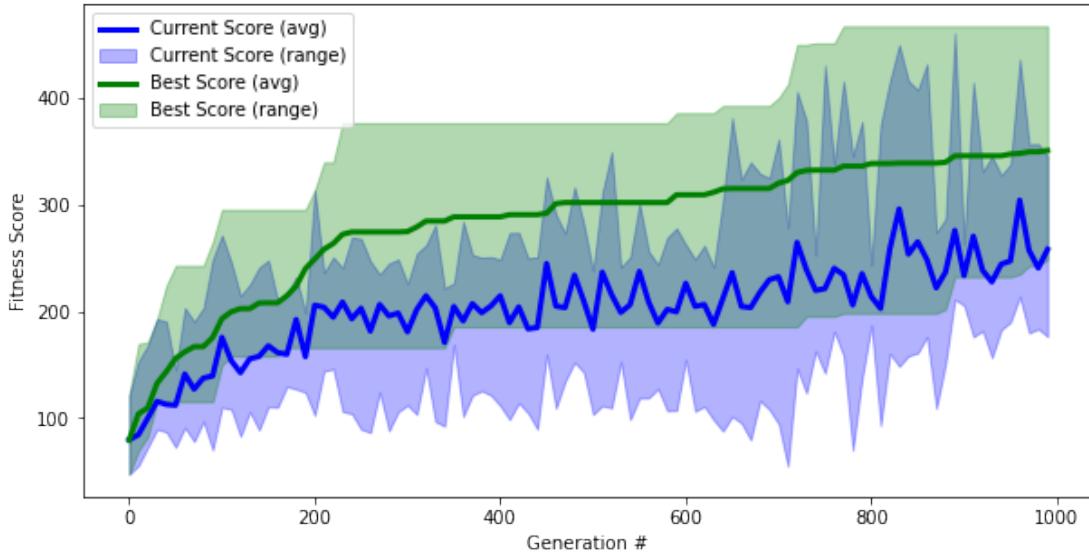


Figure 11.4: Best and current fitness scores of the 5 experiment trials (averaged and min/max range)

5 edges. This diversity of entities regardless of the size of the graphs may have been due to a naturally emerging “symbiosis” of the entities and their relationships to each. For example, although one entity class may not have a more actively influential node such as *take* or *add*, the conditional edges of another entity’s class could have depended on the presence of that entity in order to do something more influential — i.e. another entity may have needed to be *next to* or *touching* the small-graph entity in order to *clone* itself or *transform*.

11.2.2.2 Rainbow Goop FSM Generation

Because of the objective function, the FSM shifted towards having “reproductive” behaviors. The entities evolved towards focusing on having node actions that created more of themselves but also changed themselves into another entity. As such, the *add*, *transform*, and *clone* nodes were seen more frequently in the entities graphs than more destructive nodes such as *take* and *die*. Based on the initial and final map renderings, many of the initial entities on the map would try to clone itself to create more copies of itself — this allowed for deeper graph exploration to fulfill the “visited” over “unvisited” node statistic. Afterward, many entities would often *transform* into other entities — to fulfill other missing nodes and edge

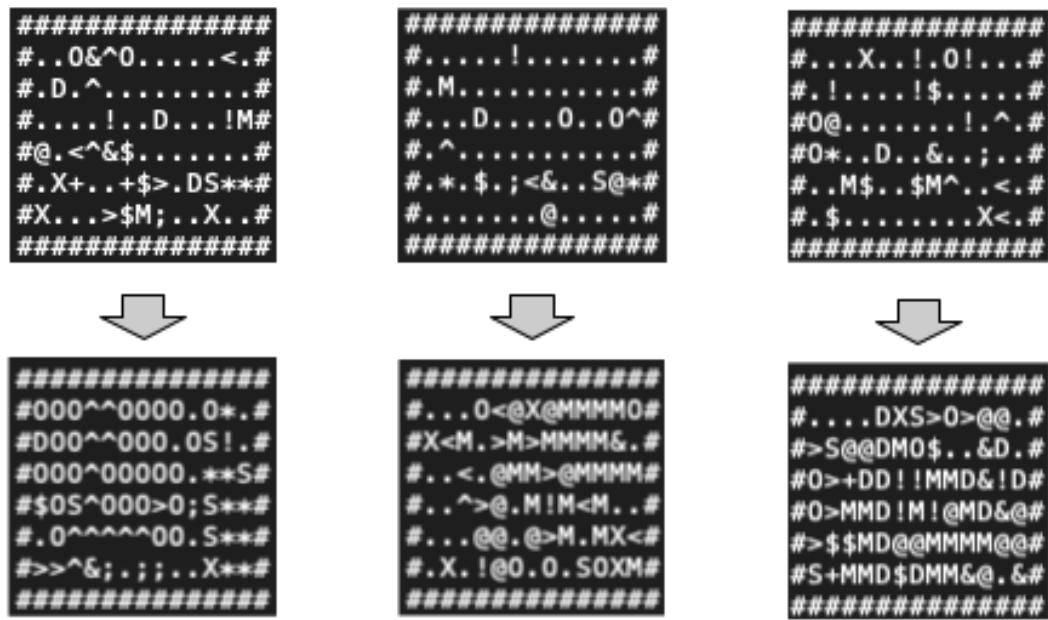


Figure 11.5: Initial and final renderings of the fortress for 3 seeds

visitations concerning other entities that may not have initially been on the map. This explains the copious — almost overpopulated — amount of entities in the final rendering of the fortress simulation. Figure 11.5 shows the initial fortress renderings and the final fortress renderings after 20 steps of simulation. More diversity of characters and entities on the map also encouraged more interactions overall. We call this type of behavior “rainbow goop” as the behavior replicates itself, but also tries to transform itself into other entities to diversify the fortress.

11.2.3 Reflection

With this experiment, we were able to evolve the fortress environments towards discovering emergent AI interactions and co-operative behaviors that formed a sort of “ecosystem” within the environment. The fitness function defined for this simple experiment led the agents to exhibit symbiotic relationships, where entity classes with both large and small FSM graphs were integral to benefit both classes’ survival in the fortress. From this, a variety of entity classes could emerge, with diverse policies of agent behavior that depend on one another for deeper exploration

of their own graphs. This led us to perform another experiment to try to find multiple fortresses that exhibit different kinds of relational behavior.

11.3 Quality Diversity Experiment

This second experiment implements the quality diversity (QD) algorithm MAP-Elites to evolve an archive of fortress environments. Diversity is maintained in these fortresses by their defined behavior characteristics (BCs). These BCs are measured based on the agent action space class definitions and the ending state of the fortress after simulation. Quality of the fortresses is maintained by evaluating what proportion of each entity class's finite-state machine definition was explored collectively during simulation. The following subsections detail the follow-up experiment using the Amorphous Fortress framework.

11.3.1 MAP-Elites for Amorphous Fortress

We implement the MAP-Elites QD algorithm [85] to evolve multiple fortresses towards a diversity of emergent behaviors. The emergent behaviors of the entities defined within this system are dependent on interactions with other instances within the same fortress. Therefore a single cell of the MAP-Elites grid contains a fortress with its own set of entity class FSM definitions. Figure 11.6 illustrates a small example of the evolution and evaluation process as the fortresses are placed in the MAP-Elites grid for the experiment (described in more detail later in this section.)

11.3.1.1 Behavior Characteristics

For a MAP-Elites implementation, the dimensions of the archived QD grid are known as the behavior characteristics (BCs.) These BCs designate how the individuals from a population are separated and maintained for sample diversity and replaced within the cell to improve quality. We define the following behavior

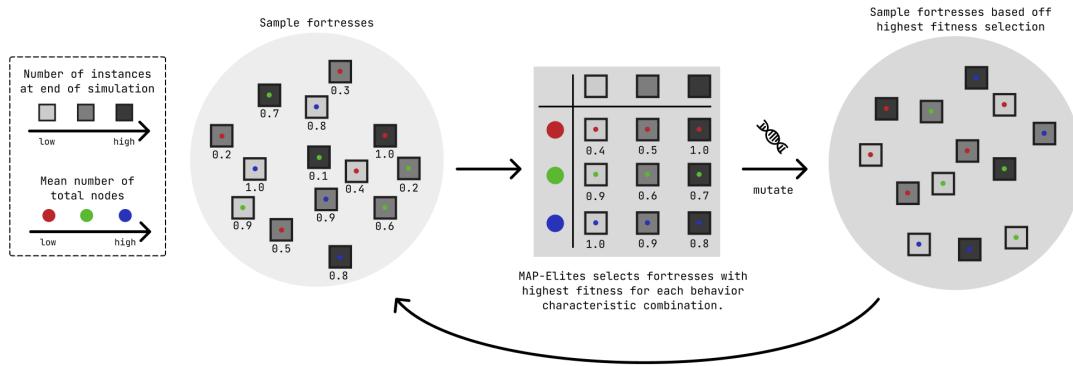


Figure 11.6: Diagram of the MAP-Elites algorithm applied to the Amorphous Fortress QD experiment

characteristics that are used for experiment: a) the mean number of total instances in the fortress at the end of the simulation (population) and b) the mean number of total nodes across all entity class definitions (total FSM size.)

For behavior characteristic *a* based on the number of entity instances, this dimension is intended to explore the population of a fortress, whether the entity class combinations result in an overpopulation of entity instances, an extinction of all instances, or a stability or “equilibrium” of instances within the fortress. The values of this dimension can range from 0 to 156—the maximum number of instances allowed to exist in the fortress before it terminates based on an “overpopulation” condition.

Behavior characteristic *b*, based on the collective class FSM size, looks to examine the “complexity” and “depth” of the entity classes; whether the combined set includes a majority of simple entity class definitions with only 1 action node or conversely with extremely large entity class definitions. The values of this dimension can range from 15 nodes—where each of the entity classes has only 1 node—to 1400 nodes—where each possible node is included every entity class FSM definition. The exploration of this dimension by the MAP-Elites algorithm will demonstrate the growth and utility of varying sized entity class FSMs.

11.3.1.2 Population

An evaluated population consists of a fortress with a set of agent entity class definitions. The population size for this experiment was 10 individuals per generation—9 sampled from elites and 1 randomly created similar to the initialization. The singular random fortress is injected into the set of mutant elites to encourage exploration within the MAP-Elites grid and to prevent the algorithm from reaching a local minimum during evolution. We parallelize the evaluation of these 10 individuals to speed up evolution. On initialization, all 15 of the FSMs are defined for each fortress individual. In this step, the number of total nodes to be added over all FSMs in the fortress is sampled uniformly to encourage a larger spread of randomly initialized individuals over the MAP-Elites grid (along the n. nodes axis), thus allowing for more uniform exploration of cells as evolution progresses.

We randomly initialize fortresses so as to sample uniformly along the axis measuring the number of aggregate FSM nodes. We first uniformly sample this aggregate number, then split it into as many summands as there are entity types using an evenly weighted multinomial distribution, where each summand corresponds to the number of nodes to be assigned a given entity. It is possible in this setting for an entity type to be assigned more nodes than there are distinct node types; in this case, we (greedily) re-assign the surplus nodes to one or more non-overfilled entity classes, until no surplus nodes remain.

11.3.1.3 Mutation

A fortress individual in the population is mutated by modifying its genotype: the class level definitions of the FSMs. Each fortress contains 15 entity classes, where each class can have a minimum of 1 action node and a maximum of 95 action nodes. This process is done similarly to the first experiment. However, as a modification for the MAP-Elites experiment, the node mutation is adjusted to increase exploration within the grid. Unlike the previous experiment where a single node was modified per coin-flip chance, a range of nodes can be added, removed,

or altered into another action node definition. For example, 10 nodes can be added to one entity class definition, while 4 are removed from another (or the same if randomly chosen again).

11.3.1.4 Fitness

The fortress sample individuals are evaluated based on the ending state of the fortress. The fortress is simulated for 100 steps, where each instance of an entity class present in the fortress enacts the current action node of its FSM graph once per step and then evaluates the next action node to move to based on the state conditions it ends in. The fitness function of the MAP-Elites implementation of the system is similar to the hillclimber experiment. This fitness definition encourages each class entity to have the full possibility of its emergent behaviors demonstrated within the simulation. The final exploration of a class definition's nodes and edges are also aggregated over evaluation trials in case different behaviors occur due to different seed evaluations. The fitness function for a fortress is defined with the following equation: $f = e/t$ where f is the fitness value from 0 to 1, e is the total number of explored nodes—nodes that were activated during simulation—for all entity class definitions in the fortress individual and t is the total number of nodes—activated or un-activated—for each entity class in the fortress individual.

11.3.1.5 Entropy of the FSM definitions

Entropy examines the distribution of the sizes across the entity class definitions. The values of this dimension ranges from 0 to 1, with 0 meaning all of the entity class FSMs have the same number of nodes and no variation, and 1 meaning the number of nodes are different for each entity class FSM definition. We use Shannon Entropy to calculate the entropic value of the FSM sizes with a b base N where N is the number of FSM size bins (which for this experiment is equal to the number of entity classes).

Figure 11.7 shows how the BCs, fitness value, and entropy value are calculated

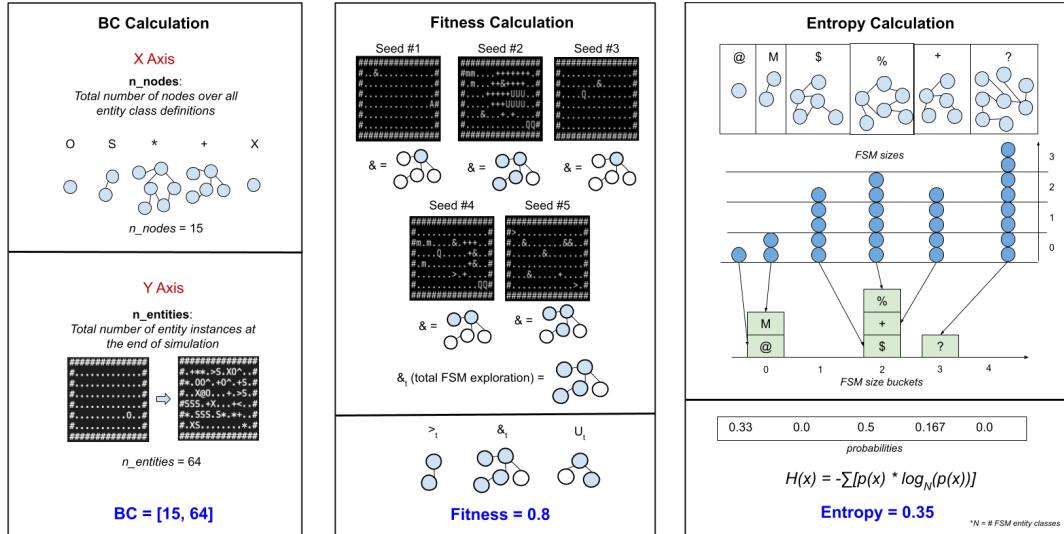


Figure 11.7: Diagram of the calculations for MAP-Elites BCs, fitness values, and entropy value

for any given fortress.

11.3.2 Experiment setup

Using the fitness function, Amorphous Fortress (AF) mutation operators, and AF initialization schemes described above, we use standard MAP-Elites to iterate on a population of AF samples, evaluating them to determine fitness measures and characteristics, sorting them into their respective MAP-Elites grid cells, re-sampling from the grid to create the new population of samples and repeating for a set number of generations. In this experiment, we use a population sample size of 10 fortresses each with 15 entity class definitions, evaluate these fortresses across 10 randomly chosen seeds, and evolve the populations for 10,000 generations. We use a small number of samples for the population with significantly longer generation time to encourage the algorithm to explore more of the cells of the MAP-Elites grid. The initial population of fortresses is made up of FSM definitions with only a single uniformly random action node. We use two combinations of behavior characteristics in this experiment to explore the QD space of the evolved fortresses as well as the emergent behaviors from the elite fortresses saved.

11.3.3 Results

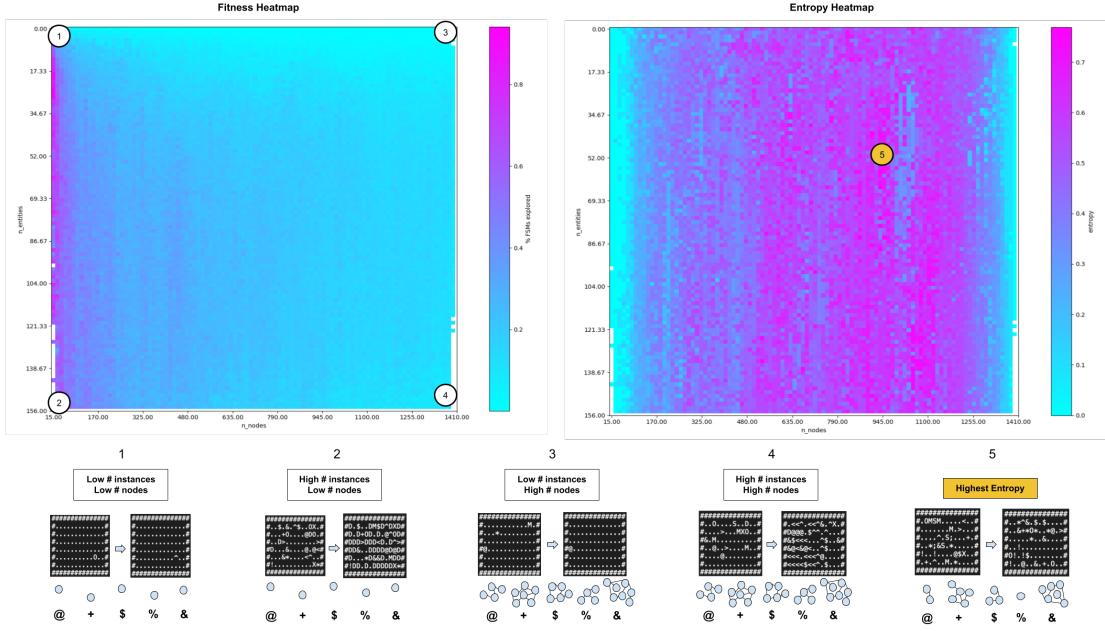


Figure 11.8: Results of the AF-QD experiment: exploring **number of instances** at the end of the simulation vs. **total number of nodes**. The graph on the left shows the heatmap with respect to the fitness measurement—the proportion of FSMs explored in the fortresses. The graph on the right shows the same archive of fortresses measured with the heatmap showing the measure of FSM entropy sizes.

11.3.3.1 Map Coverage

Figure 11.8 shows the heatmap for the MAP-Elites grid with the fitness for this experiment measuring the percentage of the entity class FSM visited. Nearly every possible MAP-Elites grid cell was filled for this experiment. Fortresses with fewer total nodes had the highest fitness values in the grid—most likely because it becomes increasingly challenging to explore different nodes in larger graphs. Predictably, many cells that contained fortresses with high instance numbers were terminated from overpopulation. The number of total entity instances does not seem to be limited by the total number of nodes, with fortress environments that lead to either extremes of near extinction (0-1 instances) or overpopulation (156 instances.)

After the QD search process, we additionally evaluate the diversity of the entity classes within each fortress, measured as the entropy over the distribution of entity

class FSM sizes. We note that high entropy—exhibited in a large swatch of the network with a medium-high number of nodes—corresponds to sets of entities with variably sizes FSMs. When such individuals are fit (and FSMs have few ineffective nodes/edges), we can guarantee that different types of entities will exhibit diverse behavior. In this case, a hypothetical learning agent will be forced to adapt to a diversity of behavior profiles, increasing the richness of its task.

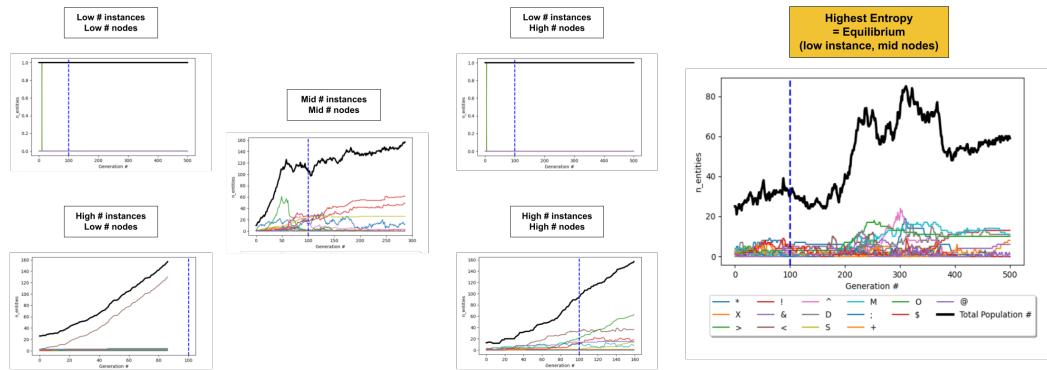


Figure 11.9: **Growth of the population** number for each entity class in the MAP-Elites archive. The vertical dashed blue line is the number of iterations the fortress was simulated for during evolution. The cell with the highest entropy of entity class definitions achieves an ecological equilibrium. This behavior is distinct from the extremes of the archive, which either overpopulate or rarely add any additional entities.

11.3.3.2 Fortress Population Equilibrium

We observed an interesting phenomena within the MAP-Elites cells concerning the population numbers of each of the entities. Ideally, we were aiming to find fortresses with balanced interactions between entities. Figure 11.9 shows the growth of the population in the fortresses found in the ending axis of the MAP-Elites archive. The cells found at the extreme points of the archive (i.e. cells with lowest and highest possible behavior characteristics) exhibited uncooperative behavior between the instances. Fortresses with lower instance numbers refused to populate and caused a stagnation in the fortress. Conversely, fortresses with higher instance numbers quickly overpopulated. However, fortress individuals found in the middle of the archive had more “equilibrium” and cooperation. The fortresses in the middle

of the archive achieved much more diversity in terms of the entity class population; some entity classes having a sudden growth in instance number before dying off, while others slowly expanded their presence over time. This “equilibrium” was most noticeable in the fortress individual that demonstrated the highest entropy between entity class FSM sizes. This fortress showed a near perfect balance between all entity classes; neither dominating nor diminishing in numbers. The entities found in this fortress find a “harmony” of co-existence where the ecosystem does not find itself in danger of overpopulation nor extinction. From this, we can conclude that having a diversity of entity class sizes leads to better balance of entity populations and allows for more exploration of co-operative class behaviors.

11.4 Summary

This chapter describes the Amorphous Fortress framework – a microscopic and fully transparent zero-player simulation environment made up of finite state machine agents. The potential of the system is explored through two evolutionary experiments: a hillclimber experiment intended to generate emergent behaviors from the multi-agent interaction and a quality diversity experiment intended to find variety in agent behavior across environments. With the hill-climber experiment, when defined with a fitness function intended to generate large, explored FSM agents, a symbiotic collaborative relationship emerged from the evolved entity interactions. With the quality diversity experiment, behaviors more closely resembling real-world ecosystems such as extinction, overpopulation, competition, and varying growths and declines of classes over time that reached an equilibrium state were discovered in the saved archive of fortresses. This open-source framework lays the foundation for more research focusing on explainable AI, multi-agent learning, and open-ended simulation studies.

11.5 Reflection

I made this system with the intent to design artificial AI environments that were also abstract; like watching bacteria in a petri dish. As I developed it with my collaborators, we slowly started to see the potential of this system for general and abstract AI environment design – similar to other frameworks such as GVG-AI, Sturgeon, and Griddly. There had been work for designing games and simulation environments for AI agents to interact and learn in. But there was a lack of work for framework design for multi-agent systems exhibiting different kinds of game behaviors and emergent behaviors. I also wanted a system that was capable of producing weird interactions like the ones players would discover in open-ended games like Breath of the Wild, Sims, and Dwarf Fortress. I also wanted a framework with the simplest AI definitions possible for explainability and scalability – hence why the system was written using FSM.

I had fun developing even the test environments for this project and seeing how the agents would interact under the randomized circumstances. Even with the abstract generated environments, watching the amorphous agents change and evolve in the simulation was intriguing. This project led into the Amorphous Fortress Online work (described in the following chapter – Chapter 11.) Here I believed that by opening the framework to a multi-user interface would allow even more open-ended environments with more emergent AI behaviors to be found.

11.6 Takeaways

- Finite state machines, while simple, can be extremely versatile and produce emergent AI behaviors
- Simple framework design can be unexpectedly open-ended
- Multi-agent AI design is easy but can make for complex behaviors similar to that of video games

- Modularity and scalability are key factors for effective framework design
- Quality diversity algorithms applied to abstract frameworks lead to a large diversity of environments

Chapter 12

Collaborative Abstract Game and AI Environment Design

In the research works described in the previous chapters, AI-assisted content generation has previously only been applied to familiar domains and games and limited to creating small artefacts such as images, levels, structures, or sprites. The Amorphous Fortress framework experiments looked to see if it was possible to generate new domains that still offered diversity and variety in the mechanics and interactions that occurred. The abstraction of the fortress environments made helped automate the evolution process, but like the Baba is Y'all system, it could benefit from user involvement to discover more domains. Like Baba is Y'all to Keke AI Competition and the Aesthetic Bot to \$5 model system, this collected data could be used to further more AI research with crowd-sourced labeling and collaborative content submissions.

This chapter describes Amorphous Fortress Online system – an collaborative, interactive web application that allows users to create, share, and modify abstract simulation environments using the same framework. This system expands on the lessons of the content generation and mixed-initiative systems described in the previous chapters to create a framework that allows for free-range user collaboration and build the groundwork for versatile new content generation and recommendation

AI models. The website features multiple interactive editors and accessible settings to view the multi-agent interactions directly from the browser. This system serves to provide a database of thematically diverse AI and game environments that use the emergent behaviors of simple AI agents that go beyond the definitions of familiar game domains.

12.1 System Description

Amorphous Fortress Online follows a similar system design to the Baba is Y'all 2.0 web application – facilitating user content creation, editing, and sharing (see Chapter 4). The following subsections detail the module components of system.

12.1.1 Framework

The framework for the Amorphous Fortress Online system utilizes a Javascript port of the Amorphous Fortress framework described in Chapter 11. This port expands the Python framework by adding three player action nodes to the entity FSM node definition list. These nodes depend on user input to enact the action – allowing a user-controlled player option for the fortresses to turn them into game environments. These three nodes *player_move*, *player_push*, and *player_move_wall* are based on their autonomous action nodes *move*, *push*, *move_wall* respectively. However, instead of using random movements, they use 5 directional user inputs: up, down, left, right, and a skip turn action which keeps the entity idle.

12.1.2 Entity Editor

When designing a new fortress or editing a previously submitted fortress in Amorphous Fortress Online, users are initially taken to the entity editor screen – referenced in Figure 12.1. Entities are designed and edited with an interactive drag-and-drop and dropdown menu system. The nodes and edges of an entity's FSM are visualized and organized in a canvas editor (A). Users can add new nodes

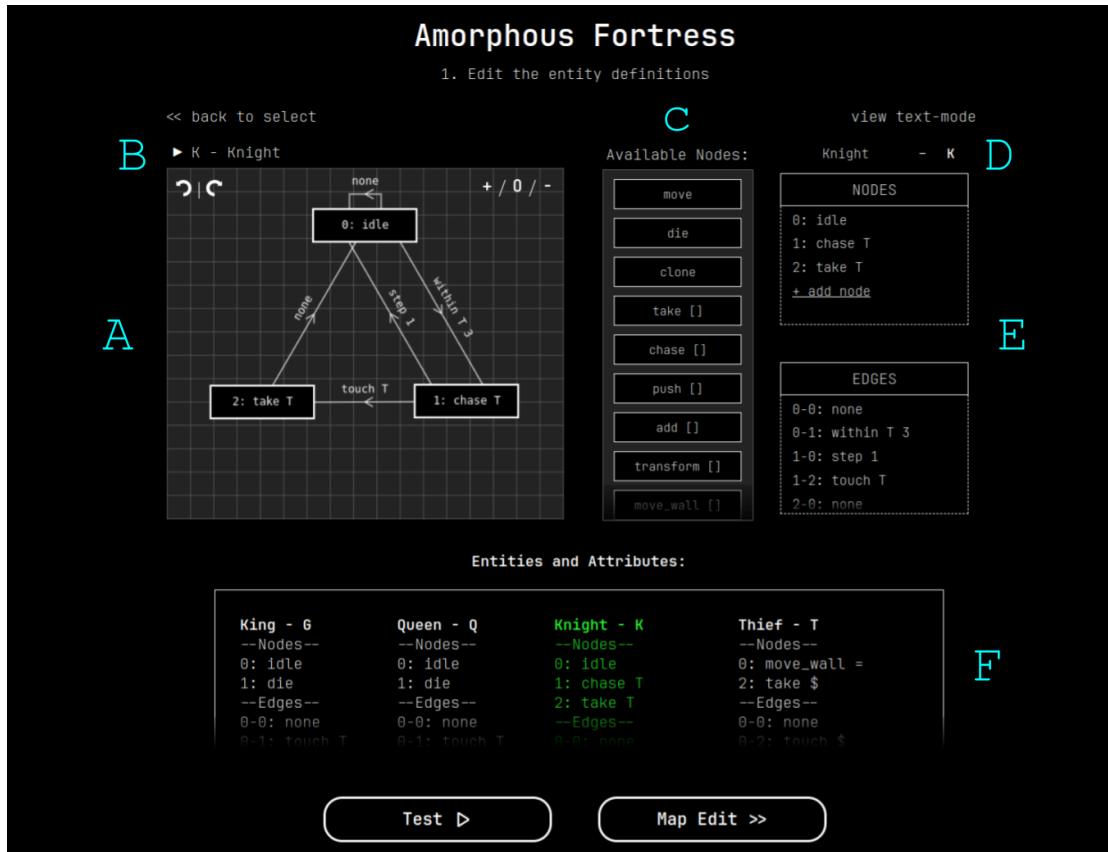


Figure 12.1: The entity editor screen for Amorphous Fortress Online labeled as follows: A) FSM canvas visual editor; B) Entity selection dropdown; C) Canvas node list; D) Name and character input area; E) Entity node and edge list edit area; F) Fortress entity list

to the canvas by dragging nodes from the available node list section (C.) In the canvas, users can edit a node's value or delete it by right clicking on its FSM box and selecting through its dropdown menu screen. Edges can be added by selecting the starting node box and then clicking on its connecting end node. The edge condition can also be changed in a similar fashion as the nodes by right clicking on it. Alternatively, users can use list editor menu (E) to make changes directly based on value and add and delete entity FSM elements.

The user can quickly switch between the editing the current entity definitions through a dropdown menu above the visual FSM editor or by clicking on the entity's full definition box from the entity list area (F.). Users can input whatever name or character representation they would like their entity to have in the fortress (D.) Entities in the fortress are added or deleted with a dropdown menu (B) above

the canvas editor.

Each drop down menu has a limited selection of options for the user to choose from when designing their entity. The options for the node and edge values are limited to only the characters that have already been defined in fortress. For example, in a fortress with the entity character representations *M*, *\$*, and *+* for the action node *push* only the options *push M*, *push \$*, and *push +* would be shown.

12.1.3 Fortress Editor

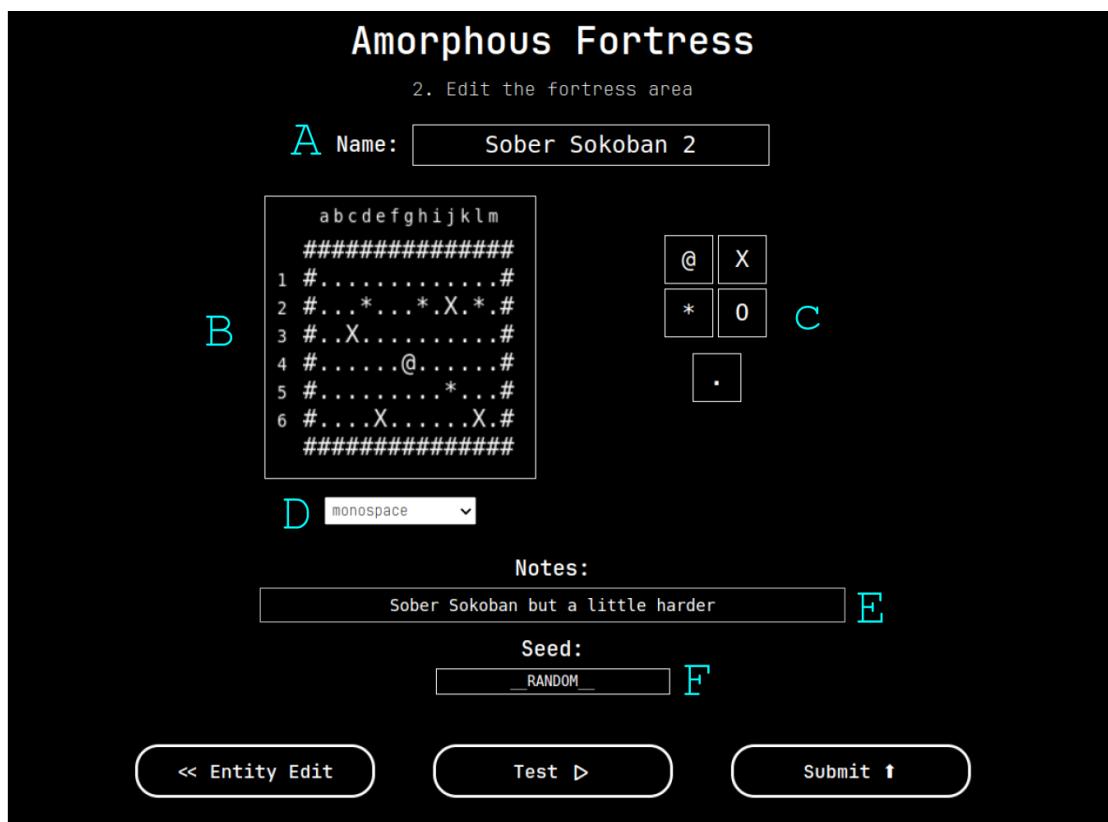


Figure 12.2: The fortress editor screen for Amorphous Fortress Online labeled as follows: A) Fortress name input area; B) Fortress placement canvas C) Entity palette; D) Font selection; E) Notes input area; F) Seed input area

After editing their entities, users can use the fortress editor to then place them in the fortress environment – referenced in Figure 12.2. The fortress editor features a canvas window (B) with the fortress layout in a 14 by 6 grid space area. Users select the entity based on their character representation from a palette (C) on the right and place them onto the fortress map on the left of the screen. The font

of the characters and fortress ASCII representation can be changed to 6 different options through a dropdown selection (D.) The fortress editor also includes input areas for users to name their fortress (A), provide notes or a description about the fortress (E), and specify a seed (F) to initialize the fortress with when a user plays it. If the seed is specified with the value “*_RANDOM_*” then the fortress will initialize with a random seed every time the simulation is reset.

12.1.4 Text Editor

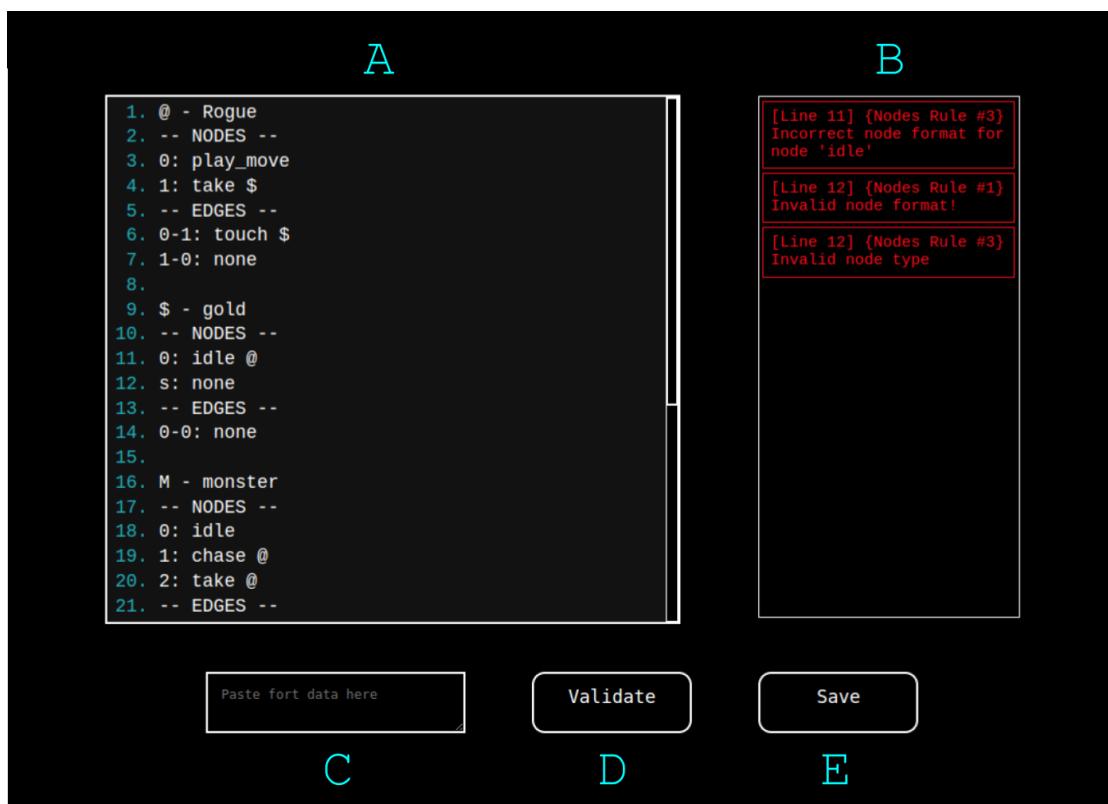


Figure 12.3: The text editor screen for Amorphous Fortress Online labeled as follows: A) Fortress text editor area; B) Compiler console; C) Raw text paste area; D) Validation check button; E) Save fortress button

For advanced users, Amorphous Fortress Online also includes a text editor – see Figure 12.3 – accessible from the entity editor screen to allow users to directly edit the fortress definition. The text editor acts as an alternative to both the interactive entity and fortress editors. This particular editor is based off of the original Amorphous Fortress fortress definition files. Users can directly specify the

entity FSM definitions and initial fortress layout with the text area interface (A.) The text editor features a custom made compiler for the Amorphous Fortress engine that checks for invalid nodes or edges, such as unknown node definitions, invalid FSM parameters, invalid characters, or invalid fortress layouts such as unknown characters or incorrect fortress dimensions. To save the fortress (E) to the entity and fortress GUI editors, users must first validate (D) their input text through the compiler. This is to ensure that the text version of the fortress definition can be fully interpreted by the engine without issue. The compiler returns the type of error found from the text editor as well as the line number in a console (B) to the right of the editor for easier debugging for the user. Users can also paste entire fortress definitions into a smaller text area (C) and edit it line by line.

12.1.5 Play Screen

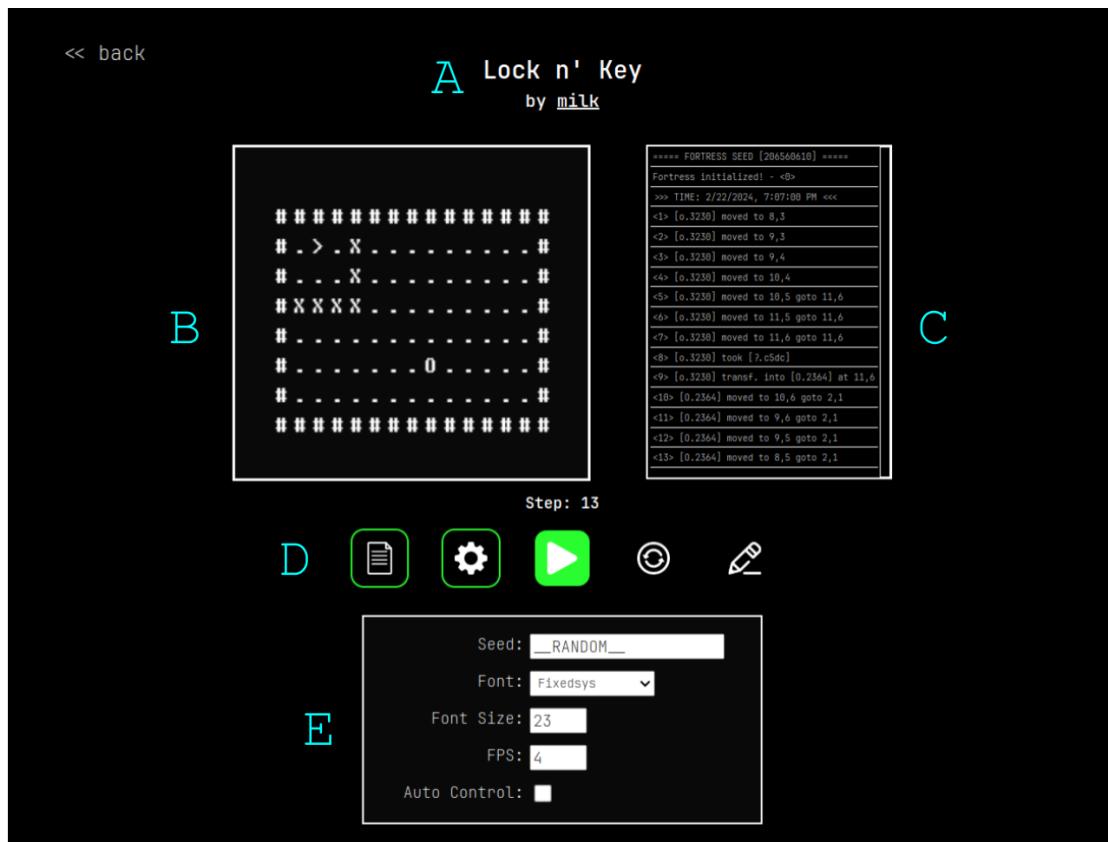


Figure 12.4: The fortress play screen for Amorphous Fortress Online labeled as follows: A) Name and author of the fortress; B) Main fortress view area; C) Fortress simulation log; D) Simulation menu bar; E) Play screen settings area

From both the entity editor and fortress editor screen, users can test and simulate their fortresses before submitting to the Amorphous Fortress Online database in the play screen (shown in Figure 12.4). The play screen details the name and author of the current fortress (A) and shows the fortress state during simulation in the main fortress view window (B) starting with its initial entity placement state. In the menu bar (D) under the main fortress view window, users can start, pause, or reset the simulation by pressing the play button or reset buttons. The play screen also includes a settings area (E) with many user adjustable options for altering the display or the fortress's multi-agent simulation settings. Here, the seed for random agent behavior can be modified to recreate specific agent interactions that occur in the fortress. The font type – with 6 different font options – and size of the font displaying the characters in the fortress can be changed by the user. The speed of the simulation can also be adjusted in this settings area. To the right of the fortress view window, the user can display the log of the simulation (C) maintained by the fortress engine. Here every interaction that occurs with the entities is recorded at each timestep.

For fortresses that include player action nodes, the user can control entities highlighted with a cyan background and specify their input directions with 3 different keyboard control schemes (see Table 12.1). Players can also start and stop the simulation with the P key and reset the simulation with the R key. The settings area features a toggle option to turn off player control and will instead default the movement of player controlled entities in the simulation to automated random directional key presses.

Control Scheme	Up	Down	Left	Right	Skip
Arrow Keys	↑	↓	←	→	Space
WASD	W	S	A	D	F
Vi Keys	K	J	H	L	;

Table 12.1: Control scheme options for player controlled entities

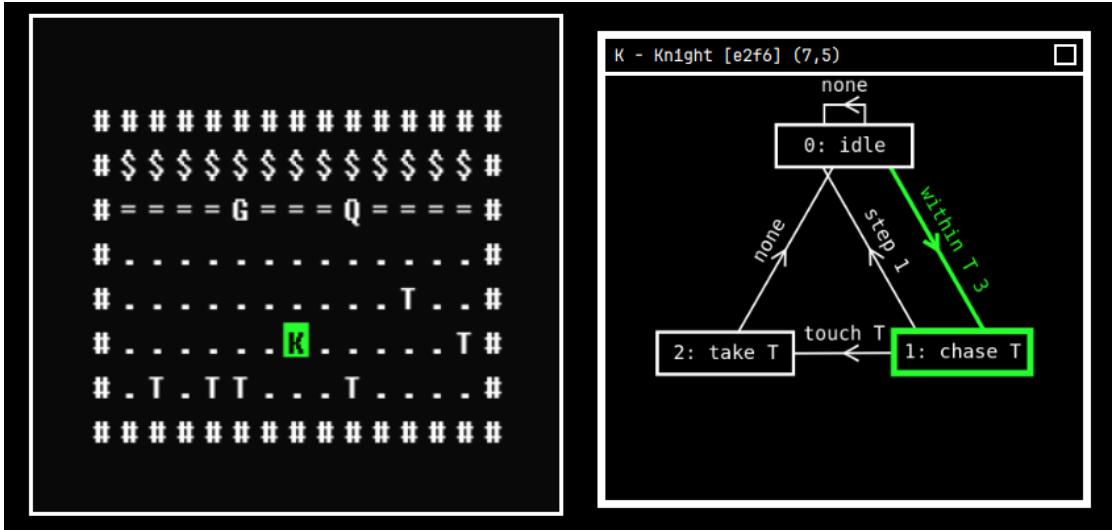


Figure 12.5: The X-Ray feature (right) highlighting the Knight entity K (left) in the Thief Castle fortress (#36.) The green outlined FSM components indicate active or recently activated behavior logic. Currently its 1: *chase T* node is active and because of the edge transition from the 0-1: *within 3 T* edge.

12.1.6 X-Ray Feature

The play screen features a pop-up and movable X-ray window (see Figure 12.5) to view an entity’s current FSM state during runtime. This window shows the entity’s FSM definition as organized by the user in the entity editor screen which is saved with the fortress data upon submission. Active nodes and most recently transitioned edges are highlighted in green in the X-ray window and it is constantly updated with each step in the simulation. Entities currently being viewed in the X-ray appear highlighted in green in the main view window. This feature allows for more explainability with the agent behaviors and helps designers debug their agents during fortress creation.

12.1.7 Main Screen

Once a user finishes editing their fortress, it is uploaded from the fortress editor to the Amorphous Fortress Online database and shown on the main landing page of the website (Figure 12.6). The main page shows the most 120 recent fortresses submitted to the database with their fortress name, author, and initial fortress layout. Hovering over a fortress in the matrix list allows users to see the defined

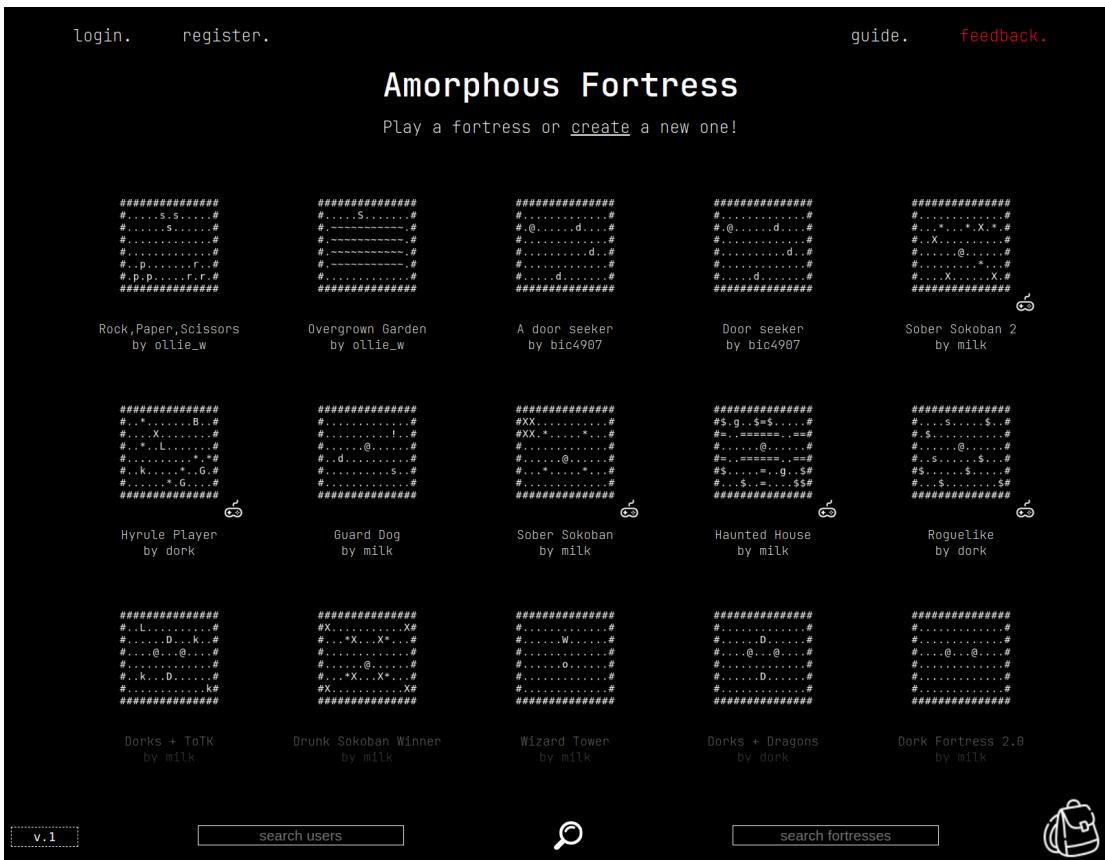


Figure 12.6: Main screen fortress selection

entity names in the fortress. Fortresses that include player action nodes include an outlined game controller next to their fortress rendering to indicate to users that they can be user controlled.

When a user clicks on the fortress, they can view the full information, including the entity definitions, notes description about the fortress, and a link to the parent fortress ID if the fortress was remixed and edited by the author from a previously submitted fortress. An example fortress view is shown in Figure 12.7. From this information view, users can either select to play the fortress themselves in the play screen window or edit and remix the fortress to make a new submission.

The main screen features two search bars at the bottom of the page to allow users to find specific fortresses submitted to the database. Users can search for fortresses by username, fortress name, or both. The most recently submitted fortresses matching the search criteria are returned. Users can register a username in order to claim authorship to their fortresses with a login authentication system.

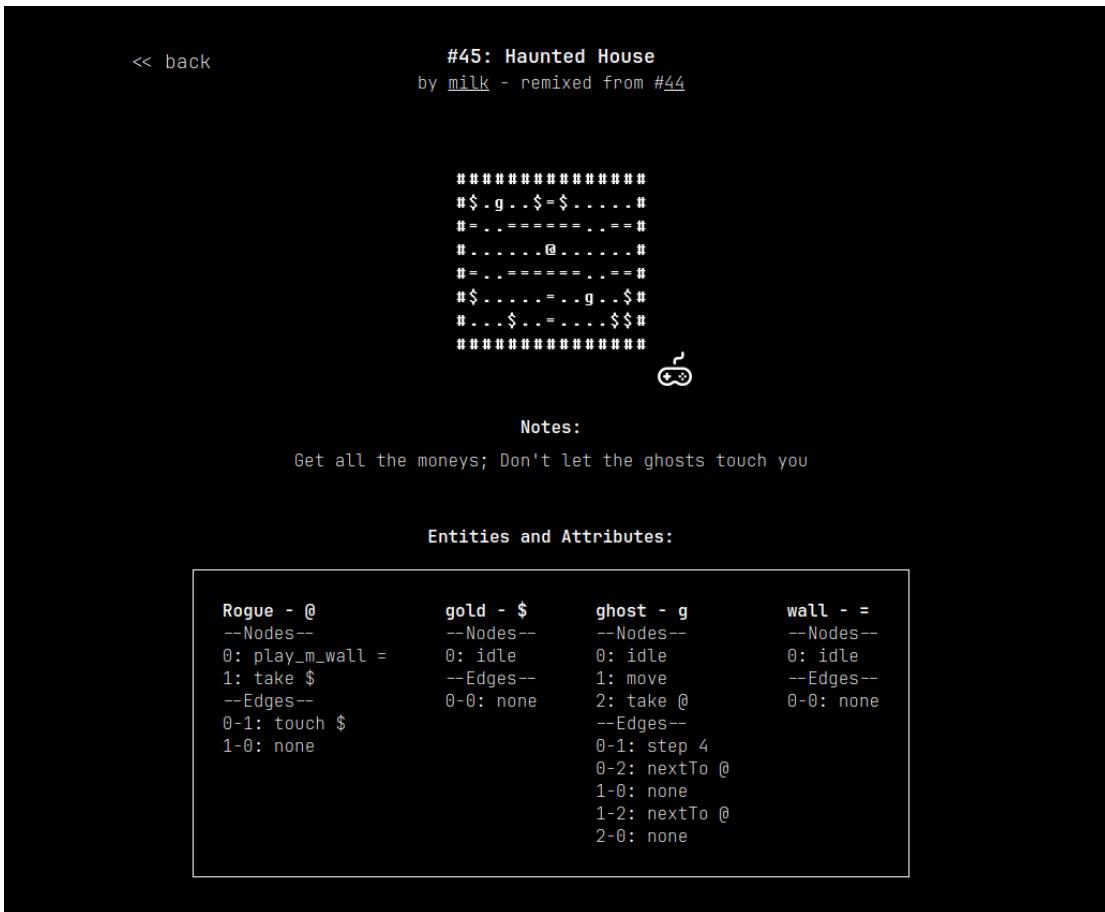


Figure 12.7: Fortress info viewer

For password recovery, there is also an optional input to provide an email address with the username registration. Fortresses submitted anonymously without a user login will be authored by the default "dork" user account.

12.1.7.1 Backpack Feature

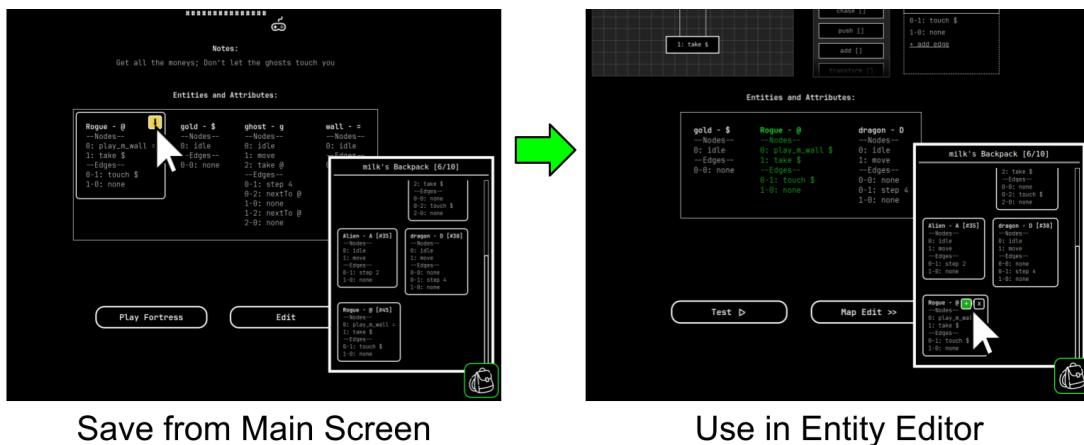


Figure 12.8: When logged in, a user can save an entity from one fortress to use in a new fortress with the backpack feature.

The backpack feature of Amorphous Fortress Online (demonstrated in Figure 12.8) allows users to save entity characters from submitted fortresses to their user account. By clicking on an entity definition from the fortress view window of the main page, users can save entities to their "backpack" storage – a pop-up window found at the bottom of the screen. Up to 10 entities can be saved to the user's "backpack" storage. These saved entities can then be added to a new fortress in the entity editor. If a backpack entity has a reference to a character not found in the fortress, it will be randomly replaced with another entity. For example, if a backpack entity with the character representation, *A*, contains the node *take \$* but the fortress only contains entities with the characters *&*, *M*, and *+*, then the node will be changed on addition to the fortress to either *take &*, *take M*, *take +*, or *take A*. This feature facilitates collaboration and remixing across environment design and the usage trends of different entity definitions and roles can be studied across fortress development.

12.1.8 User Guide

A user guide¹ is provided for Amorphous Fortress Online as a tutorial, reference manual, and explanation for elements of the engine. The guide features sections on the Amorphous Fortress engine itself, the main screen, entity, fortress, and text editors, and the play screen. On each page of the guide are demonstrative GIFs to show the player how to interact with the Amorphous Fortress Online interface using examples from previously submitted fortresses.

12.2 Short Study

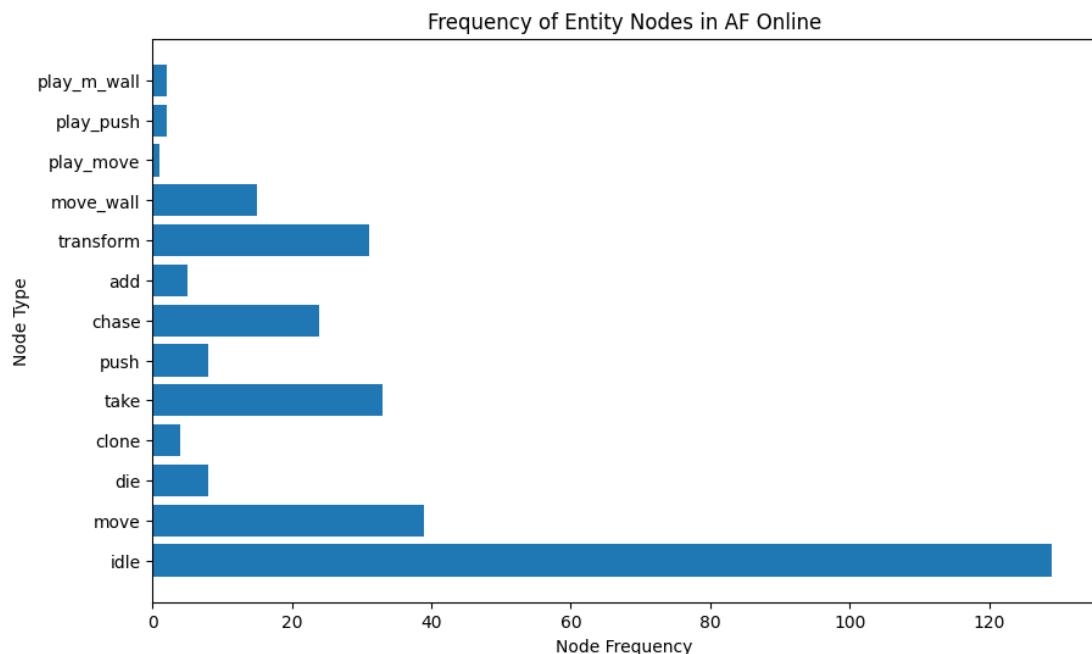


Figure 12.9: Frequency of entity action nodes

We released the Amorphous Fortress Online v1 site on February 20, 2024. The website was promoted on various social media sites including X (formerly known as Twitter), BlueSky, LinkedIn, Facebook, and Reddit.

While we didn't receive many new fortress submissions, we collected a total of 174 plays from the fortresses. Six different users submitted a total of 33 fortresses to the website. Unfortunately, a majority of these fortresses come from the authors

¹<http://amorphous-fortress.xyz/wiki/>

– only five additional fortresses were submitted after the release of the site. This stark difference of fortress plays vs. fortress creation can possibly attribute to a large learning curve to using the system. An unfamiliarity with the engine as opposed to more familiar domains and games that have online editors such as *Free Rider* and *Baba is You* may have also attributed to the lack of submissions.

Outside of the author’s submission of 21 fortresses, users submitted an average of two fortresses. There were a total of 143 entities made in the website and a total of 298 nodes. Figure 12.9 shows the frequency of each node type in the system. Interestingly, about 34.38% of submitted fortresses were remixed, with the longest lineage of fortresses having a depth of 4 fortresses. Five fortresses with a *play_move*, *play_push*, or *play_move_wall* were made.

12.3 Future Work

Amorphous Fortress Online is intended to be a system to lay the foundation for future generative and recommendation game AI studies by utilizing the user submitted data. With this data, we would like to accentuate the environment design process and make the system more engaging by introducing mixed-initiative and collaborative AI systems trained from on-going submitted fortress data.

12.3.1 Mixed-Initiative Fortress Design

The lack of fortress submissions is one of the most pertinent areas to address for future work. To remedy the intimidation factor of using the system and trying to make a new fortress from scratch or from a remix of another user’s fortress, we will introduce the fortress evolver system used in the design of the previous iterations of the system [23, 41]. This generator will allow users to have a starting point and edit the fortress towards their liking – much like the creative process in the Baba is Y’all system. This mixed-initiative interface will also encourage and develop an effective feedback system between the AI generator and the human

designer to create novel fortress environments for the system.

12.3.2 Entity Recommendation

Designing and constructing thematically and semantic relevant entities for a fortress is a tedious process. This process is slightly alleviated with the backpack feature, however we would like to incorporate the entity and fortress-focused evolutionary components of the previous Amorphous Fortress works in the online version of the system to create a human-AI collaborative pipeline. Using the quality diversity methodology from previous work on the Amorphous Fortress [41], we will build a recommendation system for the entity editor. This system will suggest thematically relevant entities to the user that also enhance the multi-agent interactions within the fortress. This recommendation system will be modeled after the variance of roles entities and NPCs have in video games, ranging from decoration entities and items to simple background characters that act as NPCs to more complex entities such as enemy characters. The complexity of these entity FSM definitions would complement the user’s current entities in the fortress and guide the fortress towards emergent behavior for the environment. This work is inspired by Machado’s work with sprite recommendations from the Pitako system [79].

12.3.3 Sprite Generation

The visuals of the Amorphous Fortress Engine are abstract and require more creative imagination from the user to interpret the actions and behaviors of the entities in the fortress. To make the system more visually appealing and engaging, the next iteration of Amorphous Fortress Online will feature a sprite editor to allow users to replace the entity ASCII character definitions with small 8x8 pixel 4-color sprites. The minimalist sprite designs is inspired by that of other easy-to-use microgame engines like PICO-8 and Bitsy. After users submit labeled text-sprite associations to their entities in the fortress, we will use this data to train a small

generator model similar to architecture of the \$5 model [83] to recommend and generate new sprites for the fortresses. This project will create a new database of labeled small scale images that will also continuously be updated based on site usage.

12.3.4 Narrative Generation

While the interactions of the agents in the fortress can be interpreted by a trained human eye – either from viewing these logs or from watching the visual representation of the entities move throughout the fortress – the abstract representation of each entity limits interpretable narratives or meaning from the events that occur in the fortress. Communities for simulation games such as Dwarf Fortress and The Sims have developed on forums and other sites that allow players to develop their own narrative from the events that occur in their own game environments; even when they are not explicitly stated or shown in the code or logs of the game itself. These emergent narratives demonstrate the versatility of the simulation itself, but also allow the players and users to connect to the artificial entities and give them more depth and explainability. A textual generator such as a transformer could be used to annotate the logs and create generative stories on the Amorphous Fortress environment similar to how users form narratives for other simulation games. A user study would be conducted asking participants to compare human annotated narratives against generated narratives for aesthetic quality on how accurate the narrative is reflected in the simulation.

12.4 Summary

This chapter describes the system design of Amorphous Fortress Online – an online simulation / game design system based around community-centric user generated environment creation. We describe the features and interfaces available to users to engage with the site and submit open-ended and thematically diverse

games and multi-agent environments. The growth and addition of new fortresses will set the groundwork to build a large dataset that can be used for future AI research in agent training and generative models.

12.5 Reflection

I was pretty disappointed in the results of the user study after release of the system. Honestly, this had the most disappointing user engagement of all of the systems in the thesis. However, the lack of engagement also gave me some insights into what the system was missing and how I could improve as a designer of AI collaboration tools. When I developed the Amorphous Fortress framework and developed the website for Amorphous Fortress Online, I was expecting to see people use the framework to make open-ended but thematically diverse environments. The examples and promotions I did for the website – boasting that one could re-make their favorite game mechanics and characters from movies – weren’t enough to engage users for the time needed to learn how to use the system, develop in it, and remix other environments submitted to the website. I ended up as the main contributor to the website.

The main focus of the thesis was on AI collaborative content generation, and this system is the only one included in the thesis that doesn’t incorporate an assistive AI system. Ironically, and further proving the point of the thesis claim, I believe AI assistance would have engaged users more and made the system less daunting to learn. If an interface such as a text-to-environment system was implemented that also allowed users to manually edit the fortress as they needed, I think more fortresses would have been designed as I had hoped people would interact with the website.

But maybe I was so focused on developing a system that I wanted, that I thought others would use in the way that I would use, that I had lost sight of whether or not users would actually want to have this kind system or framework.

My disappointment stemmed from my high expectations of the users I thought that would engage with it. I wanted to design an online system similar to the content sharing websites and games I had played and explored such as the FreeRider Online, LittleBigPlanet, and the Museum of ZZT. But the Amorphous Fortress Online couldn't quite reach the intended audience or get the level of user engagement that I had hoped for.

I don't see the system overall as a failure, but I believe there were steps I could have taken or features I could have implemented before its official release that would have been better for the system long-term. I take this final system of my thesis as a lesson for myself in system design; how to better align the interfaces I produce to user needs. But I have a stronger belief in AI assisting tools; they could be beneficial collaborators when put in the right contexts and when they are actually needed by users to help them with content generation and creative development.

12.6 Takeaways

- AI assistance could help for content generation in an unfamiliar domain
- Personal aesthetic preference may not match general audience aesthetic preference
- Complex systems need thorough tutorialization and guidance for better user experience and engagement
- Unfamiliar or novel designed domains are less likely to engage to users
- It's more accessible and easier to play or evaluate generated content than it is to manually design new content
- What a designer wants may not reflect what users would want

- Lack of user testing from multiple background and demographics for a designer interface can be detrimental to the success of a system
- Developers should study implicit user trends (i.e. feature usage not explicitly described or reported by users) and design and adapt their systems around the change in trends
- Engaging UI design is key for creating an interactive online system

Chapter 13

Conclusions

This final chapter summarizes the 12 works presented in the thesis, provides some reflections, and proposes some next steps for this research direction. The beginning of the thesis stated this claim: *when applied to creative domains, online systems allow users to more effectively design, share, and evaluate content with AI tools.* The 10 previous chapters look to uphold this claim with their contributions: collaboratively generating diverse game content online, designing online systems for open-ended environments and problems, utilizing online data to further AI research, gathering feedback from designers for future AI-assisted tool development.

13.1 Online AI Systems

The results presented in Chapter 3’s system – Mech-Elites – highlighted how automated systems on their own are capable of generating a large diversity of content given the right constraints and objectives, but are not able to fully explore the space and still miss particular content artefacts. Chapter 4’s system – Baba is Y’all – attempts to remedy this limitation by introducing users into the generation process. Here, the online website with the AI generation and recommendation back-end system facilitate user-AI collaboration to encourage the development of novel levels and find the missing game level artefacts that could not be generated in the previous system. However, this system and the user study conducted on

it exposed deeper challenges from user-AI systems as a whole: developing and improving AI systems that train on human-AI created content, evaluating the quality of generated content quickly, receiving user evaluations in a way that doesn't cause user fatigue, encouraging the use of AI systems in creative design, and understanding creative motivations and utility of AI collaboration. This project ignited the start of many more research projects that looked to address these challenges.

Chapter 5 described the Keke AI Competition – an AI competition that recruited participants to design new automated solvers for the Baba is You game. This competition encouraged users to train their solvers on the diverse level data from the Baba is Y'all system. The results of the competition showed the effectiveness and utility of the collected online data from the website, but also showed that developing effective solvers still proved challenging.

Chapter 6 looked to simultaneously address the lack of user evaluations on the Baba is Y'all levels and the lack of a well-defined metric for evaluating aesthetically pleasing content. This chapter described the results and impact of the Aesthetic Bot system – an online Twitter bot that collected user evaluations on AI and human designed visual game maps that trained on the votes users provided on the Twitter posts. This work was successful in getting hundreds of user evaluations and user-made maps but demonstrated that aesthetic evaluation is an open-ended, subjective metric that is difficult to train AI systems to learn. Chapter 7 took the map data collected from the Aesthetic Bot system, created descriptive labels for the maps with crowd-sourced effort, and trained the micro text-to-image system called the \$5 Model. This system showed the utility and benefits of a small-scale generative model as applied to game content generation.

13.2 The User and the AI

Based on the Baba is Y'all system, a key factor determinate of an AI-collaboration system's effectiveness depended on the user's willingness and motivation to use it. The following research chapters looked to discover how an AI's output could align with the user's creative intent to encourage more user-AI creative collaboration.

Chapter 8 describes the Minecraft House Evolver system – a 3D voxel structure generator that limits the user's creative input for the content generation process. The user study responses showed that while the system was effective for quickly generating content, the lack of control it caused a disparagement in user's creative alignment and the system's output. However, the users expressed interest in AI-collaboration systems in general. With the AI systems, unpredictability in output, they were also able to create content they originally had not intended to, but led them to new creative discoveries.

Chapter 9 explores more human-AI content creation and output by examining content generation for abstract game design. Another user study was conducted to gather human evaluations on game feature recommendations based on a text input prompt. While the human-made feature recommendations mostly outperformed the AI generated features in an anonymous comparison, the difference in votes was close enough to suggest that AI-generated content could rival human-made content given enough training and fine-tuning to the user's preferences.

Chapter 10 applied AI-generated content directly to game design in a formal study with the online, accessible LLM, ChatGPT, as the game designer. Here, 3 games – a baseline game with no features, a human-designed game, and a ChatGPT designed game – were created for 3 different game genres and evaluated by users. With the evaluations, the AI-designed ChatGPT games was voted higher than the baseline games across all genres, but was still subpar compared to the pure human designed games. From a developer standpoint, the creative intent and direction of the game was difficult in some areas to communicate to the large-language model. However, many interesting and novel features emerged from the ChatGPT

suggestions that were not otherwise considered by the human designers.

Finally, the Amorphous Fortress framework – as described in Chapter 11 – and its user interactive application in the Amorphous Fortress Online system – detailed in Chapter 12 – showcased the data generation and training potential for an online collaborative user system based around an open-ended domain. Most notably, it highlighted the need for an AI generation or recommendation tool when users are faced with an unfamiliar domain and system.

13.3 Future Work

As mentioned in the previous chapter, many of the future directions for this research stem from the Amorphous Fortress Online system. It acts as a spiritual successor to the Baba is Y'all work – the project that led to the development of many of the works that make up this thesis. From the Amorphous Fortress Online system, I plan to create the following research projects: an integrated mixed-initiative fortress generator – to speed up fortress design and aid user learning with the framework; an entity recommendation system – to demonstrate semantic and functional similarities in the agent definitions; a sprite generator – a system intended to be an applicable version of the \$5 model and to make the fortresses more visually engaging; and a narrative generator – to create stories and situations from the fortress and form an attempt at translating AI behaviors to human language.

I plan to expand on previous works such as the Aesthetic Bot and the Keke AI Competition that are unrelated to the Amorphous Fortress framework. The Keke AI Competition will be ported to Python to attract more participants. The Aesthetic Bot evaluations and training will be applied to a new online system to teach specific aesthetics to level design in a similar mixed initiative fashion to that of RLBrush[38] and Lode Encoder[10]. The abstract game feature recommendation and generation tool will be combined with the Ink Splotch experiment to study

the effects of AI generated game design on output and user engagement in an accessible and intuitive online interface. I would also like to make a system that generates multiple, diverse, themed level sections (like to stitch together in a large world map similar to that of Link’s Awakening). The future projects listed here are in no way encompass everything that I’d like to explore in this field, but I believe the limitations and successes of previous projects act as great starting points.

13.4 Reflection

While AI systems are still a long way off from becoming common-place collaborators that can work as effectively as human-collaborators with creative design, these systems provided many insights on how to reach this goal. By making these interfaces online, AI systems are allowed to be more accessible and have real-world application for users and creators. Online systems allow multiple users to interact with the same AI system so that it learns a more general and diverse spread of creative information. The data created from these systems – both AI, human, and human-AI made – can also be used to train more AI systems that can produce novel content.

Future work will look to improve AI systems as a whole, but will be done so with online interfaces. While this research work addressed the challenges mentioned before, it exposed even more limitations that AI systems still face when used in creative domains. More user studies and the development of online AI interfaces that facilitate multi-user and user-AI collaboration will certainly provide more insight and ideally solutions to these challenges. At the end of the day, even if AI system will never be the perfect collaborators for humans, they can certainly help to make new and interesting content that humans alone could never create.



Figure 13.1: Thanks for reading! Let's make more online AI systems :D

Bibliography

- [1] Alberto Alvarez. 2020. *Exploring the Dynamic Properties of Interaction in Mixed-Initiative Procedural Content Generation*. Ph. D. Dissertation. Malmö universitet.
- [2] Alberto Alvarez, Steve Dahlskog, Jose Font, and Julian Togelius. 2019. Empowering quality diversity in dungeon design with interactive constrained map-elites. In *2019 IEEE Conference on Games (CoG)*. IEEE, 1–8.
- [3] Alberto Alvarez and Jose Font. 2022. TropeTwist: Trope-based Narrative Structure Generation. *arXiv preprint arXiv:2204.09672* (2022).
- [4] Asad Anjum, Yuting Li, Noelle Law, M Charity, and Julian Togelius. 2024. The Ink Splotch Effect: A Case Study on ChatGPT as a Co-Creative Game Designer. In *Proceedings of the 19th International Conference on the Foundations of Digital Games*.
- [5] Anna Anthropy. 2014. *ZZT: Boss Fight Books # 3*. Vol. 3. Boss Fight Books.
- [6] Maren Awiszus, Frederik Schubert, and Bodo Rosenhahn. 2020. TOAD-GAN: Coherent style level generation from a single example. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 16. 10–16.
- [7] Maren Awiszus, Frederik Schubert, and Bodo Rosenhahn. 2021. World-gan: a generative model for minecraft worlds. In *2021 IEEE Conference on Games (CoG)*. IEEE, 1–8.

- [8] Alexander Baldwin, Steve Dahlskog, Jose M Font, and Johan Holmberg. 2017. Mixed-initiative procedural generation of dungeons using game design patterns. In *2017 IEEE conference on computational intelligence and games (CIG)*. IEEE, 25–32.
- [9] Richard Bartle. 1996. Hearts, clubs, diamonds, spades: Players who suit MUDs. *Journal of MUD research* 1, 1 (1996), 19.
- [10] Debosmita Bhaumik, Ahmed Khalifa, and Julian Togelius. 2021. Lode Encoder: AI-constrained co-creativity. In *2021 IEEE Conference on Games (CoG)*. IEEE, 01–08.
- [11] Philip Bontrager, Ahmed Khalifa, Andre Mendes, and Julian Togelius. 2016. Matching games and algorithms for general video game playing. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*.
- [12] Philip Bontrager, Wending Lin, Julian Togelius, and Sebastian Risi. 2018. Deep Interactive Evolution. In *Computational Intelligence in Music, Sound, Art and Design - 7th International Conference, EvoMUSART 2018, Parma, Italy, April 4-6, 2018, Proceedings (Lecture Notes in Computer Science, Vol. 10783)*, Antonios Liapis, Juan Jesús Romero Cardalda, and Anikó Ekárt (Eds.). Springer, 267–282.
- [13] Philip Bontrager, Wending Lin, Julian Togelius, and Sebastian Risi. 2018. Deep interactive evolution. In *International Conference on Computational Intelligence in Music, Sound, Art and Design*. Springer, 267–282.
- [14] Cameron Browne. 2012. Yavalath: Sample chapter from evolutionary game design. *ICGA Journal-International Computer Games Association* 35, 1 (2012), 20.
- [15] Cameron Browne and Frederic Maire. 2010. Evolutionary game design. *IEEE Transactions on Computational Intelligence and AI in Games* 2, 1 (2010), 1–16.

- [16] Eric Butler, Adam M Smith, Yun-En Liu, and Zoran Popovic. 2013. A mixed-initiative tool for designing level progressions in games. In *Proceedings of the 26th annual ACM symposium on User interface software and technology*. 377–386.
- [17] Sara Cardinale, Michael Cook, and Simon Colton. 2022. AI-Driven Sonification of Automatically Designed Games. (2022).
- [18] Michael Cerny Green, Ahmed Khalifa, Gabriella AB Barros, Tiago Machado, Andy Nealen, and Julian Togelius. 2018. AtDELFI: automatically designing legible, full instructions for games. (2018).
- [19] M Charity, Yash Bhartia, Daniel Zhang, Ahmed Khalifa, and Julian Togelius. 2023. A Preliminary Study on a Conceptual Game Feature Generation and Recommendation System. *2023 IEEE Conference on Games (CoG)* (2023).
- [20] M Charity, Isha Dave, Ahmed Khalifa, and Julian Togelius. 2022. Baba is Y'all 2.0: Design and Investigation of a Collaborative Mixed-Initiative System. *IEEE Transactions on Games* (2022).
- [21] M Charity, Michael Cerny Green, Ahmed Khalifa, and Julian Togelius. 2020. Mech-elites: Illuminating the mechanic space of gvg-ai. In *Proceedings of the 15th International Conference on the Foundations of Digital Games*. 1–10.
- [22] M Charity, Ahmed Khalifa, and Julian Togelius. 2020. Baba is y'all: Collaborative mixed-initiative level design. In *2020 IEEE Conference on Games (CoG)*. IEEE, 542–549.
- [23] M Charity, Dipika Rajesh, Sam Earle, and Julian Togelius. 2023. Amorphous Fortress: Observing Emergent Behavior in Multi-Agent FSMs. *Accepted to 1st ALIFE for and from games Workshop at ALIFE 2023* (2023).
- [24] M Charity and Julian Togelius. 2022. Aesthetic bot: interactively evolving game maps on Twitter. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 18. 18–25.

- [25] M Charity and Julian Togelius. 2022. Keke AI Competition: Solving puzzle levels in a dynamically changing mechanic space. In *2022 IEEE Conference on Games (CoG)*. IEEE, 570–575.
- [26] M Charity, Mayu Wilson, Steven Lee, Dipika Rajesh, Sam Earle, and Julian Togelius. 2024. Amorphous Fortress Online: Collaboratively Designing Open-Ended Multi-Agent AI and Game Environments. *Submitted to 2024 IEEE Conference on Games (CoG)* (2024).
- [27] Kate Compton. 2016. So You Want to Build a Generator. <https://www.tumblr.com/galaxykate0/139774965871/so-you-want-to-build-a-generator>
- [28] Kate Compton and Michael Mateas. 2015. Casual Creators.. In *ICCC*. 228–235.
- [29] Michael Cook. 2022. Puck: a slow and personal automated game designer. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 18. 232–239.
- [30] Michael Cook and Simon Colton. 2014. Ludus Ex Machina: Building A 3D Game Designer That Competes Alongside Humans.. In *ICCC*. 54–62.
- [31] Michael Cook, Simon Colton, and Jeremy Gow. 2012. Initial results from co-operative co-evolution for automated platformer design. In *Applications of Evolutionary Computation: EvoApplications 2012: EvoCOMNET, EvoCOMPLEX, EvoFIN, EvoGAMES, EvoHOT, EvoIASP, EvoNUM, EvoPAR, EvoRISK, EvoSTIM, and EvoSTOC, Málaga, Spain, April 11-13, 2012, Proceedings*. Springer, 194–203.
- [32] Michael Cook, Simon Colton, and Jeremy Gow. 2016. The angelina videogame design system—part i. *IEEE Transactions on Computational Intelligence and AI in Games* 9, 2 (2016), 192–203.

- [33] Michael Cook, Simon Colton, and Jeremy Gow. 2016. The angelina videogame design system—part ii. *IEEE Transactions on Computational Intelligence and AI in Games* 9, 3 (2016), 254–266.
- [34] Michael Cook, Simon Colton, Azalea Raad, and Jeremy Gow. 2013. Mechanic miner: Reflection-driven game mechanic discovery and level design. In *Applications of Evolutionary Computation*. Springer.
- [35] Seth Cooper. 2023. Sturgeon-MKIII: Simultaneous Level and Example Playthrough Generation via Constraint Satisfaction with Tile Rewrite Rules. In *Proceedings of the 18th International Conference on the Foundations of Digital Games*. 1–9.
- [36] Alicia David and Peyton Glore. 2010. The impact of design and aesthetics on usability, credibility, and learning in an online environment. *Online Journal of Distance Learning Administration* 13, 4 (2010).
- [37] Richard Dawkins. 1986. The Blind Watchmaker, WW. W. Norton& Co., New York (1986).
- [38] Omar Delarosa, Hang Dong, Mindy Ruan, Ahmed Khalifa, and Julian Togelius. 2021. Mixed-initiative level design with rl brush. In *International Conference on Computational Intelligence in Music, Sound, Art and Design (Part of EvoStar)*. Springer, 412–426.
- [39] Vincent Dumoulin, Jonathon Shlens, and Manjunath Kudlur. 2016. A learned representation for artistic style. *arXiv preprint arXiv:1610.07629* (2016).
- [40] Sam Earle. 2020. Using fractal neural networks to play simcity 1 and conway’s game of life at variable scales. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* (2020).
- [41] Sam Earle, M Charity, Dipika Rajesh, Mayu Wilson, and Julian Togelius. 2023. Quality Diversity in the Amorphous Fortress (QD-AF): Evolving

- for Complexity in 0-Player Games. *Accepted to Agent Learning in Open Endedness Workshop at NeurIPS 2023* (2023).
- [42] Sam Earle, Justin Snider, Matthew C. Fontaine, Stefanos Nikolaidis, and Julian Togelius. 2021. Illuminating Diverse Neural Cellular Automata for Level Generation. arXiv:2109.05489 [cs.NE]
 - [43] Marc Ebner, John Levine, Simon M Lucas, Tom Schaul, Tommy Thompson, and Julian Togelius. 2013. Towards a video game description language. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik.
 - [44] Raluca D Gaina, Diego Pérez-Liébana, and Simon M Lucas. 2016. General video game for 2 players: Framework and competition. In *Computer Science and Electronic Engineering*. IEEE, 186–191.
 - [45] Adrian Gonzalez, Matthew Guzdial, and Felix Ramos. 2020. Generating Gameplay-Relevant Art Assets with Transfer Learning. *arXiv preprint arXiv:2010.01681* (2020).
 - [46] Ian Goodfellow, Jean Pouget-Abadie, Mehdi Mirza, Bing Xu, David Warde-Farley, Sherjil Ozair, Aaron Courville, and Yoshua Bengio. 2020. Generative adversarial networks. *Commun. ACM* 63, 11 (2020), 139–144.
 - [47] Jonathan Gray, Kavya Srinet, Yacine Jernite, Haonan Yu, Zhuoyuan Chen, Demi Guo, Siddharth Goyal, C Lawrence Zitnick, and Arthur Szlam. 2019. Craftassist: A framework for dialogue-enabled interactive agents. *arXiv preprint arXiv:1907.08584* (2019).
 - [48] Djordje Grbic, Rasmus Berg Palm, Elias Najarro, Claire Glanois, and Sebastian Risi. 2021. EvoCraft: A New Challenge for Open-Endedness. Springer-Verlag.
 - [49] Michael Cerny Green, Ahmed Khalifa, Gabriella AB Barros, and Julian Togellius. 2017. ”Press Space to Fire”: Automatic Video Game Tutorial

- Generation. In *Artificial Intelligence and Interactive Digital Entertainment*. AAAI.
- [50] Michael Cerny Green, Victoria Yen, Sam Earle, Dipika Rajesh, Maria Edwards, and Lisa B Soros. 2021. Exploring open-ended gameplay features with Micro RollerCoaster Tycoon. *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment* (2021).
 - [51] Cristina Guerrero-Romero and Diego Perez-Liebana. 2021. Map-elites to generate a team of agents that elicits diverse automated gameplay. In *2021 IEEE Conference on Games (CoG)*. IEEE, 1–8.
 - [52] Matthew Guzdial, Nicholas Liao, and Mark Riedl. 2018. Co-creative level design via machine learning. *arXiv preprint arXiv:1809.09420* (2018).
 - [53] Kristian Hjaltason, Steffen Christoffersen, Julian Togelius, and Mark J Nelson. 2015. Game mechanics telling stories? An experiment.. In *FDG*. Citeseer.
 - [54] Mads Johansen and Michael Cook. 2021. Challenges in generating juice effects for automatically designed games. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 17. 42–49.
 - [55] Raffi Kamalian, Eric Yeh, Ying Zhang, Alice M Agogino, and Hideyuki Takagi. 2006. Reducing human fatigue in interactive evolutionary computation through fuzzy systems and machine learning systems. In *2006 IEEE International Conference on Fuzzy Systems*. IEEE, 678–684.
 - [56] Manuel Kerssemakers, Jeppe Tuxen, Julian Togelius, and Georgios N Yannakakis. 2012. A procedural procedural level generator generator. In *2012 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 335–341.
 - [57] Ahmed Khalifa, Philip Bontrager, Sam Earle, and Julian Togelius. 2020. Pcgrl: Procedural content generation via reinforcement learning. In *Proceedings*

- of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 16. 95–101.
- [58] Ahmed Khalifa, Michael Cerny Green, Gabriella Barros, and Julian Togelius. 2019. Intentional computational level design. In *Proceedings of The Genetic and Evolutionary Computation Conference*. 796–803.
 - [59] Ahmed Khalifa, Michael Cerny Green, Diego Perez-Liebana, and Julian Togelius. 2017. General video game rule generation. In *2017 IEEE Conference on Computational Intelligence and Games (CIG)*. IEEE, 170–177.
 - [60] Ahmed Khalifa, Michael Cerny Green, Diego Perez-Liebana, and Julian Togelius. 2019. General Video Game Rule Generation. arXiv:1906.05160 [cs.AI]
 - [61] Ahmed Khalifa, Scott Lee, Andy Nealen, and Julian Togelius. 2018. Talakat: Bullet hell generation through constrained map-elites. In *Proceedings of The Genetic and Evolutionary Computation Conference*. 1047–1054.
 - [62] Ahmed Khalifa, Diego Perez-Liebana, Simon M Lucas, and Julian Togelius. 2016. General video game level generation. In *Genetic and Evolutionary Computation Conference*. ACM, 253–259.
 - [63] Ahmed Khalifa and Julian Togelius. 2020. Multi-Objective level generator generation with Marahel. In *International Conference on the Foundations of Digital Games*. 1–8.
 - [64] Steven Orla Kimbrough, Gary J Koehler, Ming Lu, and David Harlan Wood. 2008. On a feasible–infeasible two-population (fi-2pop) genetic algorithm for constrained optimization: Distance tracing and no free lunch. *European Journal of Operational Research* 190, 2 (2008), 310–327.
 - [65] Diederik P Kingma, Max Welling, et al. 2019. An introduction to variational autoencoders. *Foundations and Trends® in Machine Learning* 12, 4 (2019), 307–392.

- [66] Martin Klissarov, Pierluca D’Oro, Shagun Sodhani, Roberta Raileanu, Pierre-Luc Bacon, Pascal Vincent, Amy Zhang, and Mikael Henaff. 2023. Motif: Intrinsic motivation from artificial intelligence feedback. *arXiv preprint arXiv:2310.00166* (2023).
- [67] Max Kreminski. 2023. “Generator’s Haunted”: A Brief, Spooky Account of Hauntological Effects in the Player Experience of Procedural Generation. In *Proceedings of the 18th International Conference on the Foundations of Digital Games*. 1–3.
- [68] Antonios Liapis, Hector P. Martinez, Julian Togelius, and Georgios N. Yannakakis. 2013. Adaptive game level creation through rank-based interactive evolution. In *2013 IEEE Conference on Computational Intelligence in Games (CIG)*. IEEE, 1–8. <https://doi.org/10.1109/CIG.2013.6633651>
- [69] Antonios Liapis, Georgios Yannakakis, and Julian Togelius. 2013. Designer modeling for personalized game content creation tools. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 9. 11–16.
- [70] Antonios Liapis and Georgios N Yannakakis. 2016. Boosting computational creativity with human interaction in mixed-initiative co-creation tasks. In *Digital Games Research Association (DiGRA) & Foundation of Digital Games (FDG)*.
- [71] Antonios Liapis, Georgios N Yannakakis, and Julian Togelius. 2011. Neuroevolutionary constrained optimization for content creation. In *2011 IEEE Conference on Computational Intelligence and Games (CIG’11)*. IEEE, 71–78.
- [72] Antonios Liapis, Georgios N Yannakakis, and Julian Togelius. 2011. Optimizing visual properties of game content through neuroevolution. In *Seventh Artificial Intelligence and Interactive Digital Entertainment Conference*.

- [73] Antonios Liapis, Georgios N Yannakakis, and Julian Togelius. 2012. Limitations of choice-based interactive evolution for game level design. In *Eighth Artificial Intelligence and Interactive Digital Entertainment Conference*.
- [74] Antonios Liapis, Georgios N Yannakakis, and Julian Togelius. 2013. Sentient world: Human-based procedural cartography. In *International Conference on Evolutionary and Biologically Inspired Music and Art*. Springer, 180–191.
- [75] Jialin Liu, Sam Snodgrass, Ahmed Khalifa, Sebastian Risi, Georgios N Yannakakis, and Julian Togelius. 2021. Deep learning for procedural content generation. *Neural Computing and Applications* 33, 1 (2021), 19–37.
- [76] Dagmar Loftsdottir and Matthew Guzdial. 2022. SketchBetween: Video-to-Video Synthesis for Sprite Animation via Sketches. In *Proceedings of the 17th International Conference on the Foundations of Digital Games*. 1–7.
- [77] Simon M Lucas and Vanessa Volz. 2019. Tile pattern kl-divergence for analysing and evolving game levels. In *Proceedings of the Genetic and Evolutionary Computation Conference*. 170–178.
- [78] Yecheng Jason Ma, William Liang, Guanzhi Wang, De-An Huang, Osbert Bastani, Dinesh Jayaraman, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Eureka: Human-level reward design via coding large language models. *arXiv preprint arXiv:2310.12931* (2023).
- [79] Tiago Machado, Dan Gopstein, Andy Nealen, and Julian Togelius. 2019. Pitako-recommending game design elements in cicero. In *2019 IEEE Conference on Games (CoG)*. IEEE, 1–8.
- [80] Yotam Mann. 2016. Ai duet. *Experiments with Google. See,* <https://experiments.withgoogle.com/ai/ai-duet> (2016).
- [81] Marian Mazzone and Ahmed Elgammal. 2019. Art, creativity, and the potential of artificial intelligence. In *Arts*, Vol. 8. MDPI, 26.

- [82] Timothy Merino, M Charity, and Julian Togelius. 2023. Interactive Latent Variable Evolution for the Generation of Minecraft Structures. In *Proceedings of the 18th International Conference on the Foundations of Digital Games*. 1–8.
- [83] Timothy Merino, Roman Negri, Dipika Rajesh, M Charity, and Julian Togelius. 2023. The five-dollar model: generating game maps and sprites from sentence embeddings. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 19. 107–115.
- [84] Afshin Mobramaein, Morteza Behrooz, and Jim Whitehead. 2018. CADI—A conversational assistive design interface for discovering pong variants. In *Artificial Intelligence and Interactive Digital Entertainment*. AAAI.
- [85] Jean-Baptiste Mouret and Jeff Clune. 2015. Illuminating search spaces by mapping elites. *arXiv preprint arXiv:1504.04909* (2015).
- [86] David Mullich. 2018. Game Mechanics vs. game features. <https://davidmullich.com/2018/08/06/game-mechanics-vs-game-features/>. Accessed: 3-16-2023.
- [87] Mark Nelson, Simon Colton, Edward Powley, Swen Gaudl, Peter Ivey, Rob Saunders, Blanca Perez Ferrer, and Michael Cook. 2017. Mixed-initiative approaches to on-device mobile game design. In *CEUR Workshop*.
- [88] Mark J Nelson and Michael Mateas. 2008. An interactive game-design assistant. In *Proceedings of the 13th international conference on Intelligent user interfaces*. 90–98.
- [89] Florence Smith Nicholls and Michael Cook. 2023. 'That Darned Sandstorm': A Study of Procedural Generation through Archaeological Storytelling. *Proceedings of the 18th International Conference on the Foundations of Digital Games* (2023).

- [90] Emma Stensby Norstein, Kai Olav Ellefsen, and Kyrre Glette. 2022. Open-ended search for environments and adapted agents using map-elites. In *International Conference on the Applications of Evolutionary Computation (Part of EvoStar)*. Springer, 651–666.
- [91] P Russel Norvig and S Artificial Intelligence. 2002. A modern approach. *Prentice Hall Upper Saddle River, NJ, USA: Rani, M., Nayak, R., & Vyas, OP (2015). An ontology-based adaptive personalized e-learning system, assisted by software agents on cloud storage. Knowledge-Based Systems* 90 (2002), 33–48.
- [92] David Olsen and Michael Mateas. 2009. Beep! Beep! Boom! Towards a planning model of coyote and road runner cartoons. In *Proceedings of the 4th International Conference on Foundations of Digital Games*. 145–152.
- [93] Jeff Orkin. 2006. Three states and a plan: the AI of FEAR. In *Game developers conference*, Vol. 2006. CMP Game Group SanJose, California, 4.
- [94] Leonardo T Pereira, Breno MF Viana, and Claudio FM Toledo. 2021. Procedural enemy generation through parallel evolutionary algorithm. In *2021 20th Brazilian Symposium on Computer Games and Digital Entertainment (SBGames)*. IEEE, 126–135.
- [95] Ethan Perez, Florian Strub, Harm De Vries, Vincent Dumoulin, and Aaron Courville. 2018. Film: Visual reasoning with a general conditioning layer. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 32.
- [96] Diego Perez-Liebana, Jialin Liu, Ahmed Khalifa, Raluca D Gaina, Julian Togelius, and Simon M Lucas. 2019. General video game ai: a multi-track framework for evaluating agents, games and content generation algorithms. *Transactions on Games* (2019).
- [97] Diego Perez-Liebana, Spyridon Samothrakis, Julian Togelius, Tom Schaul,

- and Simon M Lucas. 2016. General video game ai: Competition, challenges and opportunities. In *AAAI Conference on Artificial Intelligence*.
- [98] Martin Pichlmair and Mads Johansen. 2021. Designing game feel: A survey. *IEEE Transactions on Games* 14, 2 (2021), 138–152.
- [99] Martin Pichlmair and Charlene Putney. 2020. Procedural Generation for Divination and Inspiration. In *International Conference on the Foundations of Digital Games*. 1–7.
- [100] Thomas Pierrot and Arthur Flajolet. 2023. Evolving Populations of Diverse RL Agents with MAP-Elites. *arXiv preprint arXiv:2303.12803* (2023).
- [101] Aditya Ramesh, Prafulla Dhariwal, Alex Nichol, Casey Chu, and Mark Chen. 2022. Hierarchical text-conditional image generation with clip latents. *arXiv preprint arXiv:2204.06125* 1, 2 (2022), 3.
- [102] Juan Ramos et al. 2003. Using tf-idf to determine word relevance in document queries. In *Proceedings of the first instructional conference on machine learning*, Vol. 242. Citeseer, 29–48.
- [103] André Siqueira Ruela and Frederico Gadelha Guimarães. 2017. Procedural generation of non-player characters in massively multiplayer online strategy games. *Soft Computing* 21, 23 (2017), 7005–7020.
- [104] Akash Saravanan and Matthew Guzdial. 2022. Pixel VQ-VAEs for improved pixel art representation. *arXiv preprint arXiv:2203.12130* (2022).
- [105] Jacob Schrum, Jake Gutierrez, Vanessa Volz, Jialin Liu, Simon Lucas, and Sebastian Risi. 2020. Interactive evolution and exploration within latent level-design space of generative adversarial networks. In *Proceedings of the 2020 Genetic and Evolutionary Computation Conference*. 148–156.
- [106] Jimmy Secretan, Nicholas Beato, David B D Ambrosio, Adelein Rodriguez, Adam Campbell, and Kenneth O Stanley. 2008. Picbreeder: evolving pictures

- collaboratively online. In *Proceedings of the SIGCHI conference on human factors in computing systems*. 1759–1768.
- [107] Noor Shaker, Mohammad Shaker, and Julian Togelius. 2013. Ropossum: An authoring tool for designing, optimizing and solving cut the rope levels. In *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*.
 - [108] Noor Shaker, Julian Togelius, and Mark J Nelson. 2016. *Procedural content generation in games*. Springer.
 - [109] Miguel Sicart. 2008. Defining game mechanics. *Game studies* (2008).
 - [110] Abhishek Sinha, Jiaming Song, Chenlin Meng, and Stefano Ermon. 2021. D2c: Diffusion-decoding models for few-shot conditional generation. *Advances in Neural Information Processing Systems* 34 (2021), 12533–12548.
 - [111] Matthew Siper, Ahmed Khalifa, and Julian Togelius. 2022. Path of destruction: Learning an iterative level generator using a small dataset. In *2022 IEEE Symposium Series on Computational Intelligence (SSCI)*. IEEE, 337–343.
 - [112] Kristin Siu, Eric Butler, and Alexander Zook. 2016. A programming model for boss encounters in 2d action games. In *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*.
 - [113] Gillian Smith, Jim Whitehead, and Michael Mateas. 2010. Tanagra: A mixed-initiative level design tool. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*. 209–216.
 - [114] Ivan Y Smolyakov and Sergey A Belyaev. 2019. Design of the software architecture for starcraft video game on the basis of finite state machines. In *2019 IEEE Conference of Russian Young Researchers in Electrical and Electronic Engineering (EIConRus)*. IEEE, 356–359.

- [115] Eva Grouling Snider. 2023. Coyote time: What games can teach us about forgiveness in learning . <https://blogs.bsu.edu/teaching-innovation/2023/11/15/coyote-time-games-teach-forgiveness-learning/#:~:text=Named%20after%20the%20Wile%20E,small%20amount%20of%20wiggle%20room.>
- [116] Sam Snodgrass and Santiago Ontanón. 2016. Learning to generate video game maps using markov models. *IEEE transactions on computational intelligence and AI in games* 9, 4 (2016), 410–422.
- [117] Robyn Speer, Joshua Chin, and Catherine Havasi. 2017. Conceptnet 5.5: An open multilingual graph of general knowledge. In *Proceedings of the AAAI conference on artificial intelligence*, Vol. 31.
- [118] Joseph Suarez, Yilun Du, Clare Zhu, Igor Mordatch, and Phillip Isola. 2021. The neural mmo platform for massively multiagent research. *arXiv preprint arXiv:2110.07594* (2021).
- [119] Shyam Sudhakaran, Miguel González-Duque, Claire Glanois, Matthias Freiberger, Elias Najarro, and Sebastian Risi. 2023. Prompt-Guided Level Generation. In *Proceedings of the Companion Conference on Genetic and Evolutionary Computation*. 179–182.
- [120] Adam Summerville, Chris Martens, Ben Samuel, Joseph Osborn, Noah Wardrip-Fruin, and Michael Mateas. 2018. Gemini: Bidirectional generation and analysis of games via asp. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 14. 123–129.
- [121] Adam Summerville, Sam Snodgrass, Matthew Guzdial, Christoffer Holmgård, Amy K Hoover, Aaron Isaksen, Andy Nealen, and Julian Togelius. 2018. Procedural content generation via machine learning (PCGML). *IEEE Transactions on Games* 10, 3 (2018), 257–270.

- [122] Yuqian Sun, Zhouyi Li, Ke Fang, Chang Hee Lee, and Ali Asadipour. 2023. Language as reality: a co-creative storytelling game experience in 1001 nights using generative AI. In *Proceedings of the AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, Vol. 19. 425–434.
- [123] Steve Swink. 2008. *Game feel: a game designer’s guide to virtual sensation*. CRC press.
- [124] Sarjak Thakkar, Changxing Cao, Lifan Wang, Tae Jong Choi, and Julian Togelius. 2019. Autoencoder and evolutionary algorithm for level generation in lode runner. In *2019 IEEE Conference on Games (CoG)*. IEEE, 1–4.
- [125] Graham Todd, Sam Earle, Muhammad Umair Nasir, Michael Cerny Green, and Julian Togelius. 2023. Level Generation Through Large Language Models. In *Proceedings of the 18th International Conference on the Foundations of Digital Games*. 1–8.
- [126] Julian Togelius and Jurgen Schmidhuber. 2008. An experiment in automatic game design. In *2008 IEEE Symposium On Computational Intelligence and Games*. IEEE, 111–118.
- [127] Julian Togelius, Georgios N Yannakakis, Kenneth O Stanley, and Cameron Browne. 2010. Search-based procedural content generation. In *European Conference on the Applications of Evolutionary Computation*. Springer, 141–150.
- [128] Nao Tokui, Hitoshi Iba, et al. 2000. Music composition with interactive evolutionary computation. In *Proceedings of the third international conference on generative art*, Vol. 17. 215–226.
- [129] Ruben Rodriguez Torrado, Philip Bontrager, Julian Togelius, Jialin Liu, and Diego Perez-Liebana. 2018. Deep Reinforcement Learning for General Video Game AI. In *Computational Intelligence and Games*. IEEE, 1–8.

- [130] Mike Treanor, Bryan Blackford, Michael Mateas, and Ian Bogost. 2012. Game-o-matic: Generating videogames that represent ideas. In *Proceedings of the The third workshop on Procedural Content Generation in Games*. 1–8.
- [131] Susanna Värtinen, Perttu Hämäläinen, and Christian Guckelsberger. 2022. Generating role-playing game quests with gpt language models. *IEEE Transactions on Games* (2022).
- [132] Ashish Vaswani, Noam Shazeer, Niki Parmar, Jakob Uszkoreit, Llion Jones, Aidan N Gomez, Łukasz Kaiser, and Illia Polosukhin. 2017. Attention is all you need. *Advances in neural information processing systems* 30 (2017).
- [133] Tony Veale and Mike Cook. 2018. *Twitterbots: Making machines that make meaning*. MIT Press.
- [134] Vanessa Volz, Jacob Schrum, Jialin Liu, Simon M Lucas, Adam Smith, and Sebastian Risi. 2018. Evolving mario levels in the latent space of a deep convolutional generative adversarial network. In *Proceedings of the genetic and evolutionary computation conference*. 221–228.
- [135] Guanzhi Wang, Yuqi Xie, Yunfan Jiang, Ajay Mandlekar, Chaowei Xiao, Yuke Zhu, Linxi Fan, and Anima Anandkumar. 2023. Voyager: An open-ended embodied agent with large language models. *arXiv preprint arXiv:2305.16291* (2023).
- [136] Jonah Warren. 2019. Tiny online game engines. In *2019 IEEE Games, Entertainment, Media Conference (GEM)*. IEEE, 1–7.
- [137] Jiajun Wu, Chengkai Zhang, Tianfan Xue, Bill Freeman, and Josh Tenenbaum. 2016. Learning a probabilistic latent space of object shapes via 3d generative-adversarial modeling. *Advances in neural information processing systems* 29 (2016).
- [138] Georgios Yannakakis and Julian Togelius. 2018. *Artificial Intelligence and Games*. Springer.

- [139] Georgios N Yannakakis, Antonios Liapis, and Constantine Alexopoulos. 2014. Mixed-initiative co-creativity. (2014).
- [140] Nick Zangwill. 2003. Aesthetic judgment. In *The Stanford Encyclopedia of Philosophy*, Edward N. Zalta (Ed.).
- [141] Hongyi Zhang, Moustapha Cisse, Yann N Dauphin, and David Lopez-Paz. 2017. mixup: Beyond empirical risk minimization. *arXiv preprint arXiv:1710.09412* (2017).
- [142] Hongwei Zhou, Yichen Gong, Luvneesh Mugrai, Ahmed Khalifa, Andy Nealen, and Julian Togelius. 2018. A hybrid search agent in pommerman. In *Proceedings of the 13th international conference on the foundations of digital games*. 1–4.
- [143] Zisen Zhou and Matthew Guzdial. 2021. Toward Co-creative Dungeon Generation via Transfer Learning. *arXiv preprint arXiv:2107.12533* (2021).
- [144] Alexander Zook and Mark Riedl. 2014. Automatic game design via mechanic generation. In *Conference on Artificial Intelligence*. AAAI.