CICERO: Computationally Intelligent Collaborative EnviROnment for game and level design

Tiago Machado
New York University
tiago.machado@nyu.edu

Andy Nealen
New York University
nealen@nyu.edu

Julian Togelius
New York University
julian.togelius@nyu.edu

Abstract

Mixed-initiative AI–based game design tools assist game designers by automating or semi-automating part of the design process, while also allowing free editing of game physics, graphics and/or mechanics. This can be done by providing suggestions, feedback, and constraint checking, often based on automatic playtesting. Several prototype tools that incorporate such abilities have been developed in recent years, however, they are all specific to a single game or a very narrow game genre. We present CICERO, a general-purpose AI-assisted design tool built on top of the General Video Game AI (GVGAI) framework. By leveraging the general game playing and game representation abilities of that framework, we show how several types of AI-based design assistance can be used across multiple games.

INTRODUCTION

In the past few years, numerous game design tools have been released. They offer several features that facilitate the tasks of independent and professional developers (Craighead, Burke, and Murphy 2007).

Among their features, we can easily find many of them associated with graphics, physics, network and social media. Although AI is also included, they are often focusing on content creation that is related to pathfinding and non-player characters behaviors. (Pattrasitidecha 2014).

However, what is absent in many of these tools, commercial or otherwise, is the use of AI to assist the developers themselves. An AI assistant, among other things, has the potential to make game development more accessible for non-experts, and allow experts to focus on high-level design tasks.

Concurrently, we have seen some projects developed by researchers that have been developing AI-assisted Game Design tools. These systems, many of them based on search techniques, help designers by creating and evaluating game contents like levels and maps (Smith, Whitehead, and Mateas 2010; Shaker, Shaker, and Togelius 2013; Liapis, Yannakakis, and Togelius 2013a; Ferreira 2015; Horn et al. 2014). What, essentially, could be interpreted as the main function of AI-Game design assistants: a machine working in collaboration with humans. (Yannakakis, Liapis, and Alexopoulos 2014).

Another feature is the use of visualization techniques. In the work of Bowman et al. (Bowman, Elmqvist, and Jankun-Kelly 2012) and El-Nasr et al. (El-Nasr, Drachen, and Canossa 2013). It is clear that visualization has become a popular tool among developers for game data telemetry analysis, debugging, balancing, and playtesting. An example of a commercial tool is Bioware‘s Skynet. It performs telemetry among several servers and game sessions. Unity Analytics, developed by the Unity Game Engine team, also provides data that developers can use to understand what players are doing and tune the game. It also offers other metrics that help developers devise strategies to retain and attract new players.

All these techniques together can serve developers as if another team member was performing specifics tasks, hard for humans but easy for computers. They can foster less tangible skills like creativity and increase quality and productivity. Ben Schneiderman, in his paper about creativity support tools (Shneiderman 2007), discusses how researchers and designers can create tools that allow people to be more creative more often. Basically, three creativity schools are presented: Structuralist, where creativity is achieved by following structured methods; Inspirationalist, in which creativity arises from unusual and unexpected experiences; Situationalist, indicates that social work is creativity. Although, many of the existing tools introduce some of the ideas discussed by Schneiderman, the development of AI Game Design Assistants that can increase the capacities and qualities of a team is still in its infancy.

Many of these works are focused on a single game. Despite interesting results, their methods cannot, easily, be adapted to other projects without rewriting the software. Even Sentient Sketchbook is limited, though to a small range of StarCraft-like games (Liapis, Yannakakis, and Togelius 2013b). Furthermore, there are many open design questions about which elements and features to include in AI Game Assistants that make sense in the daily activities of a designer.

In this paper, we discuss the design of an AI Game Design Assistant Tool based on the General Video Game Framework (GVGAI). GVGAI is a framework for creating AI controllers that are able to play different games and fits our gen-

erality principle in the design of this system.

Related Works

In this section, we described work that is closely related to the development of AI Game Design assistants. We also, discuss some work that offers support for designers by gathering and evaluating data from game playing sessions.

AI Game Design Assistant Tools

Tanagra (Smith, Whitehead, and Mateas 2010) is a system that assists humans in the design of levels for 2D platform games. The system's UI allows user edits in real time. It generates several possibilities for a level, and guarantees that all of them are playable, which eliminates the necessity of play testing to find salient and game-breaking level design flaws.

Ropossum (Shaker, Shaker, and Togelius 2013), is a system which generates and simulates levels for the game Cut The Rope. It helps users in the creation and evaluation of their own levels. One of these modules is an evolutionary framework for procedural content generation, the second evolves playable content and tests levels designed by humans. Both modules are optimized to allow real-time feedback after user inputs. The system, from a given state, generates the next possible actions of the player and explores it using a depth-first-tree until it finds a solution, if available, at the given level.

Sentient Sketchbook (Liapis, Yannakakis, and Togelius 2013a) is also an assistant for generating game levels, in this case focusing on strategy games, or dungeons for roguelike games. The novelty of the system lies in its real time design suggestions. The users interacts by editing their levels while the system works in the background generating recommendations based on the user's design. It is based on evolutionary techniques seeded by the user's inputs that guarantee that all the suggestions are playable.

StreamLevels UI allows users to draw strokes on a canvas or upload it from real data (Ferreira 2015). The strokes represent a player trace, and the system uses it to create the levels. StreamLevels does not formally generate the levels, it is only used for defining the overall shape of the level. The idea is to have a general level editor, so one can use it for different game mechanics.

In (Butler et al. 2013) the authors define a system that assists with level and progression edits. By progression, we can understand a sequence of different elements that a player encounters in a specific level. For example, in a puzzle game, the puzzles are the elements in a level. The sequence of all puzzles that lead to the end of the level, incorporating all skills in the game is called a progression. The user can interact with the system by editing a level and its progressions. Procedural techniques are used to check if changes, in the progressions or levels, create conflicts that can harm the design. The authors state that many of the system features are game independent, despite the fact that it was developed to attend in-house products.

The work of (Nelson and Mateas 2009) is not an implementation of an assistant tool like the previous ones, but it is an effort to understand the requirements on this type of system from the perspective of designers and developers. The authors interviewed three development teams and used some prototypes to guide the process. As a result, there is a split about tools which assist in checking if a game has the goals proposed by its design, and tools that check if the goals are reachable by the players. Another suggestion is the use of queries to identify design flaws or particular questions like "what is the most recurrent path adopted by players?".

Game Telemetry Tools

Game Telemetry has emerged as a source of Business Intelligence in the game industry. In this section, we discuss some work that shows benefits of using such resources combined with data mining, statistics and other common AI methods to provide useful information for game designers.

The work of (Bowman, Elmqvist, and Jankun-Kelly 2012) presents an overview about visualization design patterns and how to use them in order to provide information to both users and developers. The paper highlights some benefits of visualization techniques to designers such as bug detection, game balance issues, and draws inspiration from player behavior.

In (Bauckhage et al. 2012) the authors analyze data about five different action-adventure or shooter games, such as Tomb Raider: Underworld and Medal of Honor, in order to identify how long a game can keep the player's engagement. The authors present statistic measures that may indicate when a player will lose interest in a game and stop playing.

Trying to mitigate visualization problems in large data sets of gameplay sessions, the work of (Feltwell et al. 2015) uses a Dendrogram, a common visualization tool in the field of computational biology, designed to display results of hierarchical clustering of genomes. The authors collected data from 32 volunteers that played the game Red Orchestra: Ostfront 41-45. The gathered data was analyzed by professional designers. They argued in favor of the technique stating that it is very useful to help to understand a large heat map. More importantly, it helps in finding small, but important, details that are hard to notice in large data sets.

G-Player (Canossa, Nguyen, and El-Nasr 2016) is a visualization tool designed for exploratory analysis of multimodal data, utilizing a case study data from a role-playing game. Using the VPAL game, a mod based on Fallout: New Vegas, the authors demonstrate how this tool leads to improved understanding of player behavior data. The user can choose many player features to track, such as interactions with NPCs, items and visited locations. Equipped with visualization querying techniques such as spatial-temporal constraint modifiers and Boolean operators on events, the system allows comparison between individual players or groups. Professional developers from Unity and Ubisoft provided positive feedback about the tool. The authors are working to expand the tool to attend other popular game genres, like multiplayer online battle arenas (MOBAs) which have properties that fit well with the system purposes.

By analyzing the literature about AI game design assistants, we can observe some patterns.
For example, many of the tools seem to be tailored to one game or to a game genre in the best hypothesis (Shaker,

Shaker, and Togelius 2013; Smith, Whitehead, and Mateas 2010). Some works started to address the problem of providing assistance to a broad genre of games like (Ferreira 2015) and (Butler et al. 2013), although their examples are still related to a particular game.

Telemetry tools seem to provide a more general use, especially on commercial platforms. However, Canossa et al. (Canossa, Nguyen, and El-Nasr 2016) state that the next step is expanding the tool to cover a broader area of games in particular genres. Even though features like visualization seem commonplace, there are issues and design opportunities to further explore this (Feltwell et al. 2015).

Finally, we also believe that some work (Nelson and Mateas 2009) is in need of an update, as the interviews were conducted in a pre "free game engine" era.
Many tools have been released since these publications and the design community is more willing to contribute to requirements for what they need from an AI assistant.

Cicero - A VGDL AI Game Design Assistant

CICERO is an AI-based game design tool which is intended to be significantly more general than previous tools while incorporating several types of AI-based design support. The system allows the design of games and levels in a wide range of genres. It can run simulations to provide data about the users' designs, foster insights, and assistance about corrections, and suggest what to do to improve their work.
Based on our literature review, we believe that CICERO is the first effort in creating a general AI game design assistant and this is the main contribution of this work [1].

Generality

CICERO allows developers to prototype games and its levels. In order to allow designers to prototype a wide range of games, we use the General Video Game Framework - GVGAI (Perez et al. 2015), which is a framework built upon the Video Game Description Language - VGDL (Ebner et al. 2013; Schaul 2013) that offers the possibility of writing games with just a few lines of code. Together, they provide a set of tools for developers to create AI controllers that are able to play the games they have created. Our tool uses various AI controllers available in GVGAI. Some are based on algorithms like Monte Carlo Tree Search (MCTS), others are the champions of previous editions of the GVGAI competition–an annual challenge that awards the AI controller which performs best in as many games as possible. The users can select these controllers to play their games, or a manual controller to play the game by themselves. The main UI of our system is an editor for the VGDL language (see Figure 1). It allows the user to define behaviors for game elements such as NPCs and Power Ups, specify what happens when two game elements collide and determine the win and loss conditions. There is also a code area for those who like to inspect how the code is generated by their actions within the UI.

[1]Link to see the system video demonstration: https://goo.gl/hN6Nmz

Features

Besides the game editing mechanisms, CICERO offers the following features: a recommender system for game mechanics, statistics about the game rules, and a level visualization system.

Recommender Mechanics

The recommender system for game mechanics takes the current game content, compares it with other games in the VGDL library, and suggests new content from games that match with the one the user is editing.

In order to compare content in different games, our system calculates a Euclidean vector distance of the types and parameters that represent game elements in the VGDL. Content that reach the top score in a ranking is recommended (see our earlier work on the recommender system (Machado et al. 2016)).

Game Rule Statistics

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

Level Visualization - Heat Maps

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

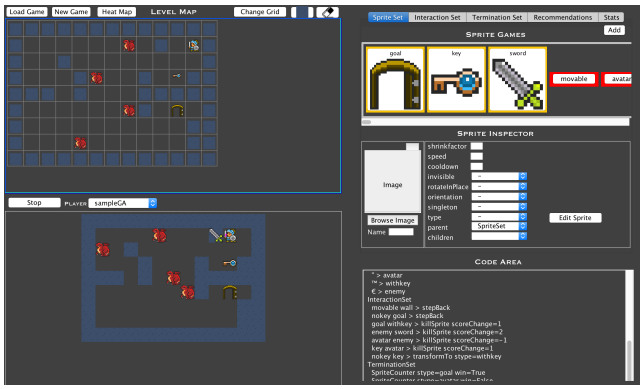

Figure 1: The system UI. On the left, the users can edit levels and play game simulations. On the right, they can specify the game's rules and game element behaviors

Technical Details

CICERO was developed as a pure Java desktop application. All of its functions only uses native Java support available in the 8th distribution of the language. We made this choice because as stated before, CICERO is developed on top of the GVG-AI Framework, which is entirely based on Java.

Game Description in VGDL

Every game written in VGDL has two text files. The first is the description of the game. The description is a tree with four nodes, each one with a specific function in describing the game:

Sprite Set This node contains every sprite in the game. We need to stress that a VGDL sprite does not refer just to an image. It actually contains a list of parameters that defines its behavior in the game. The node is the root of a Depth First Search Tree, which means that every sprite can have as many children as a designer wishes. Consequently, the children inherit parents' properties.

Level Mapping This node creates a char symbol for each sprite defined in the section above. It is useful to represent the sprites in the levels of the game.

Interaction Set This node specifies what happens every time a sprite A collides with a sprite B.

Termination Set This node verifies if certain conditions to end a game are satisfied.

All those nodes have height 1 (except the Sprite Set node) and can have as many children as the designer wants.

The second file is used to design the game level. At this point, the Level Mapping section of the first file does the binding between the two files.

Creating and Editing Functions

If the user is developing a game from scratch or editing an existed one, there are four main classes that take care of these process: Node, ControlVGDL, GameTree, GameLevel.

Node The class Node is already implemented in the GVG-AI Framework. It contains the main functions that describe a game in VGDL. This means that the tree structure of the nodes explained in the section above is handled by this class. It was extended to become a Java Button. By doing this, we could create a visual representation of every node in the tree with all the interactions allowed by the Java Swing package.

ControlVGDL The ControlVGDL class controls every action of the user when dealing with creation and edition in game structure. For example, if a sprite is added, the ControlVGDL will gather all the data from the user interface, do a parse to organize them properly and create a node to add in the sprite set tree.

GameTree and GameMap The GameTree and the GameMap classes keep all the game and level structures that the user is working, on the fly. Then when the user is done and wants to play the game, everything is automatically saved in two different files. Finally, the GVG-AI Framework, access these new files and run the user's game.

Game Rule Statistics Implementation

The GVG-AI has a class to play the game, properly called by Game. So every rule of the game is played by this class when some event triggers it. Therefore, the Statistics tool works within the Game class. Basically, every time a rule is played, a stat is created. It stores the rule and the two sprites that activated it. Every stat has a counter, it increases when a rule already stored is played again. The statistics tool creates a ranking of the most used rules in real-time. It uses the counter of every rule to sort the ranking based on the percentage use of the rule, normalized over all rules.

Visualization Implementation

The visualization prints a heat map of the game object behavior over the level. It extends every Agent class available in the GVG-AI Framework. Therefore, every Agent contains a TrackControl object. This object stores every position of the agent during game play. It also stores a count value of how many times the same position was visited. So, in every game update, the TrackControl defines the alpha channel of every position. It normalize by the most observed count value (the most visited one). Then the alpha channel of every position is defined by the count value of the position divided by the count value of the most visited one. The same principle is applied to all other game objects in the game (i.e enemies, items, etc.).

Mechanics Recommender Implementation

The Mechanics Recommender is based on similarities between the game a user is developing and the games in the VGDL library. VGDL defines mechanics based on the sprite behaviors and on the interactions among them. Sprites and Interactions are defined separately in the language and they have specific types and parameters. Therefore, in order to provide suggestions there are two main comparisons in course: Sprite similarity and Interaction similarity.

Sprite Similarity Its function is to compute the normalized vector distance between the parameters vectors and subtracts the result from a maximum pre-defined value when the sprite types are equals. It returns a minimum predefined value when the sprites types differ.

Interaction Similarity It checks when two sprite types involved in two interactions are similar. If so, the normalized vector distance between the parameters vectors is subtracted from the maximum predefined value. Otherwise, it returns the minimum pre-defined value. The distance is computed similarly to the sprite similarity case.

The comparisons take into account every game available in the VGDL library. So every comparison results in a value that is used to generate a rank. From this rank, the system picks the most and the least common sprites. The former in order to provide the sprite that is most suitable to the game in development. The latter, to influence the users to expand

their mechanics to a not so common situation, at least based on what is stored in the VGDL library. For an explanation of how the comparisons are made see Figure 2. The whole system and all of its algorithms are presented in (Machado et al. 2016).

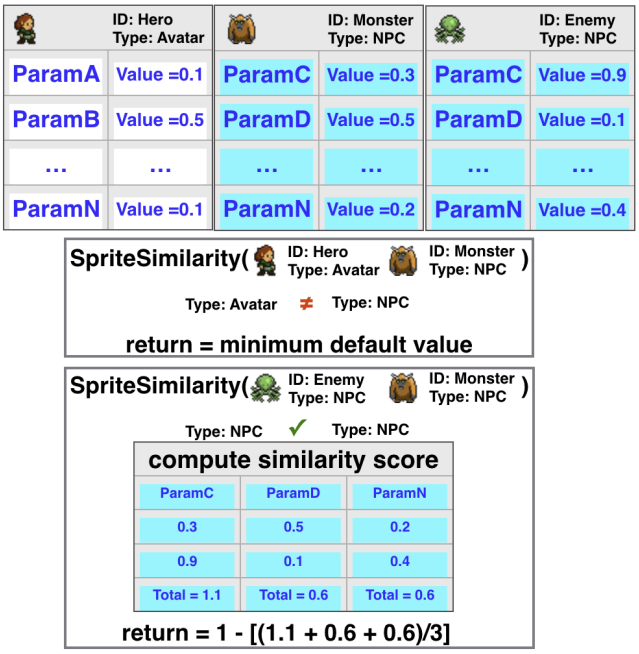

Figure 2: The table on the top shows sprites examples of different types. In the middle, the SpriteSimilarity method returns a minimum default value. In the bottom, the same method computes the similarity score for sprites of the same type.

Cicero In Action

This section shows some cases that present uses for the recommender mechanics, the statistics tool and the level visualizations.

Example of Mechanics Recommender

We developed a simple game whose main goal is to escape from a monster. See Figure 5.

We then asked the system to recommend new mechanics for this simple game. After comparing this simple game definition with all others in the library, the system prepares a ranked list and provides two suggestions–the most and the least common mechanic in similar games–to allow the users to explore between popular and unusual choices for the game they are developing.

In this case, both of the recommendations were weapons: a sword and a shooter. The suggestions bring the two elements necessary to create a new rule for our game: a specification of a sprite behavior and what happens when this sprite hits another one. The UI provides all the details of these specifications, such that the user can decide which one they want (See Figure 3). The users can accept the default

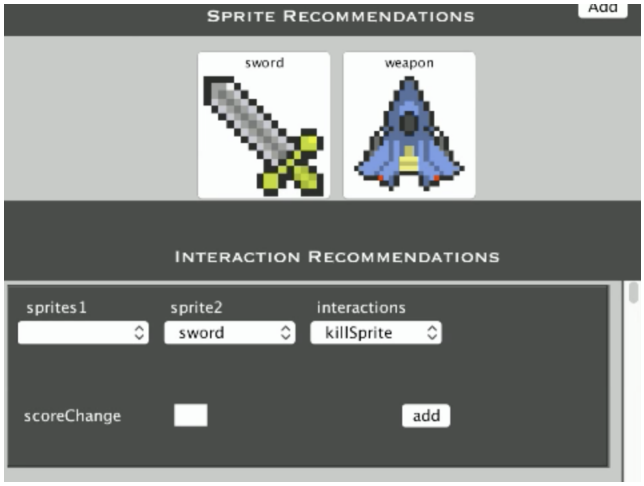

Figure 3: The system presents two recommendations. The users can inspect and pick one to add in their game

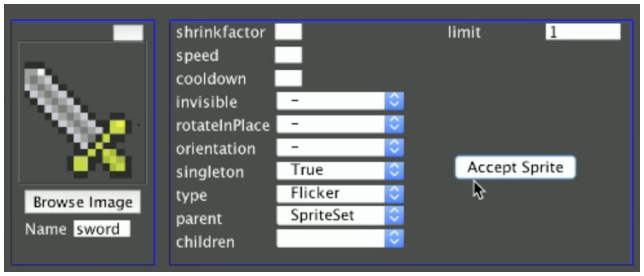

Figure 4: Each recommendation brings sprite behaviors and interaction definitions that the user can inspect and modify

suggestion(s), perform adaptations before they accepts, or ignore it(Figure 4).

Figure 5: In this simple game the player avoids being caught by a monster

Example of Game Rule Statistics

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

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

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

Figure 6: The game rules diagnostic shows that the two last rules are never used (the more red the rule, the more often it is used). The first two columns show the name of the game elements which activate a rule (third column). The fourth column contains the percentage use of the rule, normalized over all rules.

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

The other rule, the one that prohibits the player from reaching the goal without a key, allows us draw some conclusions about the original level design. First off, the rule doesn't trigger because the controller always retrieves the key. In the original level design, the key is near the player spawn point, so there is no challenge in accomplishing this objective. In order to correct it, if the designer wishes to do so, to increase the challenge and fully test all the rules, they just need to change the position of the key or player's spawn point (See Figure 7) in the UI level editor (See Figure 6 for a sample of the statistics report).

Example of the level visualization system

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

In Figures 8 and 9 we can see a heat map of the controller playing level 3 of the game "Zelda" and the level 0 of a "Space Invaders" clone, respectively.

Discussion

In the previous sections, we presented the development of CICERO, a game design editor that is designed with the purpose of being a general AI assistant across a myriad of different game genres. Currently, besides the common tasks of defining rules and levels, the system has three main features.

The first of these is the Game Mechanics Recommender. It is an attempt to provide assistance to designers in order

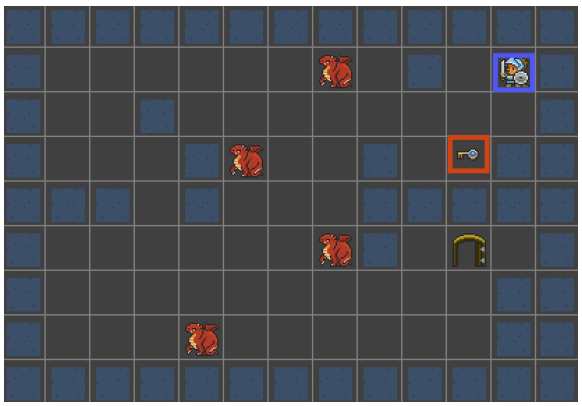

Figure 7: The key place (red square) is too close to the player spawn point (blue square). It makes the level less challenging and means that one of the rules is never tested.

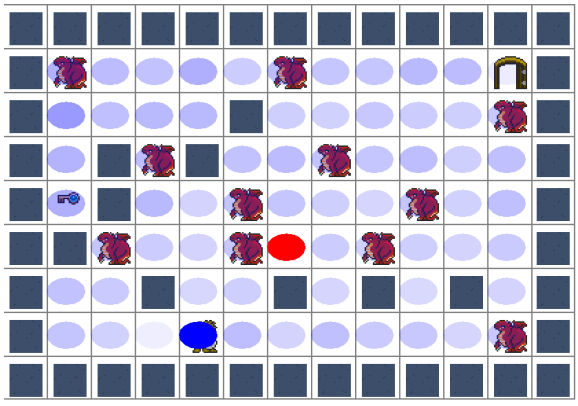

Figure 8: The user chooses to see the controller's path through the level: the more blue a spot is, more time the controller spends in that area. The same applies to the red color, that shows the places where the controller dies often.

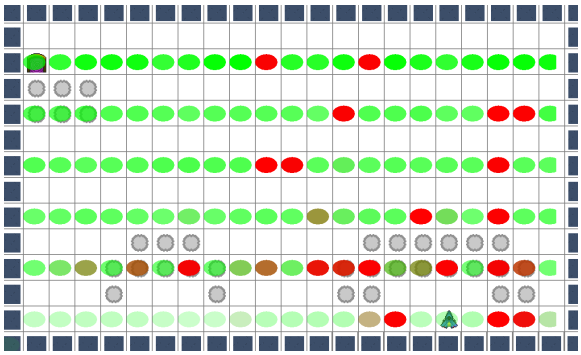

Figure 9: In this clone of Space Invaders, the user tracks the enemies' movements (green ellipses) and their most probable places of death (red ellipses). The death spots becomes more common when the enemies are near the player, which leads to a decrease of the number of enemies as well.

to facilitate their work by providing mechanics that were tested in previous games, similar to the recommender systems in online shopping, and echoing the principles of software reusability. This feature is meant to foster users' creativity in an inspirationalist way (Shneiderman 2007).

The Game Rule Statistics tool helps the user diagnose the usage of game rules at a specific level. It helps the designer to identify flaws in their rule definitions as well as to draw conclusions about how to improve the levels and the game as a whole.

Finally, the visualization system is a tool that looks to the users' current design and offers an interface where they can select what to inspect. By printing heat maps, it allows the users to see the behavior of the game elements through several game simulations.

Conclusion and Future Work

This paper presents the design of an AI Assisted Game Design Tool called CICERO. Many AI game assistants nowadays have offered several contributions to the field, however many of them can be used in limited context like a single game or a game genre in the best hypothesis.

With this in mind, CICERO follows the principle to be as general as possible. It allows the design of many games and levels, and provides three main features currently: a recommender system for game mechanics, a game rule statistics diagnosis and a visualization system.

We are now working on improving the Mechanics Recommender in order to provide accurate ways of comparing content in different games and provide better suggestions to what the users can do with their game, like suggestions based on playthrough or procedural content generation techniques.

References

Bauckhage, C.; Kersting, K.; Sifa, R.; Thurau, C.; Drachen, A.; and Canossa, A. 2012. How players lose interest in playing a game: An empirical study based on distributions of total playing times. In *2012 IEEE Conference on Computational Intelligence and Games (CIG)*, 139–146. IEEE.

Bowman, B.; Elmqvist, N.; and Jankun-Kelly, T. 2012. Toward visualization for games: Theory, design space, and patterns. *IEEE transactions on visualization and computer graphics* 18(11):1956–1968.

Butler, E.; Smith, A. M.; Liu, Y.-E.; and Popovic, Z. 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. ACM.

Canossa, A.; Nguyen, T.-H. D.; and El-Nasr, M. S. 2016. G-player: Exploratory visual analytics for accessible knowledge discovery.

Craighead, J.; Burke, J.; and Murphy, R. 2007. Using the unity game engine to develop sarge: a case study. *Computer* 4552:366–372.

Ebner, M.; Levine, J.; Lucas, S. M.; Schaul, T.; Thompson, T.; and Togelius, J. 2013. Towards a video game description language.

El-Nasr, M. S.; Drachen, A.; and Canossa, A. 2013. *Game analytics: Maximizing the value of player data*. Springer Science & Business Media.

Feltwell, T.; Cielniak, G.; Dickinson, P.; Kirman, B. J.; and Lawson, S. 2015. Dendrogram visualization as a game design tool. In *Proceedings of the 2015 Annual Symposium on Computer-Human Interaction in Play*, CHI PLAY '15, 505–510. New York, NY, USA: ACM.

Ferreira, L. N. 2015. Streamlevels: Using visualization to generate platform levels.

Horn, B.; Dahlskog, S.; Shaker, N.; Smith, G.; and Togelius, J. 2014. A comparative evaluation of procedural level generators in the mario ai framework.

Liapis, A.; Yannakakis, G. N.; and Togelius, J. 2013a. Sentient sketchbook: Computer-aided game level authoring. In *FDG*, 213–220.

Liapis, A.; Yannakakis, G. N.; and Togelius, J. 2013b. Towards a generic method of evaluating game levels. In *AIIDE*.

Machado, T.; Bravi, I.; Wang, Z.; Nealen, A.; and Togelius, J. 2016. Shopping for game mechanics.

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

Pattrasitidecha, A. 2014. Comparison and evaluation of 3d mobile game engines. *Chalmers University of Technology, University of Gothenburg, Göteborg, Sweden, Master Thesis, févr.*

Perez, D.; Samothrakis, S.; Togelius, J.; Schaul, T.; Lucas, S.; Couëtoux, A.; Lee, J.; Lim, C.-U.; and Thompson, T. 2015. The 2014 general video game playing competition.

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

Shaker, N.; Shaker, M.; and Togelius, J. 2013. Ropossum: An authoring tool for designing, optimizing and solving cut the rope levels. In *AIIDE*.

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

Smith, G.; Whitehead, J.; and Mateas, M. 2010. Tanagra: A mixed-initiative level design tool. In *Proceedings of the Fifth International Conference on the Foundations of Digital Games*, 209–216. ACM.

Yannakakis, G. N.; Liapis, A.; and Alexopoulos, C. 2014. Mixed-initiative co-creativity. In *Proceedings of the 9th Conference on the Foundations of Digital Games*.