The AIIDE-17 Workshop on
Experimental AI in Games
WS-17-19

Towards General RPG Playing

Joseph C. Osborn
Computational Media
University of California, Santa Cruz
1156 High St
Santa Cruz, California 95064

Ben Samuel
Computer Science
University of New Orleans
2000 Lakeshore Dr
New Orleans, Louisiana 70148

Adam Summerville, Michael Mateas
Computational Media
University of California, Santa Cruz
1156 High St
Santa Cruz, California 95064

Abstract

General videogame playing has come a long way in a short period of time, but remains at the level of solving relatively short games made up of distinct and isolated episodes. Even simple console role-playing games (RPGs) are far beyond the reach of current techniques, requiring the synthesis of cultural knowledge with compositional reasoning over several interconnected sub-games. We explore how the challenges of playing these games could spark new advances in compositional analysis of games and common-sense reasoning.

General RPG playing can leverage advances in episodic general game playing and in areas like text understanding, image classification, and automated game design learning. It has direct applications in design support and AI-based game design, and the techniques used to enable it could generalize to other families of games such as adventure, open-world, and simulation games. In this paper, we describe the motivation behind general RPG playing in a sub-domain of Nintendo Entertainment System (NES) RPGs, some promising approaches to some of its fundamental issues, and immediate next steps; we conclude by describing a few concrete benchmark problems on the path towards automated play of these complex games.

Introduction

General (video)game playing is an established research area with several active threads; as a natural stepping-stone to artificial general intelligence it is an attractive target for academic and industrial research labs alike (Genesereth, Love, and Pell 2005; Perez-Liebana et al. 2016a; Bellemare et al. 2015). Human-like (or at least strong) general game-playing AI also seems necessary for many approaches to automated game design support and game generation (Smith 2013; Barros et al. 2015). Unfortunately, these systems are limited to *episodic* games for which planning can succeed only over relatively short time horizons (minutes at most, or a few hundred turns in discrete games), even with considerable computational resources. Longer games are inaccessible to these techniques unless they are split into smaller episodes, but this splitting is generally *ad hoc* and game-specific; moreover it has not been shown for games with complex internal structure. Agents can do well in situations like a *DOOM* deathmatch because each individual encounter with an enemy is more or less independent, and the high-level structure of moving between firefights and looking for powerups is extremely simple. Moreover, both types of play share the same basic input modalities, short-term rewards, and so on.

Successful game-playing agents generally learn policies for maximizing their probability of victory, but rarely recover knowledge about a game's (implicit) rules or high-level design features. There are some exceptions in the domains of general game playing (Clune 2007) and general videogame playing (Perez-Liebana et al. 2016b). The automated play of specific, individual games can in theory be achieved using approaches from expert systems or deep Q-learning, but these might fail to generalize to new games or even to variations on the same game; classical search-based approaches often fail to scale to longer games without a prohibitive amount of domain knowledge. On the other hand, recent research in schema networks seems like a promising approach which reinforces the notion that we must learn design information explicitly in order to transfer knowledge and skills across games (Kansky et al. 2017). Automated game design learning has been recognized as an important problem with many application areas including general game playing (Osborn, Summerville, and Mateas 2017).

Real-time strategy games comprise an active area of research for game-playing agents generally optimized for specific games (Ontanón et al. 2013). Given the successes of RTS AI, we suggest exploring the general play of a different genre: console role-playing games (RPGs) like *Dragon Warrior* or *Final Fantasy*. These games are typified by highly compositional design relying extensively on both cultural knowledge and numeric reasoning over probabilistic combat models, exercising different aspects of commonsense reasoning than existing RTS or arcade game players. By compositional design, we mean a game design consisting of

many distinct game systems with significant interactions between systems (e.g., exploration, combat, item economies, and plot progression). These games have many iterations, sequels, hacks, and even randomizers, yielding a substantial corpus whose members share some similarities but also introduce game-specific concepts, systems, and user interfaces. We can usefully investigate problems of compositional game understanding and transfer in the special case of general RPG playing.

General RPG Playing

To address compositional game understanding, it is important to identify the sub-components (or sub-systems) that, when combined, form what a human player would likely identify as a role playing game. Breaking down RPGs into these constituent parts is an important first step in determining an effective knowledge representation. Moreover, this process provides a clear (if somewhat naive) blueprint for accomplishing our goal of developing a general RPG player, as this goal can be broken down into creating game players for each of the game's sub-systems. These sub-systems do not operate in isolation, but inform and communicate with each other; we hope to leverage previous work in RTS playing where sub-component managers cooperate to achieve a shared goal (Weber, Mateas, and Jhala 2010), lifting this to the level of learned goals and systems.

At a high level, many NES-era RPGs take the form of sweeping epics where players control an individual or a small party to overcome impossible odds and defeat the forces of evil. RPGs typically involve exploring the world, conversing with Non-Player Character (NPC) townsfolk and royalty, battling countless monsters or named villains in both random and scripted combat encounters, and applying the spoils of war towards the acquisition of equipment that assists in subsequent fights. Though each game has its own idiosyncrasies, several components seem to be shared by most games in the genre:

Movement: RPG game worlds are typically represented as a series of tile-based maps. Each tile has a variety of properties, including: their appearance; whether the player can walk through them; whether they might trigger a fight (the tile's *safety*); or if they are a *link* to another map. Some tiles, often grasslands or deserts, are passable by default, while others can only be traversed when the player owns a certain item or is riding a vehicle (e.g., *Final Fantasy's* oceans can be traversed by ship, and rivers by canoe), and still others are completely impassable. Some tiles transition the player from one map to another (e.g., stepping on a tile that looks like a picture of some buildings opens a new map, with a new tileset, that represents the town). Individual maps are often named, and successfully remembering the locations of these maps—and their relative positioning to each other—is integral for solving puzzles and advancing the plot.

Combat: Combat in NES-era RPGs generally occurs in a separate interface from movement, where battle actions are selected through menu navigation (covered in more detail below). Combat is essentially a game of resource and state management. The player's characters typically have a numeric representation of their life force (often written as *HP* for *Health/Hit Points*) that when depleted puts them in a *knocked out* state; if the entire party is knocked out, the game typically ends and the player must restart from a set location or a saved game. Most characters also have *MP* or *Magic Points* that determine how many times they can use special abilities, though the nature of this quantity is game-specific. For instance, in *Final Fantasy*, a character has eight MP values, where each value signifies the number of times they can cast a spell of the corresponding rank; in *Dragon Warrior*, MP is a single quantity that decreases by a set amount based on the cost of the spell cast, with more powerful spells costing more MP. Tools from numeric transition systems might be appropriate to apply here, for example by synthesizing integer programs approximating the combat simulation; we might also be able to learn the effects of specific combat actions by Bayesian inference.

There are also attributes like attack power and magic power which affect a character's combat prowess, that can be bolstered through purchased and looted equipment. Though typically higher numbers are better, some games involve characters that are meant to specialize in particular attributes. Besides permanent equipment, there are often consumable items that adjust resources and states (e.g., items that restore HP or MP, restore a party member with the *knocked out* status to fighting condition, etc.). To complicate things further, most games have non-combat items or abilities that indirectly affect the sub-game of resource management, such as the *Dragon Warrior* series' *Wing of the Wyvern* item which teleports the player to a previously-visited town and its available shops and amenities.

Menus: Though menus are an important aspect of combat, many other systems rely on menu navigation as well. Most RPGs include shops where equipment can be purchased with coin accrued through combat; these shops are almost always menu-based (some exceptions have the player bump into the item they wish to purchase).

Navigating menus is often a prerequisite to playing the game. Some games involve choosing a save slot (e.g., *Dragon Warrior*), while others do not (e.g., *Final Fantasy*). Many involve naming the character(s) that comprise the party. Some games ask the player to determine the make-up of the party, which has significant ramifications for future gameplay decisions. In *Final Fantasy*, the party makeup is determined at the beginning of the game, while in *Dragon Warrior 3* the number and type of characters can be changed during play at a specific location, and in *Final Fantasy 3* the party members' character classes (and thus the party makeup) can change at essentially any time outside of battle. Creating a viable party, purchasing and equipping equipment, and surviving combat are all strictly necessary for the success of a general RPG playing agent.

NPC Interaction: Outside of menu-based shops, most NPC interaction typically involves *speaking* to characters, which brings up a panel with text representing the lines of dialogue uttered by the character. Unlike many modern RPGs—where dialogue with NPCs often involves selecting choices from a dialogue tree—most NES-era RPGs afford players little opportunity to direct the flow of conversation, besides the occasional binary choice.

However, even if players cannot speak in these games, it is vital that they listen. NPCs will often tell the player where to go next; speaking to (and listening to) NPCs is often the difference between aimless wandering and goal directed play for human players. An effective general game player would seem to require some amount of natural language understanding, and ability to semantically link key phrases mentioned by NPCs (location names, items, etc.) to its understanding of the game world (e.g., dungeon and town locations). Interestingly, *Final Fantasy 2* treats some key phrases as items that must be obtained from conversation and used on other characters later; some other games highlight key terms in dialog with differently colored text.

While RPG systems interact strongly over both short and very long time horizons, they also have many symmetries and an AI player must be able to abstract over these. Navigation is the most obvious source of symmetries, but many inventory actions and combat outcomes are reversible: each combat is independent in some sense but variables like current health are carried over, so strategies have to change based on the whole initial state of the combat. On a longer time-scale, going to an inn essentially resets combat-related statistics, so in some sense an episode has ended; but now the player characters might be in a different location, or they may have become stronger by purchasing new equipment or gaining character levels. From some perspectives, events are independent; but from others, they are deeply interconnected. Human players can readily shift between these registers to achieve their goals.

RPGs are composed of a large number of these complex pieces; fortunately, the interactions between pieces are along well-defined boundaries even if these boundaries are game-specific. General RPG playing can be seen as the project of automatically decomposing a game into subsystems from fixed categories and determining the interactions between these components to inform mechanisms like hierarchical planning. In the specific case of RPG playing, we can lean on genre conventions to limit the scope of possible *types* of systems (though each game has its own specific designs of these system types), which in turn puts an upper bound on the number of distinct sorts of ways they can be combined. Taken together, this can be a useful first step on the way to fully compositional reasoning over games in general. By learning these pieces individually, we hope to create a player capable of progressing through many distinct RPGs that leverage them.

An RPG-playing agent must read on-screen text and recognize (or learn the meanings of) words like *health*, *enemy*, or *gold*—and connect these to resource economies and progression systems. It has to understand that a monster with a fiery appearance or located in a dungeon with a volcano-like appearance is likely to be weak against spells and abilities that connote wetness or coldness—and connect this to the choice of equipment and attacks in combat. It must talk to and understand non-player characters to obtain necessary information to continue in the game, for example to learn boss enemies' weaknesses or the locations of key items—and tie this into combat and navigation. Recognizing explicit instructions seems hard enough; oblique hints and analogic reasoning (for example, graphical hints in dungeons' puzzles) seem extremely challenging. It also seems likely that *transfer* between games is important—games in the same series generally build on prior games' concepts, and there are similarities across series as well (enemies can have different type affinities, with fire-type enemies being weak against water, and so on). Does it matter what order an AI plays games in?

Game Mode	*Overlay detection**
	State classification (map/combat/etc)
Movement	*Map-making**
	Tile properties
	Location names
Combat	Available actions
	Equations of combat
	Distinct character states
	Enemy properties
Inventory	*Menu navigation**
	Inventory structure
	Equipment and character statistics
	Using/equipping items appropriately
	Planning based on inventory/character statistics
Dialogue	Activating NPC dialog
	Questions with immediate effects
	Questions with deferred effects
Puzzles	Walkthrough-guided solving
	Spatial juxtaposition puzzles
	One-off/puzzle-like combat interactions
Progression	Recognizing plot-advancing moments
	Returning to previously visited locations when plot flags change
Cultural Knowledge	Bootstrapping with statistic, item, or equipment names
	Recognizing likely enemy weaknesses
	Understanding hints in NPC dialog

Table 1: Subproblems and steps towards general RPG playing. Tasks addressed in this paper are marked with *.

Finding solutions to all these problems is not only necessary for general RPG playing, but might shed light on other questions of particular interest in commonsense reasoning and could even inform the design of RPGs in the future. We summarize this section in Table 1.

Progress So Far

For now, we restrict our attention to RPGs on the Nintendo Entertainment System (NES); this is mainly because there is existing work in automatically extracting high-level design knowledge from games on this system (Summerville et al. 2017; Summerville, Osborn, and Mateas 2017). We propose a seed corpus of eight highly notable games: *Final Fantasy* and its two sequels, *Dragon Warrior* and its three sequels, and *Mother*. Enthusiasts have written *randomizers* for *Final Fantasy* and the first two *Dragon Warrior* games; these tools produce random variants which reshuffle numeric statistics and dungeon layouts. Several of these games

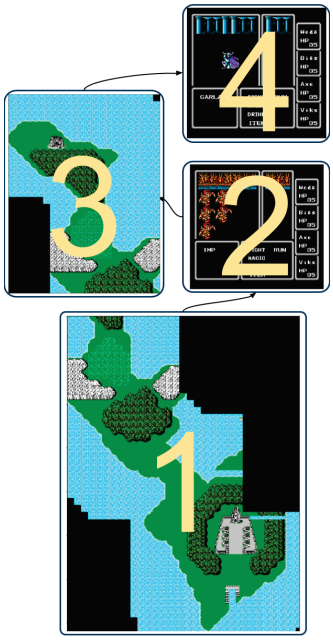

Figure 1: Automatically extracted map from *Final Fantasy*. Note that rooms 1 and 3 should be merged.

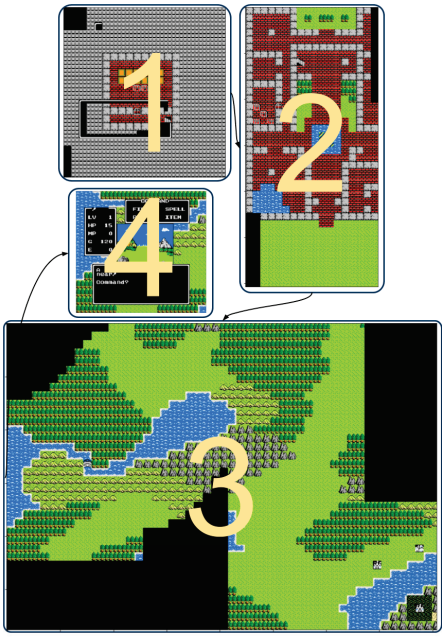

Figure 2: Automatically extracted map from *Dragon Warrior*. Note UI elements that appear in room 1.

also have manually-constructed variants (*ROM hacks*) which are essentially different games written for the same engine. In this paper, we pay special attention to *Dragon Warrior* and *Final Fantasy*.

Many of the tasks described here have the flavor of computer vision, even if they are not always done from raw pixels. We see the work so far—building maps and identifying screen transisions, detecting the windows and text of the game's user interface, and so on—as lifting us from the level of pixels up to a semantic domain where work on compositional game reasoning can begin in earnest.

Automatic Mapping

Mappy, a recent open-source project in extracting level design information from NES games, shows initial results on pulling both room-level maps and high-level links between maps from action-adventure games (Osborn, Summerville, and Mateas 2017). Running *Mappy* with manually-tuned parameters obtains pretty good results for *Final Fantasy* and *Dragon Warrior*, but tuning these parameters automatically remains important future work.

Mappy does not currently cluster similar rooms together automatically, it does not aggregate information across multiple playthroughs, and it does not recognize that multiple rooms might actually be part of one larger room (see Fig. 1). We see these as instances of essentially the same problem. In general, we feel that there are two questions of interest that help decide whether two similar rooms are in fact the same. First, do their tilemaps and sprite placements mostly agree in the parts where the rooms seem to overlap? This is the most obvious notion of similarity and the one which Mappy already uses to recommend similar rooms for merging. Second, we can also consider the links between these rooms and other rooms: If both rooms have three exits and these all take the player to the same or similar-looking rooms, that is strong evidence that both rooms are the same room.

Overlay Detection

The bigger problem for Mappy is that game *menus* often look like screen transitions or on-screen tiles (as witnessed in the Mappy paper's *Zelda* examples and room 1 of Fig. 2); so we must augment Mappy with a way to distinguish simulated space versus a discrete menu. One way of knowing what sort of screen the game is in might be to notice when the object being controlled changes position and appearance suddenly and starts following different physical rules—for instance, teleporting around rather than moving with an animation, or moving on a different grid from before, or looking like a finger or triangle instead of an animating humanoid. But there might be menus or other screen overlays like dialogue boxes at times when the player is still controlling the same character; in such cases we would not want to interpret menu content as being part of the map. We therefore need to rely on computer vision to detect overlays like menus and message boxes, considering both the currently visible screen and previously known information about the map and game world. This is complicated by the fact that menus can nest, overlap, and occlude each other.

Humans distinguish overlays from the map easily, perhaps thanks to clues like thick borders, low-complexity interiors, and the presence of text. These kinds of features are readily detected by off-the-shelf algorithms, so we developed a quick prototype of a tool to detect rectangular overlays covering up parts of an NES game's display. We first con-

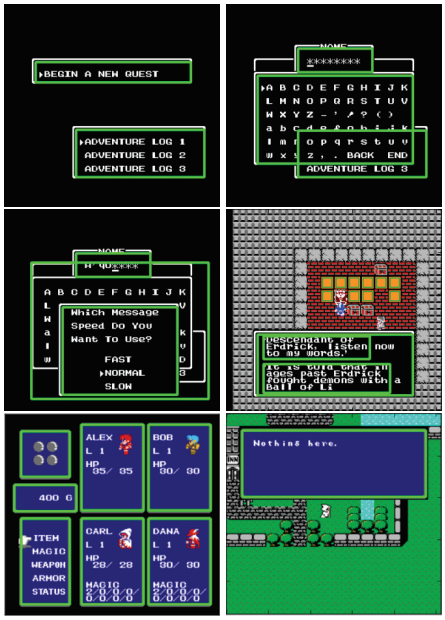

Figure 3: Overlay detection in *Dragon Warrior* (top four images) and *Final Fantasy* (bottom two images). Green rectangles show *current* overlays (live overlays that are covered up are tracked but not shown). Note false positives in the middle two images.

vert the display to grayscale and blur the image. Next we erode the image to paper over noisy lines, which are often caused by text or cursors overlapping the border. We then use OpenCV's contour detection algorithm, looking for contours which are well-approximated by rectangles and which are neither too small, too large, nor too redundant to other detected rectangles. We call any such rectangle which has been in the same spot for more than 30 frames *current*.

Since overlays can overlap, the appearance of a new overlay might cause the above algorithm to miss an overlay which is partially or completely hidden. We solve this by storing a list of *live* overlays from frame to frame, adding overlays when they become current, and removing an overlay if it is not current *and* it is not overlapped by any more recently found (and thus visually higher) overlay. This process (shown in Fig. 3) can still produce false positives, but these are generally harmless with respect to mapping and could be resolved in the future by looking at containment relationships or being more stringent about the presence of a border. The more dangerous type of false positive is recognizing a piece of level structure as an overlay; but this case can be easily avoided by retroactively discarding such false overlays if they are seen to scroll smoothly with the rest of the world. Given this overlay recognition algorithm, Mappy could be extended such that it does not consider tiles close to or contained within an overlay to be observed for the purpose of mapping.

Which Message	Nhich Nassaga
Speed Do You	Spaad Du Van
Want To Use?	Man: In usau
FAST	FASI
▷ NORMAL	yNuRNAL
SLOW	SLDN

Figure 4: A representative menu choice from *Dragon Warrior* (left) and its contents as interpreted by Tesseract (right).

Text Recognition

A large amount of information in RPGs is communicated via text. Unlike in traditional parser games, this text is given as an image, not as a string of characters. As such, Optical Character Recognition (OCR) is required to turn the image-based text into a more usable form. We have tried using Tesseract (Smith 2007), a popular open source OCR engine first developed by Hewlett-Packard Laboratories in 1985 and under the care of Google since 2006. While one of the best OCR engines available, it is apparently poorly suited out-of-the-box for NES RPGs (see Fig. 4).

This indicates that the low-resolution square font is poorly recognized. It is recognized as text, but with numerous mistakes (e.g., *e* becomes *a*). Tesseract allows for fine tuning of their model via additional training, so it might be possible to use representative low-resolution fonts to tune the model to better recognize NES-style text.

Cultural Knowledge

We also have to consider a particularly hard question, and one somewhat unique to adventure and role-playing games: where does cultural knowledge about game content come from? Many problems around figuring out what text and images mean or where objects are hidden in the world can be solved through brute force or learned via directed experimentation, but we would prefer our agent to play as much like a person as possible. One workaround might be to give the agent the information a walkthrough of the game or instruction manual might provide: what the plot events are, how to trigger them, what the strengths and weaknesses of enemies and weapons are, and so on (this proved to be helpful for automated play of *Civilization 2* (Branavan, Silver, and Barzilay 2011)). A lighter touch would be to give the equivalent of a translation guide: meanings of key words in the game plus only the most vital information that in-game text provides, in a machine-legible format. Learning concepts from a corpus of fantasy novels or fairy tales (or walkthroughs of arbitrary and perhaps unrelated games) might also help an agent recognize the valence and significance of terms like *dragon*, *swamp*, or *king*.

We can also consider giving this agent a human *coach*. If the agent can communicate some of its state to a human observer, and the observer has a way to influence the agent's learned structures, attention, or portfolio of approaches, then a human could resolve cultural issues on demand as they arise. The agent could be playing at much faster than human speed, notifying the coach when it sees some unfamiliar

object or interaction and providing its initial interpretation; this could be corrected manually, or conversely the coach would interrupt the agent with topical information about its next task. Of course, this is of limited utility for fully general game playing unless we allow and implement transfer learning between games or store the results of this process in some database available to general RPG playing agents; this coaching could be seen as a way to capture domain knowledge on an as-needed basis.

Next Steps

For agents without any knowledge of how games are structured, even a game's title screens might be challenging to get through. As mentioned earlier, an agent might need to begin by selecting a save slot or choosing to start a new game, selecting party composition, entering names, observing introductory cutscenes, and so on. It might be that random inputs could eventually get through these screens, but they actually embody several gameplay tasks that are important to understand. An initial step towards general RPG playing might be simply to get the game started; of course, choices around party makeup might be essentially arbitrary until the game's rules are better understood, but perhaps we could ask the agent to begin a game with a given target party composition and character names, or else have it report that this cannot be done before the game proper starts. Classifying the purpose of each screen the game presents on the way to playing the game seems useful and important, and could help the agent recover from game-over states if these return the player to the title screen. Here, one baseline would be random input, and the target might be to match or come close to the performance of game-specific hand-authored policies.

Once our agent is in the game and can control an avatar, we might want to automatically construct a menu hierarchy with some initial guesses about the purpose of each menu item; this can be seen as generating part of an instruction manual for how to play the game. We might also ask the agent to determine what characters are in the party, what resource pools and items might be available, and so on. We can compare these learned structures against ground truth.

A natural next step is to determine which character on screen we are controlling, and when we are exploring versus when we are in combat; here, the goal might be to explore as much of the world as possible before dying, ultimately producing a map whose size and accuracy can be used to evaluate the agent. If we know when we are in a menu, navigating on the map, in combat, or in other modes, we can also use specialized algorithms to play through those segments. We might choose to admit saving and restoring emulator states, essentially allowing the agent to perform backtracking search via time travel (perhaps learning rewards that maximize overall exploration of the rest of the game). This could bootstrap combat AI, but of course we could also approximate the underlying rules of combat by learning probabilistic models under a Markov assumption and then plan based on that representation. The latter approach would help us communicate across parts of the game, suggesting when our agent should heal up or buy new equipment—decomposition of the combat task itself is explored in (Tremblay, Dragert, and Verbrugge 2014), which calculates an optimal joint agent target selection policy as a component of a larger combat AI. Strategy game playing is a very active area of research; if we can automatically decompose an RPG into navigation, combat, and resource allocation we may obtain good results quickly by leveraging existing agents.

We can construct similar tasks for learning the rules of combat, equipping characters according to some criteria in the first town, or obtaining a plot clue from non-player characters. In this way we can eventually build up to achieving progression objectives and completing the game. Once the fundamental decomposition of the game into component sub-games is finished, we can start to improve on individual tasks in isolation, and compare performance (and game structure) of an agent across different games in the corpus.

The human coaching described earlier could form the foundation of interesting types of games: the player-as-coach or the player-as-environment. This is an exciting application area for AI-based game design (Eladhari et al. 2011), but one that has only begun to be explored. The God-game *Black & White* had players train a gargantuan creature to serve as a proxy for their divine will; by praising and punishing its behaviors, the creature eventually developed an understanding of how (the player wanted it) to engage with the world. In the real-time strategy games *Majesty* and *My Life as a King* players construct a fantasy kingdom to attract adventurers whom the player indirectly controls and nurtures via incentives. *Breath of Fire III* features a brief scenario where players train an NPC through combat; the player must experiment with a variety of techniques to elicit both offensive and defensive reactions from the NPC.

We imagine that such player-as-coach games would be evocative of helping a schoolyard friend or sibling get past a difficult battle or tricky puzzle in a game. By leveraging game and cultural literacy that the system lacks, the player's effort is rewarded not only by witnessing the AI overcoming its hardship, but in the unique pleasure that comes from sharing understanding, and from crafting an experience for another (Samuel et al. 2016). Both as a central component and as a peripheral side-game, player-as-coach has shown itself to be an engaging area for AI-based game design, and we hope this work enables it to be explored more fully.

Ultimately, we believe that by investing in high-level reasoning over game and goal structures, and by extracting design features from games in order to formulate those goals, we can lift episodic general game playing into the long-form case. These are real-world games with quirks, bugs, and cultural cachet; while the path to solving them is visible, it is not totally clear. The problems we must handle to address the concrete case of general RPG playing are also problems for other types of games and other domains besides general game playing including game design support, automatic tutorialization and adaptive difficulty adjustment, and game generation. All this makes general RPG playing an attractive and interesting problem, and we look forward to working with the larger game AI community towards solving it.

References

Barros, G. A.; Togelius, J.; Nelson, M. J.; et al. 2015. Towards generating arcade game rules with vgdl. In *Computational Intelligence and Games (CIG), 2015 IEEE Conference on*, 185–192. IEEE.

Bellemare, M.; Naddaf, Y.; Veness, J.; and Bowling, M. 2015. The arcade learning environment: An evaluation platform for general agents. In *Twenty-Fourth International Joint Conference on Artificial Intelligence*.

Branavan, S.; Silver, D.; and Barzilay, R. 2011. Nonlinear monte-carlo search in civilization ii. In *Twenty-Second International Joint Conference on Artificial Intelligence*. AAAI Press/International Joint Conferences on Artificial Intelligence.

Clune, J. 2007. Heuristic evaluation functions for general game playing. In *Proceedings of the 22Nd National Conference on Artificial Intelligence - Volume 2*, AAAI'07, 1134–1139. AAAI Press.

Eladhari, M. P.; Sullivan, A.; Smith, G.; and McCoy, J. 2011. Ai-based game design: Enabling new playable experiences. Technical report, Technical Report, UCSC-SOE-11.

Genesereth, M.; Love, N.; and Pell, B. 2005. General game playing: Overview of the aaai competition. *AI magazine* 26(2):62.

Kansky, K.; Silver, T.; Mély, D. A.; Eldawy, M.; Lázaro-Gredilla, M.; Lou, X.; Dorfman, N.; Sidor, S.; Phoenix, S.; and George, D. 2017. Schema networks: Zero-shot transfer with a generative causal model of intuitive physics. *arXiv preprint arXiv:1706.04317*.

Ontanón, S.; Synnaeve, G.; Uriarte, A.; Richoux, F.; Churchill, D.; and Preuss, M. 2013. A survey of real-time strategy game ai research and competition in starcraft. *IEEE Transactions on Computational Intelligence and AI in games* 5(4):293–311.

Osborn, J. C.; Summerville, A.; and Mateas, M. 2017. Automatic mapping of nes games with mappy. In *Proceedings of the 2017 Workshop on Procedural Content Generation*.

Perez-Liebana, D.; Samothrakis, S.; Togelius, J.; Schaul, T.; Lucas, S. M.; Couëtoux, A.; Lee, J.; Lim, C.-U.; and Thompson, T. 2016a. The 2014 general video game playing competition. *IEEE Transactions on Computational Intelligence and AI in Games* 8(3):229–243.

Perez-Liebana, D.; Samothrakis, S.; Togelius, J.; Schaul, T.; and Lucas, S. M. 2016b. General video game ai: Competition, challenges and opportunities. In *Thirtieth AAAI Conference on Artificial Intelligence*.

Samuel, B.; Ryan, J.; Summerville, A.; Mateas, M.; and Wardrip-Fruin, N. 2016. Computatrum personae: toward a role-based taxonomy of (computationally assisted) performance. *Proceedings of EXAG*.

Smith, R. 2007. An overview of the tesseract ocr engine. In *Document Analysis and Recognition, 2007. ICDAR 2007. Ninth International Conference on*, volume 2, 629–633. IEEE.

Smith, A. M. 2013. Open problem: Reusable gameplay trace samplers. In *Ninth Artificial Intelligence and Interactive Digital Entertainment Conference*.

Summerville, A.; Osborn, J. C.; Holmgård, C.; Zhang, D.; and Mateas, M. 2017. Mechanics automatically recognized via interactive observation: Jumping. In *Proceedings of the 12th International Conference on the Foundations of Digital Games*.

Summerville, A.; Osborn, J. C.; and Mateas, M. 2017. Charda: Causal hybrid automata recovery via dynamic analysis. In *Proceedings of the International Joint Conference on Artificial Intelligence*.

Tremblay, J.; Dragert, C.; and Verbrugge, C. 2014. Target selection for ai companions in fps games. In *FDG*.

Weber, B. G.; Mateas, M.; and Jhala, A. 2010. Applying goal-driven autonomy to starcraft. In *AIIDE*.