AI-assisted game debugging with Cicero

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

Daniel Gopstein
New York University
dgopstein@nyu.edu

Andy Nealen
New York University
nealen@nyu.edu

Oded Nov
New York University
onov@nyu.edu

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

***Abstract*—We present Cicero, a mixed-initiative application for prototyping two-dimensional sprite-based games across different genres such as shooters, puzzles, and action games. Cicero provides a host of features which can offer assistance in different stages of the game development process. Noteworthy features include AI agents for gameplay simulation, a game mechanics recommender system, a playtrace aggregator, heatmap-based game analysis, a sequential replay mechanism, and a query system that allows searching for particular interaction patterns. In order to evaluate the efficacy and usefulness of the different features of Cicero, we conducted a user study in which we compared how users perform in game debugging tasks with different kinds of assistance.**

***Keywords*—AI for Games, Game Debugging, Game Design, Game Prototyping, AI-Assisted Game Design Tools**

I. Introduction

Almost fifteen years ago, Jonathan Blow wrote an article entitled "Game Development: Harder Than You Think", in which he discussed the complexities of building, managing and evolving a game[1]. This complexity has only increased with new technologies and methods that have arisen due to new developments in fields like graphics, physics, networking, social media and many others. Since Blow's publication, we have seen the rise of mixed-initiative tools, a hybrid in which humans and artificial intelligence (AI) work together to solve specific problems. At the same time, the game industry has became one of the most diverse industries in the world with segments based on disparate platforms (mobile, consoles, web, etc.), diverse customers (children, teenagers, adults, the elderly, etc.) and even different goals (social and health games for example).

To address the increasing complexity outlined by Blow and the growing consumer demand, mixed-initiative approaches were embraced by game researchers and industry professionals alike. Nowadays, game designers have tools to assist them in a myriad of tasks, from level generation to game balancing. For the systems that offer AI assistance, the term AI-assisted Game Design Tools was coined. Despite their impressive results, these tools also present challenges. One of the most prevalent problems is a lack of generalizability. Many of the techniques presented are directly attached to a specific game. It then becomes difficult when a developer wants to apply the same techniques to another project without having to re-implement everything again from scratch. Another issue is the lack of empirical justification for these approaches. There is a dearth of literature on the human factors of mixed-initiatives systems designed for game development tasks.

With these challenges in mind we present Cicero, an AI-assisted Game Design Tool that allows designers to prototype and simulate their games by using intelligent agents together with other computational methods like visualizations and queries. Cicero is built on top of the General Video Game AI Framework (GVGAI) and offers more generality than systems built for stand-alone games. Users can apply all of Cicero's features to a number of different genres within the gridphysics-based-games sphere. To highlight the immediate advantage these methods offer game designers we describe a user study in which we compare how humans perform debugging games with and without AI assistance. To detail the contrasting styles of features within Cicero, we also present a comparison between its query and visualization systems.

II. Related Work

A. Mixed-initiative game development tools

Mixed-Initiative User Interfaces have been well discussed [2], [3], [4] and many fields have shown the benefits of the paradigm. It is not a novel term in the game community either, however it has only come to be embraced recently. Among the sub-fields of game design where mixed-initiative methods appear most promising, are for helping developers overcome the complexities of game design and meet the increasing demand of the game industry [5].

One of the more recent examples of Mixed-Initiative applications in games is Tanagra [6], a tool that assists humans in designing levels for 2D platform games. The system works in real time, creating many different guarenteed-playable levels for the designer to choose from. Similarly, Ropossum [7], also generates and solves levels for the popular physics puzzle game Cut The Rope. The user is assisted in the tasks of level design and evaluation. The tool is optimized to allow real-time feedback from a given state, after receiving a sequence of user inputs. It generates the possible actions for the player until it finds a solution, if available. Sentient Sketchbook [8] offers more generality than the two works already cited on this section and according to its authors it also fosters human creativity [9]. It is also a tool to assist the creation of game levels. It provides this assistance on strategy and roguelike games. The system shows level suggestions in real-time. It allows the users to interact by editing their levels while generating recommendations based on previous choices. For the game *Refraction*, Smith et al. [10] and Butler et al. [11] present independent implementations of three diverse level design automation tools. They use Procedural Content Generation (PCG) techniques and Answer Set Programming (ASP) to explore the intended design space and offer levels with playability guarantee.

Aside from level generation, Isaksen et al. [12] presented an evolutionary algorithm to find variants for the popular mobile game *Flappy Bird*. The discovered game variants showed themselves to be significantly different from the original game regarding challenge, game feel, and theme. Still aside from level generation, we have BIPED [13], which is focused on giving insights to designers in order to help them generate games. It offers assistance in the early stages of the game design process. Using BIPED, designers can leverage simple ideas and end up with a playable prototype and a formal rule system that they can test as if they were sketching their ideas for quick evaluation [14].

Mixed Initiative systems have been applied to table-top games as well. The work of Osborn et al. [15] presents a Game Description Language called *Gamelan* designed to work with board games and card games. Gamelan has a focus on automated game design support, the language contains features for analyzing games based on the computational critics framework. It provides feedback for game elements and events that the designers can use based on their own perspective.

Nelson and Mateas [16] explored the human factors of Mixed Initiative systems. They presented a series of interviews with professional game developers to understand what were the most desirable features for the next generation of AI-game design assistants. Finally, Nelson also suggests the game itself as a source of metrics. His work presents strategies about how to extract valuable information from the relations between game elements, such as a rule set, before any playtest session [17].

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

B. Game visualization and telemetry systems

Game visualization and telemetry is a topic which has been gaining much attention in recent years [18]. Major game companies like *Unity* and *Bioware* have released their own solutions with specific features to work with visualization analysis, Unity Analytics [19] and Skynet [20], respectively. Due to telemetry techniques that are operating with large sets of data, developers can now have a precise visual information about their user behaviors [21], [22], including when their interest in playing is starting to decrease[23]. One of the many projects which is benefiting from the combination of telemetry and visualization is *Data Cracker*. A tool to collect, analyze and summarize data about player performance in one of the titles of the *Dead Space* franchise [24]. The game *Cure Runners* has a visualization system used to track the player and assist designers on level balance tasks. This work is a case study about the integration of an analytics tool into a commercial game [25]. Canossa et al. present visualization and event queries on a spatio-temporal interface. The UI allows users to select the game elements whose behaviors they want to track using raw data from gameplay sessions of VPAL a mod based on the popular franchise *Fallout* [26]. As with the AI-assisted tools, most of these visualization system are tightly coupled to individual games. The visualization package of Unity Analytics is a more general approach, but it does not have agents to play the games and collect data automatically.

C. Game replay systems

For some authors, like Bowman et al. [27], game replay systems are a subset for visualization techniques in videogames also know as retrospective analysis. The most common use for this kind of tool is training. For example, the work of Wallner and Kriglstein [28] uses replay sessions for training players of the multiplayer game *World of Tanks*. Another example is the work of Srisuphab et al. [29] in which they use the replay analysis for training players of the popular and millennial board game *Go*. However, the technique can also be used for development tasks, for example, the educational game *RumbleBlocks* uses a replay system for checking if the design is fostering behaviors that will lead to learning. All the cases presented are very attached to their game projects and allow little or even zero flexibility for being used outside of them.

D. Declarative game engines and query systems for games

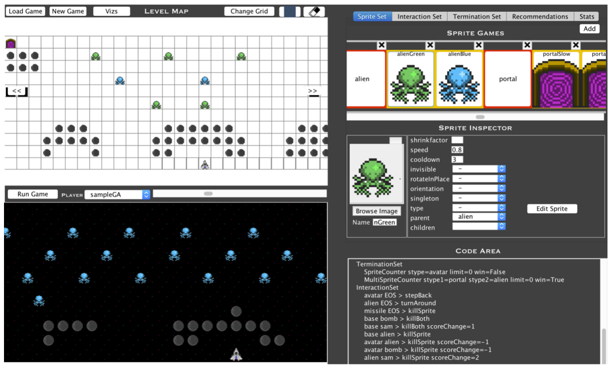
Declarative game engines expand the traditional idea of using databases only as a persistence solution. White et. al. [30] introduces the concept of *state-effect* patterns, a design pattern that allows game designers to develop part of their games declaratively. Deutch et. al. [31] describes the creation of a framework to perform data sourcing in games. It is based on SQL but extends it to allow probabilistic operations and recursive rule invocation. A more traditional use of databases can be seen on [29]. Here the authors store gameplay sessions of Go matches in a database. The goal is to use the stored matches to train novice players through a GUI.

Finally, [32] detail the process of implementing a bug tracker architecture in five different games. The infrastructure is concentrated on the main "game loop". The authors implemented specific lines of code that capture events of the games into a XML file. The bug finder process is based on properties about the expected behavior of the game expressed in a formal language. During runtime, a monitor observes the game and notifies the user when some property is violated.

We can see some examples of general methods in this section, for example Varvaressos et al. [32] architecture is applied to five games and got very impressive results. However, everything has to be made by hand by changing the main loop of the games, and it relies entirely on humans to find the bugs when playing.

III. The Cicero system

Cicero is a mixed-initiative tool for helping users with game prototyping tasks. There are two motivations behind Cicero. First is the creation of a *general* AI-assisted game design system, because existing tools nowadays are mostly limited to a single game and a single mode of design assistance. Second, we want to explore the question "Can AI make game development simpler?" in the specific terms of this work. Therefore our questions is: "Can AI make a human more accurate in game debugging tasks?". That said, while we benefit from the level of generality the GVGAI provides us, we know that such a generic "silver bullet" is still a difficult achievement [33]. At a minimum, we are contributing to the development of a more flexible tool, whose features can be applied to different games from different genres within the realm of 2D grid-logic games. Figure 1.



```
TerminationSet
   SpriteCounter stype=avatar limit=0 win=False
   MultiSpriteCounter stype1=portal stype2=alien limit=0 win=True
InteractionSet
   avatar EOS > stepBack
   alien EOS > turnAround
   missile EOS > killSprite
   base bomb > killBoth
   base sam > killBoth scoreChange=1
   base alien > killSprite
   avatar alien > killSprite scoreChange=-1
   avatar bomb > killSprite scoreChange=-1
   alien sam > killSprite scoreChange=2
```

Fig. 1. Cicero's main user interface.

A. *GVGAI & VGDL*

Cicero is based on the General Video Game AI framework (GVGAI) [34] and on the Video Game Description Language (VGDL) [35], [36]. GVGAI is a framework for general video game playing. It has an associated competition in which contestants submit their best AI agents, which are judged based on their performance of unseen games. VGDL is the language used to describe games in this framework; the language is compact, human-readable, and capable of expressing a large range of 2D games based on graphical logic. Among the kinds of games which can be described are adaptations of many classical games developed for the Atari 2600 and the Nintendo Entertainment System. Because of the popularity of the GVGAI competition, there are about 100 VGDL games available and several dozens of effective AI agents, with varying strengths on different types of games [37].
A VGDL game is written by specifying four description sets. The first one is used to describe the sprites and their behaviours. The second one describes the interaction rules, i.e. what happens when two sprites overlap each other. The third set describes the termination conditions, the rules which say if a player won or lose a game. There is one more set, used to associate sprites with symbols. This one does not influence the game rule description and game elements behaviors. The association is just a visual cue for making the graphics process simpler. See fig. 2.

```
SpriteSet
  hole   > Immovable color=DARKBLUE img=hole
  avatar > MovingAvatar
  box    > Passive img=box
InteractionSet
  avatar wall > stepBack
  box avatar  > bounceForward
  box wall    > undoAll
  box box     > undoAll
  box hole    > killSprite scoreChange=1
TerminationSet
  SpriteCounter stype=box    limit=0 win=True
```

Fig. 2. A Sokoban game written in VGDL

B. *Agent-based testing*

The availability of a simple, analytically tractable game specification language and of a sizable library of AI agents provides GVGAI with important benefits compared to standard game editors and engines such as *Unity*, *Unreal* or *GameMaker*. The AI agents – can play (to some extent) all games that can be specified in VGDL, what allow us to perform automatic gameplay and game testing.
The agents are versions of Graph-Search based algorithms like breadth-first, depth-first search and Astar, Genetic algorithms, and Monte Carlo Tree-Search. When a game is running, the agents simulate future states by using a forward model. Within the states generated by the model, the agents follow their heuristics in order to find the best action to take in the state they are. The use of a forward model is not mandatory, therefore algorithms which does not require it can also be implemented. Practically speaking, our system is flexible enough to allow that any kind of GVGAI agent, independent of the algorithm they are based on, can play the games. For the purposes of this paper, we are using a specific one, *Adrienctx*, a former champion of the GVGAI competition, able to play and perform well in several games and avoid problems associated to non-human players like jitteriness, useless moves and no long term planning at some extent [38]. Cicero offers users quick access to the agents available in the GVGAI framework and also allow that any new one can be uploaded to the tool. Thus, users can easily configure their games and choose a myriad of agents to simulate their games.

Because of the lack of uniformity in specification of games developed in other game engines, there are no general game-playing agents for arbitrary engines. So while, for example, Unity is a much more versatile and powerful game engine in general, the lack of general AI agents makes it impossible to develop a tool such as Cicero for that engine. The recent Unity ML Agents framework is taking some early steps towards making it possible to interface advanced AI with Unity, we are nowhere near having general-purpose game-playing agents in Unity and there are still some open questions like a lack of a fast simulation for instance.

C. *SeekWhence*

SeekWhence is a replay analysis tool. It allows users to replay a stored gameplay session played by an agent or a human player. They can navigate back and forth in the session step-by-step, frame-by-frame. In accordance with the definition of replay analysis of Bowman et al. [27], SeekWhence is a retrospective tool designed for game analysis tasks such as debugging and balancing. However, SeekWhence expands the concept of replay tools. It is not only a collection of frames, but also a collection of game states. This means that users can retrieve any game state and edit it by changing game elements and/or by picking other agents to play. The implementation of SeekWhence consists of a graphical interface written in Java with basic video controls for going back and forth in a stored gameplay session. To (re)create all the states first, we have to store all the information of a game state individually and index it by game ticks. Therefore you have a linked list whose nodes consists of the game ticks and the set of elements in a game state. We have published a paper at the Foundations of Digital Games 2017 Conference[39]. This paper contains all the details about the motivation and implementation of the SeekWhence infrastructure.

D. Playtrace aggregation

Playtrace aggregation is made available as a visualization system. We collect the positions of every game element (player, enemies, objects, etc.) and assign each a user-configurable color. We then aggregate every position of the objects throughout a play session as a heat map of locations around the level (Figure 3). It works in real time and can be used with SeekWhence for sequential analysis.

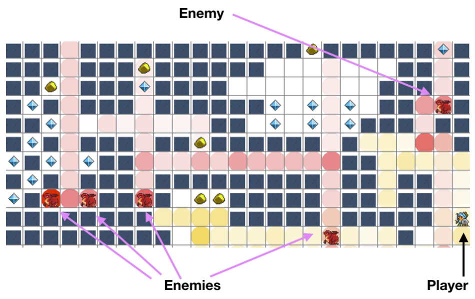


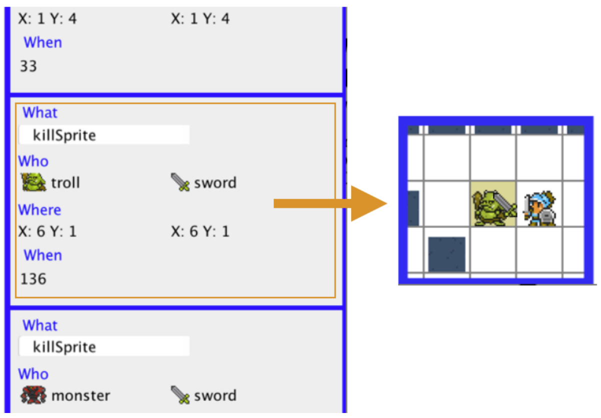
Fig. 3. Vizualization of playtrace aggregation. Red dots show how enemies are exploring a level. Yellow ones show how a player is exploring it.

E. Mechanics recommender

The Mechanics Recommender is a tool to recommend game mechanics whenever a user wishes. It is based on similarities between the descriptions of the game in development and the descriptions of available VGDL games. Therefore when a user asks for recommendations, the system performs a search in the whole library of VGDL games and returns mechanics that might be a good fit for the game a user is working on, letting up to them the final decision of what mechanic to incorporate. The recommender is designed to be a Creativity Supporting Tool in the inspirationalist school as described by [40]. The mechanics recommender makes suggestions that can help users accomplish specific goals while fostering their creativity at the same time. The similarities compare the parameter list of the sprites in the game a user is developing against each game in the VGDL repository. The comparison uses the Euclidean distance. Therefore, a ranking is created and those games in the repository which have the least distance to the user's game are used to provide the suggestions of future sprites to add with interactions between these sprites and the ones a user already have.

F. Kwiri

Kwiri is a query system motivated by the idea that a designer of a game will often want to figure out when and where something happens. The answers to these questions might not be evident when either playing a game or watching a replay. For example, imagine that a particular Non Player Character (NPC) occasionally dies even when not intersecting one of the player's bullets. To find out what is the cause of the problem, the game designer would have to play the game repeatedly until the event happened again, or watch a long replay session attentively. But what if they could simply ask the game when and where the NPC died, and who or what killed it? Kwiri makes use of the fact that the games available to Cicero have formally defined mechanics, and deliver the capacity to interrogate replays for particular combinations of events. Figure 4.


X: 1 Y: 4 X: 1 Y: 4
When
33

What
killSprite
Who
troll sword
Where
X: 6 Y: 1 X: 6 Y: 1
When
136

What
killSprite
Who
monster sword

Fig. 4. Kwiri shows the results of a query on a panel list. It contains information of the event, the elements involved, their positions and the frame number of when it happened. When clicking on a panel, it opens the replay tool (SeekWhence) and jumps exactly to the frame where the event took place, highlighting it with visualization cues.

Kwiri is an extension of SeekWhence and it is implemented in SQL and its GUI is written in Java. The users are able to input queries about any game element (player, NPCs, items, etc.), events (deaths, item pick-up, attacks, etc), frames and positions. After system retrieval, users can navigate around the data using *Seekwhence* features.

IV. User study

A quantitative research design was used to study user performance in different tasks using Cicero features in game debugging tasks.

A. Previous experiment

We conducted an informal experiment months ago before starting to write this paper. In that study, three members of our lab and a PhD student of our department took part in the experiment. We were studying *SeekWhence* and the visualization system. The users had the goal of listing the number of inconsistencies they found. We explained all of the available tools and let them freely use whichever they wanted to accomplish the task. Two things became evident; first, as the users were free to select an AI or to test the games themselves, we were not able to appropriately measure how good each method was independently. Second, we suspected that the users were taking too long to find inconsistencies with the visualization system. They were going back and forth several times with *SeekWhence* to confirm their guesses, and sometimes they gave up and provided incorrect answers. So we hypothesized that a query-based system could be more appropriate to filter and show game events to the users. This might keep users from having to waste too much time and provide wrong guesses. These two observations were our main motivation to expand the user study. Thus we created tasks to evaluate how accurate humans and AI are in game debugging tasks and what advantage query and a visualization systems can provide.

B. Pilot Study

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

C. Subjects

After all the kinks were worked out of our experimental design, we recruited 32 students from our university to take part in the main study. They were, approximately, 30% female and 70% male. 15% were masters students, 30% were graduate students and 55% percent were undergraduate students. All of them were enrolled in either the Computer Science or Digital Media departments. All subjects were recruited via their departments' email-lists. Those who completed the experiment were awarded a $10 USD Amazon Gift Card. Users took, on average, approximately 30 minutes to complete the experiment.

D. Research setting

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

E. Users Tasks

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

In all the tasks we were testing the hypothesis: humans perform better in game debugging tasks with AI assistance rather than without it. The independent variables of this study were the games used for each one of the tasks and the players (humans or AI). The dependent variable was the accuracy in detecting inconsistencies. The AI agent player used was the *Adrienctx* agent, it is implemented based on an open loop approach [41] and it is a previous winner of the GVGAI competition [34]. Therefore, able to perform well in different game challenges, and as stated in the subsection "Agent-Based Test" it avoids, at some extent, disturbing characteristics common to non-human players like jitterines and no long term planning.

1) Invincible Barriers: Human without AI assistance VS. Human with AI assistance: The first task was to find inconsistencies in the rules of our clone of the space-shooter game *Space Invaders*. There was only one inconsistency, barriers which could never be destroyed.

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

Humans were given special advantages that the AI alone did not have. For the AI, we ran the game just once. For the humans we allowed them to play as many times as they wanted for up to 80 seconds (more than 3x the time needed for the AI to finish its simulation). For both players (AI and humans) we changed the positions of the barriers in the level. The total number of indestructible barriers was 9 out of 39. During this task, the users played two versions of *Space Invaders*, a normal one and an extreme one. In the extreme version, we changed the parameters of the game to make enemies drop bombs constantly and to make the human player immortal. Thus, we made it easier for humans to find the inconsistency in the game in two separate ways. For both game types, normal and extreme, humans have more time to play, and in the extreme version they could not die. Figure 5.

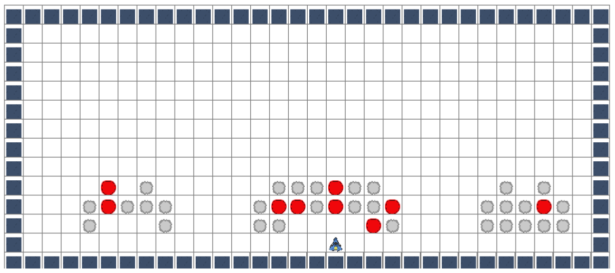


Fig. 5. Example of a Space Invaders level. The barriers in red are the inconsistencies the users should find.

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

3) F-Walls: Human without AI assistance VS. Human with AI assistance: For this task, we altered a clone of the *Zelda* cave levels. We removed the enemies and we planted fake walls (hence the name of this task: F-walls). The goal here was to identify how many fake walls subjects could find. We divided the participants in two groups, one started by playing by themselves while the other one started by watching an AI

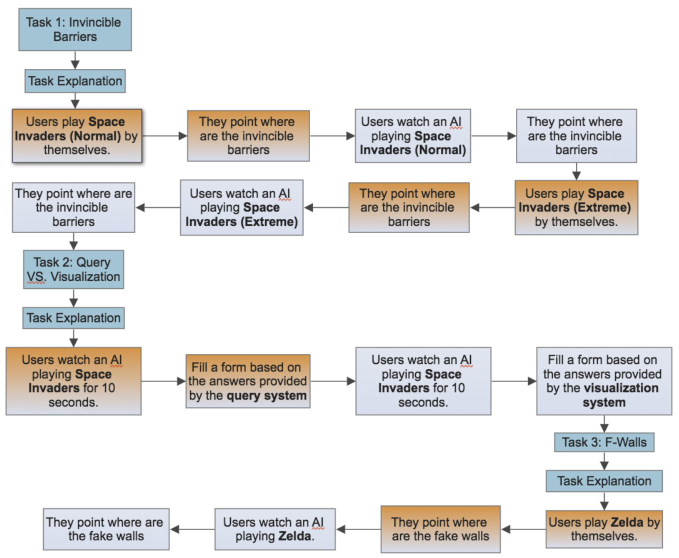

Task 1: Invincible Barriers
Task Explanation
Users play Space Invaders (Normal) by themselves.
They point where are the invincible barriers
Users watch an AI playing Space Invaders (Normal)
They point where are the invincible barriers
Users play Space Invaders (Extreme) by themselves.
They point where are the invincible barriers
Users watch an AI playing Space Invaders (Extreme)
They point where are the invincible barriers
Task 2: Query VS. Visualization
Task Explanation
Users watch an AI playing Space Invaders for 10 seconds.
Fill a form based on the answers provided by the query system
Users watch an AI playing Space Invaders for 10 seconds.
Fill a form based on the answers provided by the visualization system
Task 3: F-Walls
Task Explanation
Users play Zelda by themselves.
They point where are the fake walls
Users watch an AI playing Zelda.
They point where are the fake walls

Fig. 6. This flowchart shows the procedure taken by users of the group A. They started the first and third tasks without AI assistance, and started the second task by using the query system. By switching the orange blocks with the gray ones, we have the procedure taken by the group B.

playing. After they or the AI finished, they were asked to put red marks on the level to indicate the location of the fake walls. The users had 80 seconds to play while the AI simulated the game five times taking, on average, less than 40 seconds in total. So that users didn't to rely on their memories, which would negatively affect their performance, they were allowed to use the visualization system or SeekWhence. The query system was not an option since it does not capture events without explicit interactions between two sprites.

F. Procedure

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

For the second task (query vs. visualization), we explained to the users that we would have the AI play the games for ten seconds. Afterwards we would open a panel (the query or the visualization) and ask them to fill out a form based on what the panel was showing to them. Figure 6.

V. Results

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

A. Task 1: Space Invaders

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

1) Normal Case: To evaluate if humans with AI assistance were performing better than humans without assistance a paired samples t-test (one-tailed) was used to compare the scores of the groups. The result was statistically significant ($p = 1.002e-05$) and the AI assistance improved the accuracy of the users by almost three times (2.7x). The AI assistance also helped users avoid mistakes. The mean showed that while users without AI assistance were making 1.75 wrong choices, users with AI assistance made none. This result was significant with $p = 0.0038$. We also ran precision and recall tests for both groups. The results for the tasks with AI assistance show that it improves both precision (100%) and recall (˜82.29%) over the tasks without AI assistance (precision: ˜83.14%, recall: ˜52.08%).

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

B. Task 2: Query VS. Visualization

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

C. Task 3: F-walls

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

VI. Discussion

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

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

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

Despite many of the results confirmed our expectations, more tests are necessarily in order to achieve statistical significance for all the cases explored in this study. Concerning to efficiency, we suggest more work on agent customization. The dexterity presented by the agents evaluated in this paper is enough for playing games like Space Invaders in capabilities that overcome humans. However for finding inconsistencies in levels of a bigger size and complexity it would be necessary thousands (or more) simulations. Still, because the goal of the agent is beating the game, even with a huge amount of simulations there are no guarantees it would cover all the cases and finding all the inconsistencies expected by the developers. Therefore we encourage more research on designing general agents oriented to find game design flaws.

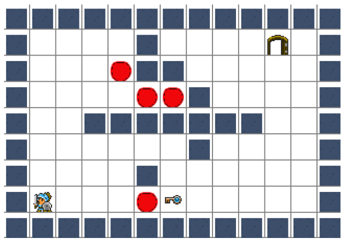


Fig. 7. Example of a Zelda level. In red, the positions of the fake walls that the user should find.

VII. Conclusion and Future Work

Cicero is a mixed-initiative system for game prototyping and debugging tasks. While many mixed-initiative tools are directly attached to a single game or a game genre, Cicero has features that can be applied to all games and genres within the GVGAI domain. Through a user study we found that in game debugging tasks, users perform better with AI assistance in games that require dexterity. We also found that in tasks where both a query and visualization system can be used, the query system leads users to be more accurate in their outcomes. We applied the same tool to two rather different games (clones of Space Invaders and Zelda) in the tests, thus demonstrating its generality. We suggest for future work in mixed-initiative tools for video games that new methods ought be made available across different game projects and be evaluated with user studies to measure their impact.

Acknowledgment

Tiago Machado is supported by the Conselho Nacional de Desenvolvimento Cientfico e Tecnologico (CNPQ), under the Science without Borders scholarship 202859/2015-0

References

[1] J. Blow, "Game development: Harder than you think," *Queue*, vol. 1, no. 10, pp. 28–37, Feb. 2004. [Online]. Available: http://doi.acm.org/10.1145/971564.971590

[2] E. Horvitz, "Principles of mixed-initiative user interfaces," in *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, ser. CHI '99. New York, NY, USA: ACM, 1999, pp. 159–166. [Online]. Available: http://doi.acm.org/10.1145/302979.303030

[3] J. E. Allen, C. I. Guinn, and E. Horvtz, "Mixed-initiative interaction," *IEEE Intelligent Systems and their Applications*, vol. 14, no. 5, pp. 14–23, Sep 1999.

[4] J. S. Brown, R. R. Burton, and F. Zdybel, "A model-driven question-answering system for mixed-initiative computer-assisted construction," *IEEE Transactions on Systems, Man, and Cybernetics*, vol. SMC-3, no. 3, pp. 248–257, May 1973.

[5] S. M. Lucas, M. Mateas, M. Preuss, P. Spronck, and J. Togelius, "Artificial and Computational Intelligence in Games (Dagstuhl Seminar 12191)," *Dagstuhl Reports*, vol. 2, no. 5, pp. 43–70, 2012. [Online]. Available: http://drops.dagstuhl.de/opus/volltexte/2012/3651

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

[7] N. Shaker, M. Shaker, and J. Togelius, "Ropossum: An authoring tool for designing, optimizing and solving cut the rope levels." in *AIIDE*, 2013.

[8] A. Liapis, G. N. Yannakakis, and J. Togelius, "Sentient sketchbook: Computer-aided game level authoring." in *FDG*, 2013, pp. 213–220.

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

[10] A. M. Smith, E. Andersen, M. Mateas, and Z. Popović, "A case study of expressively constrainable level design automation tools for a puzzle game," in *Proceedings of the International Conference on the Foundations of Digital Games*. ACM, 2012, pp. 156–163.

[11] E. Butler, A. M. Smith, Y.-E. Liu, and Z. Popovic, "A mixed-initiative tool for designing level progressions in games," in *Proceedings of the 26th annual ACM symposium on User interface software and technology*. ACM, 2013, pp. 377–386.

[12] A. Isaksen, D. Gopstein, J. Togelius, and A. Nealen, "Exploring game space of minimal action games via parameter tuning and survival analysis," *IEEE Transactions on Computational Intelligence and AI in Games*, vol. PP, no. 99, pp. 1–1, 2017.

[13] A. M. Smith, M. J. Nelson, and M. Mateas, "Prototyping games with biped," in *Proceedings of the Fifth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, ser. AIIDE'09. AAAI Press, 2009, pp. 193–194. [Online]. Available: http://dl.acm.org/citation.cfm?id=3022586.3022624

[14] ——, "Computational support for play testing game sketches." 2009.

[15] J. C. Osborn, A. Grow, and M. Mateas, "Modular computational critics for games," in *Proceedings of the Ninth AAAI Conference on Artificial Intelligence and Interactive Digital Entertainment*, ser. AIIDE'13. AAAI Press, 2014, pp. 163–169. [Online]. Available: http://dl.acm.org/citation.cfm?id=3014712.3014738

[16] M. J. Nelson and M. Mateas, "A requirements analysis for videogame design support tools," in *Proceedings of the 4th International Conference on Foundations of Digital Games*, ser. FDG '09. New York, NY, USA: ACM, 2009, pp. 137–144. [Online]. Available: http://doi.acm.org/10.1145/1536513.1536543

[17] M. J. Nelson, "Game metrics without players: Strategies for understanding game artifacts," in *Proceedings of the 19th AIIDE Conference on Artificial Intelligence in the Game Design Process*, ser. AIIDE'11-19. AAAI Press, 2011, pp. 14–18. [Online]. Available: http://dl.acm.org/citation.cfm?id=3014786.3014789

[18] M. S. El-Nasr, A. Drachen, and A. Canossa, *Game Analytics: Maximizing the Value of Player Data*. Springer Publishing Company, Incorporated, 2013.

[19] "Unity technologies. unity game engine," https://unity3d.com, 2017, accessed: 2017-03-01.

[20] G. Zoeller, "Development telemetry in video games projects," in *Game developers conference*, 2010.

[21] P. Coulton, W. Bamford, K. Cheverst, and O. Rashid, "3d space-time visualization of player behaviour in pervasive location-based games," *International Journal of Computer Games Technology*, vol. 2008, p. 2, 2008.

[22] Y.-E. Liu, E. Andersen, R. Snider, S. Cooper, and Z. Popović, "Feature-based projections for effective playtrace analysis," in *Proceedings of the 6th international conference on foundations of digital games*. ACM, 2011, pp. 69–76.

[23] T. Mahlmann, A. Drachen, J. Togelius, A. Canossa, and G. N. Yannakakis, "Predicting player behavior in tomb raider: Underworld," in *Computational Intelligence and Games (CIG), 2010 IEEE Symposium on*. IEEE, 2010, pp. 178–185.

[24] B. Medler *et al.*, "Generations of game analytics, achievements and high scores," *Eludamos. Journal for Computer Game Culture*, vol. 3, no. 2, pp. 177–194, 2009.

[25] G. Wallner, S. Kriglstein, F. Gnadlinger, M. Heiml, and J. Kranzer, "Game user telemetry in practice: A case study," in *Proceedings of the 11th Conference on Advances in Computer Entertainment Technology*, ser. ACE '14. New York, NY, USA: ACM, 2014, pp. 45:1–45:4. [Online]. Available: http://doi.acm.org/10.1145/2663806.2663859

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

[27] B. Bowman, N. Elmqvist, and T. Jankun-Kelly, "Toward visualization for games: Theory, design space, and patterns," *IEEE transactions on visualization and computer graphics*, vol. 18, no. 11, pp. 1956–1968, 2012.

[28] G. Wallner and S. Kriglstein, "Visualizations for retrospective analysis of battles in team-based combat games: A user study," in *Proceedings of the 2016 Annual Symposium on Computer-Human Interaction in Play*, ser. CHI PLAY '16. New York, NY, USA: ACM, 2016, pp. 22–32. [Online]. Available: http://ezproxy.library.nyu.edu:2595/10.1145/2967934.2968093

[29] A. Srisuphab, P. Silapachote, T. Chaivanichanan, W. Ratanapairojkul, and W. Porncharoensub, "An application for the game of go: Automatic live go recording and searchable go database," in *TENCON 2012-2012 IEEE Region 10 Conference*. IEEE, 2012, pp. 1–6.

[30] W. White, B. Sowell, J. Gehrke, and A. Demers, "Declarative processing for computer games," in *Proceedings of the 2008 ACM SIGGRAPH symposium on Video games*. ACM, 2008, pp. 23–30.

[31] D. Deutch, O. Greenshpan, B. Kostenko, and T. Milo, "Declarative platform for data sourcing games," in *Proceedings of the 21st International Conference on World Wide Web*, ser. WWW '12. New York, NY, USA: ACM, 2012, pp. 779–788. [Online]. Available: http://doi.acm.org/10.1145/2187836.2187942

[32] S. Varvaressos, K. Lavoie, A. B. Massé, S. Gaboury, and S. Hallé, "Automated bug finding in video games: A case study for runtime monitoring," in *Software Testing, Verification and Validation (ICST), 2014 IEEE Seventh International Conference on*. IEEE, 2014, pp. 143–152.

[33] S. D. Fraser, F. P. Brooks, Jr., M. Fowler, R. Lopez, A. Namioka, L. Northrop, D. L. Parnas, and D. Thomas, ""no silver bullet" reloaded: Retrospective on "essence and accidents of software engineering"," in *Companion to the 22Nd ACM SIGPLAN Conference on Object-oriented Programming Systems and Applications Companion*, ser. OOPSLA '07. New York, NY, USA: ACM, 2007, pp. 1026–1030. [Online]. Available: http://doi.acm.org/10.1145/1297846.1297973

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

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

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

[37] P. Bontrager, A. Khalifa, A. Mendes, and J. Togelius, "Matching games and algorithms for general video game playing," in *Twelfth Artificial Intelligence and Interactive Digital Entertainment Conference*, 2016.

[38] A. Khalifa, A. Isaksen, J. Togelius, and A. Nealen, "Modifying mcts for human-like general video game playing," in *Proceedings of the Twenty-Fifth International Joint Conference on Artificial Intelligence*, ser. IJCAI'16. AAAI Press, 2016, pp. 2514–2520. [Online]. Available: http://dl.acm.org/citation.cfm?id=3060832.3060973

[39] T. Machado, A. Nealen, and J. Togelius, "Seekwhence a retrospective analysis tool for general game design," in *Proceedings of the 12th International Conference on the Foundations of Digital Games*, ser. FDG '17. New York, NY, USA: ACM, 2017, pp. 4:1–4:6. [Online]. Available: http://doi.acm.org/10.1145/3102071.3102090

[40] B. Shneiderman, "Creativity support tools: Accelerating discovery and innovation," *Commun. ACM*, vol. 50, no. 12, pp. 20–32, Dec. 2007. [Online]. Available: http://doi.acm.org/10.1145/1323688.1323689

[41] D. Perez Liebana, J. Dieskau, M. Hunermund, S. Mostaghim, and S. Lucas, "Open loop search for general video game playing," in *Proceedings of the 2015 Annual Conference on Genetic and Evolutionary Computation*, ser. GECCO '15. New York, NY, USA: ACM, 2015, pp. 337–344. [Online]. Available: http://doi.acm.org/10.1145/2739480.2754811