

“Press Space To Fire”: Automatic Video Game Tutorial Generation

Michael Cerny Green, Ahmed Khalifa, Gabriella A. B. Barros, and Julian Togelius

Tandon School of Engineering, New York University, New York, USA
mcg520@nyu.edu, aak538@nyu.edu, gabbbarros@gmail.com, julian@togelius.com

Abstract

We propose the problem of tutorial generation for games, i.e. to generate tutorials which can teach players to play games, as an AI problem. This problem can be approached in several ways, including generating natural language descriptions of game rules, generating instructive game levels, and generating demonstrations of how to play a game using agents that play in a human-like manner. We further argue that the General Video Game AI framework provides a useful testbed for addressing this problem.

Introduction

Artificial intelligence techniques can be used in and with games in many different ways, to solve problems and create experiences, as well as to advance AI. Very coarsely, AI can be applied to generate content (Shaker, Togelius, and Nelson 2016), play games, and model players (Yannakakis and Togelius 2017), though there are many examples of usage of AI which do not fit these categories cleanly. Looking at AI-based game design patterns, one can see AI occasionally being used in somewhat more obscure roles such as spectacle, trainee or co-creator (Treanor et al. 2015).

With this paper, we seek to introduce yet another interesting problem, and role, for AI in games. The problem is that of generating tutorials (or instructions) and the role is that of teacher. We can loosely define it as: *given a game, generate a way to teach players how to play it*.

Most video games feature some kind of tutorial or instructions to assist players in getting started, and creating such tutorials is a complex task that requires skill and time. In other words, it is a great candidate for total or partial-automation. Furthermore, if we can find good methods for generating tutorials, these methods can be used to help teach people to perform a large variety of other tasks.

This paper surveys the (scant) literature on game tutorials and makes a few basic distinctions between types of tutorials. We then discuss some possible approaches to tutorial generation—it turns out this problem has much in common with, and builds on advances in, human-like game playing as well as procedural content generation (Shaker, Togelius, and

Nelson 2016). Finally, we discuss what it would take to generate tutorials within the General Video Game AI (GVG-AI) framework.

Background

Tutorials are the first interactions players encounter in a game. They help players understand game rules and, ultimately, learn how to play with them. In the game industry, developers experimented with different tutorial formats (Therrien 2011). In the arcade era, when most games were meant to be picked up and played quickly, they either had very simple mechanics, or they contained mechanics that players could relate to: “Press right to move”, “Press up to jump”, and so on. As a result, these games usually lacked a formal tutorial. As their complexity increased and home consoles started to explode in popularity, formal tutorials became more common.

Some game developers tried using an active learning approach which was optimized for players that learn through experimentation and exploring carefully designed levels. Games like *Megaman X* (Capcom, 1993) follow this approach. Other developers relied on old-school techniques, teaching the player everything before they could play the game, such as in *Heart of Iron 3* (Paradox Interactive, 2009). While one cannot argue that one technique is always superior to another, different techniques suit different audiences and/or games (Andersen et al. 2012; Williams 2009; Ray 2010).

Tutorials have evolved significantly over time, from the simple directive of Pong (“Avoid missing the ball for high-score”) to the exquisitely detailed in-game database of Civilization (Therrien 2011). Suddaby describes multiple types of tutorials (Suddaby 2012), from none at all to *thematically relevant contextual lessons*, where the tutorial is ingrained within the game environment.

Tutorial types are related to the different learning capabilities of the users who play them. Sheri Graner Ray (2010) discusses different *knowledge acquisition styles* in addition to traditional learning styles: Explorative Acquisition and Modeling Acquisition. The first style incorporates a child-like curiosity and “learning by doing”, whereas the second is about knowing how to do something before doing it. We can define at least two distinct tutorial styles from this, one being exploratory during gameplay and the other being more

instructional before the game even begins.

Williams suggests that active learning tutorials, which stress player engagement and participation with the skills being learned, may be ineffective when the player never has an isolated place to practice a particularly complex skill (Williams 2009). In fact, Williams argues that some active learning tutorials actually ruin the entire game experience for the player because of this reason. According to Andersen et al., the effectiveness of tutorials on gameplay depends on how complex a game is to begin with (Andersen et al. 2012), and sometimes are not useful at all. Game mechanics that are simple enough to be discovered using experimental methods may not require a tutorial to explain them. From these two sources, we find our first two boundaries for tutorial generation: there exists mechanics that are too simple to be taught in a tutorial, and there are mechanics complex enough that they may need to be practiced in a well-designed environment to hone.

In general, a game developer would want to use the most suitable tutorial style for their game. For that purpose, they must understand different dimensions/factors that affect the tutorial design process and outcome. Andersen et al. (Andersen et al. 2012) measured how game complexity affects the perceived outcome of tutorials. In their study, they defined 4 dimensions of tutorial classification:

- **Tutorial Presence:** whether the game has a tutorial or not.
- **Context Sensitivity:** whether the tutorial is a part of story and game or separate and independent from them.
- **Freedom:** whether the player is free to experiment and explore or is forced to follow a set of commands.
- **Availability of Help:** whether the player can request for help or not.

The classification proposed by Andersen et al. is binary. However, it is useful to see tutorials situated on a continuum between these extremes, as this allows us to gain a more nuanced understanding of game tutorials. For example: Figure 2 shows the tutorial in *Braid* (Number None, Inc, 2008) for a time rewinding mechanic. The tutorial only appears based on a certain event, i.e. the player's death. Players will not know about the mechanic until their first death. Instead of having the tutorial available at anytime or showing how to use the mechanic at the beginning, the developer reveals it when it is first necessary.

Sampling this space and comparing it with current game tutorials, we can find patterns repeated in multiple games. We can highlight the following tutorial types, which are not the only tutorials present in the space, but appear to be the most common ones:

- **Teaching using instructions:** These tutorials explain how to play the game by providing the player with a group of instructions to follow, similar to what is seen in boardgames. For example: Strategy games, such as *Starcraft* (Blizzard, 1998), teach the player by taking them step by step towards understanding different aspects of the game.
- **Teaching using examples:** These tutorials explain how to play by showing the player an example of what will hap-

pen if they do a specific action. For example: *Megaman X* uses a Non Playable Character (NPC) to teach the player about the charging skill (Egoraptor 2011).

- **Teaching using a carefully designed experience:** These tutorials explain how to play the game by giving the player freedom to explore and experiment. For example: in *Super Mario Bros* (Nintendo, 1985), the world 1-1 is designed to introduce players to different game elements, such as goombas and mushrooms, in a way that the player can not miss (Credits 2014). One way of seeing that is that early obstacles are instances of *patterns*, which reoccur later in the game in more complex instantiations or combinations (Dahlskog and Togelius 2012).



Figure 1: Street Fighters arcade cabinet. The cabinet shows different combos that can be done.

A game can have more than a single tutorial type from the previous list. Arcade games used demos and instructions to both catch the attention of the player and help them learn it. The demos help to attract more players, while simultaneously teaching them how to play. On the other hand, showing an information screen before the game start, such as in *Pacman* (BANDAI NAMCO, 1980), or displaying instructions on the arcade cabin, frequently seen in fighting games, helps the player understand the game and become invested in it. Figure 1 shows *Street Fighters* arcade cabinet where different characters combos and moves are written on it. *Megaman X* uses a carefully designed level to teach the player what to do, but still gives an example of the powershot attack if the player missed it.

Previous work has been done related to tutorial generation, especially in the area of aiding beginners, such as in Blackjack heuristics (de Mesentier Silva et al. 2016), by evolving the heuristics to be effective and concise. Similarly, TutorialPlan (Li, Zhang, and Fitzmaurice 2013) generates text and image instructions for users to learn AutoCAD. Work has also been done in automatic tutorial generation for API coding libraries (H 2014), which claimed that the resulting generated tutorials helped users learn libraries more effectively than current auto-generated tutorials. Alexander et al. postulated that open world mechanics could be transformed into quests (Alexander and Martens 2017). By for-



Figure 2: Braid teaching time rewinding mechanic when the player dies.

malizing the game logic of Minecraft into rules, they were able to create action graphs representing the player experience, and create quests and achievements based off those actions.

Game-O-Matic (Treanor et al. 2012) is a system which generates arcade style games and instructions for them by using a story-based concept-map inputted by a user. After the game is created, Game-O-Matic generates a tutorial page, explaining who the player will control, how to control them, and winning/losing conditions, by using the concept-map and relationships between objects within it.

How could tutorials be generated?

There already exist numerous artificial agents for a variety of video games. We theorize that if an agent can beat a video game, it could also build a tutorial using parts of the methodology that it used to win. An agent could do so by constructing a game-mechanic graph, similar to the mission-graph as described by Dormans (Dormans 2010).

With the knowledge of what it takes to beat a level (or to accomplish a specific goal), the agent could construct the graph so that leaf nodes contain terminal states, such as finishing a side quest or beating the level. Nodes in between the initial state and a terminal one would contain mechanics that lead to a terminal state such as player death or winning the game.

Figure 3 displays a tutorial graph created for *Space Invaders* (Taito, 1978). The green nodes are actions that equate to a critical victory path. Red nodes are a critical loss, and doing these actions would eventually result in losing the game. All nodes could be transformed into steps in a tutorial. The steps will explain how to achieve victory, avoid loss, or other subgoals such as how to score points. It is important to mention that nodes not on a critical path of either type can optionally be present in a tutorial and are not necessary.

The graph is flexible enough to be incorporated into different types of tutorial methods, for example those described in the previous section using *instruction*, *examples*, and a *carefully designed experience* for *Space Invaders*:

- **Tutorial using instructions:** Using the text in the node as

a start, a grammar could piece together sentences explaining each critical node on the winning and losing paths. The user would see text with phrases like “Press the left and right arrow keys to move left and right.”

- **Tutorial using examples:** Since the graph was built by an agent, we can assume the agent learned these mechanics and can replicate them. The tutorial generator could build an example by creating a level stage that would isolate the action or behavior contained within each node. The user would see a human-like artificial agent, such as one created by Khalifa et al. (Khalifa et al. 2016a), using the isolated movement mechanic, moving left and right on the screen. Using a human-like AI should help ensure that it would take similar actions to that of a human being, so that the observing player might learn more effectively.
- **Tutorial using a carefully designed experience:** This combines the *tutorial using examples* experience with levels that revolve around the mechanics to be learned. Rather than an artificial agent playing, the player would be in control. A level generator would be generating levels based around the critical path nodes. One such designed level would introduce player movement and shooting, and eventually move up to shooting aliens, finally teaching that killing all aliens will win the level, whilst still allowing the player to move around freely and unrestricted.

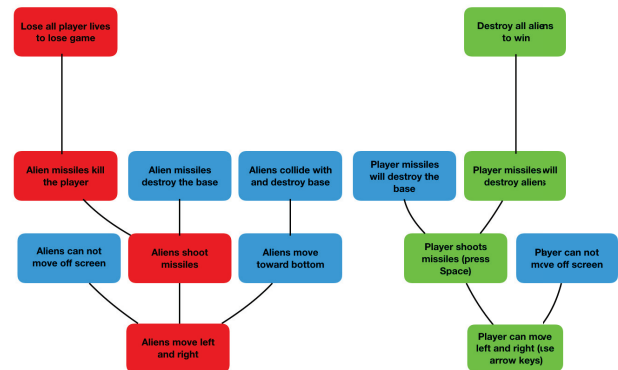


Figure 3: A *Space Invaders* tutorial graph. Green nodes are on a critical victory path. Red nodes are on a critical loss path

A tutorial generation system for teaching using instructions would require some sort of text generation engine, which would create text to be shown to the player to teach them how to play. The most obvious choice for this would be a grammar-based system, as it would allow the greatest amount of flexibility, especially over a simple text-replacement system. Grammars have been shown to be highly effective generators of content in the past (Rumelhart 1975; Pemberton 1989; Dormans 2011; Dormans and Bakkes 2011; Dart, De Rossi, and Togelius 2011; Togelius, Shaker, and Dormans 2016; Callaway and Lester 2002). One such example of a grammar-based text generative tool is Tracery (Compton, Filstrup, and others 2014), which was

made to generate stories with a pinch of the nonsensical. Stories generated by Tracery are not bound to be causal or even make much sense. Past research (Lang 1999) has shown that a system that generates sophisticated story-lines requires massive amounts of meta-data, which is often unwieldy, expensive in overhead, and arguably inflexible. Luckily, a tutorial generation system would not be required to tell a story, but rather teach the player how to play a game. Therefore, we believe that a grammar-based generator will suffice, even if it does not write causally.

Another way to generate text is using an artificial agent would explain its actions as it plays the game. Schrodt et al created an artificial agent to play Super Mario, which literally "thought-out-loud" as it played the game (Schrodt, Röhm, and Butz 2017). This technique of voicing intent or decision during play is known as *framing* (Charnley, Pease, and Colton 2012). The human-like AI mentioned before could be modified to explain its decision-making in real-time in order to teach a human-player how to play.

GVG-AI Tutorial Generation

The GVG-AI Framework provides a testbed for researchers to solve the problem of general video game-playing artificial intelligence, a competition where competitors can design AI agents that play a variety of unseen games efficiently. Multiple competition tracks are available to compete in, including Agent (Perez-Liebana et al. 2016), Level Generation (Khalifa et al. 2016b), Multiplayer Planning (Gaina, Pérez-Liebana, and Lucas 2016), Learning, and Rule Generation.

In this paper, we propose a method of tutorial generation and provide a possible beginning for a tutorial generation track for GVG-AI.

Because of the wide-ranging nature of game types in the GVG-AI framework, generated tutorials would have to be applicable to a variety of game styles and mechanics. As all games in GVG-AI are written in VGDL, tutorial generation can be done by simply reading the various interactions and terminations in the game's VGDL file and translating it into an easy-to-read, concise format. Before the first level of the game begins, a text display demonstrating button usage, enemy types, the player, and collectibles would be shown, as well as pointing out the main goal of the game. Tutorial generation can be divided into *mechanic discovery*, *graph creation*, and *text generation*, all of which will be described in the following subsections.

Mechanic Discovery

To generate a tutorial, our engine must first learn all relevant game mechanics. Using the *Sprite*, *Interaction*, and *Termination* sets contained within the game's VDGL file, the engine would record interactions between various sprites, movement controls, and terminal states. For example, Figure 4 shows the interaction and termination sets for Space Invaders. The *avatar* is defined in the SpriteSet to be a FlakAvatar, which means it can shoot *missiles*. In the Interaction-Set, missiles that collide with EOS (End of Screen) are destroyed.

An alternative to reading the interaction rules is using artificial agents to play the game. An AI agent would explore the game space, and discover game mechanics on its own.

Build Graph

The engine must have some way of understanding and relating game mechanics to one another. We propose a graph-based rule-set interpretation. The engine would create a graph based off the discovered mechanics, where each node contains a mechanic of some kind. Nodes that are connected to each other are related in some way. Leaf nodes can contain terminal states for the game, such as win or loss. If the terminal state is a winning state, this would become a potential candidate for the *critical path*, the shortest interaction chain necessary to win the level out of all interaction chains. This path would be the key goal the generator would recommend doing. Other paths that include losing states would correspond to things the generator would recommend *not* to do. Paths that have score changes, either in a positive or negative way, would correlate to actions the generator would recommend to either do often or avoid doing as much as possible. Multiple critical paths might result from this, which means the generator would have to find the most efficient critical path.

Alternatively, the engine could use a human-like AI (Khalifa et al. 2016a) armed with the knowledge of these discovered mechanics. This agent could play the game similarly to a human player to discover the critical path. Any game mechanics that it associates with loss would be placed on a critical loss path, and any game mechanics that it associates with winning would be placed on a critical win path.

Here is an example of how the engine would build an interaction chain. In GVG-AI the arrow keys are assumed to be the movement controls. Referencing Figure 4, the avatar (the player) is defined to be a "FlakAvatar" which is restricted to only horizontal movement. Thus the first green node would be created with references to "move", "player", and "left-right" movement using "arrow keys". Now within the interaction set, the system would first look for associations to "avatar" to add to the chain. It would see the first interaction "avatar EOS > stepBack", which means the avatar can not move outside the bounds of the screen. A node would be created with associations to "player", "not move" and "EOS" and linked to the previous node, as they are related via "avatar" and "movement".

Generate Text with Grammar

A tutorial is not complete without some vehicle through which to educate the player about the game. Using the graph of game mechanics, the engine would generate a text-based tutorial to display instructions to the player using a grammar for building blocks. Grammar 1 displays an example grammar that could be used for a GVG-AI Tutorial Generator. Using the grammar, the system would designate which of the sentence types are applicable for a given node. For example. The bottom-right green node ("Player can move left and right (using arrow keys)") references a behavior about movement (an Action Verb), the player (a Sprite), and infers arrow key controls (Control Verb and Button).

$\langle \text{tutorial} \rangle ::= \langle \text{win} \rangle \quad \langle \text{lose} \rangle \quad \langle \text{negative} \rangle \quad \langle \text{positive} \rangle$
 $\quad \langle \text{mechanics} \rangle \langle \text{controls} \rangle$

$\langle \text{win} \rangle ::= \text{'To win'} \langle \text{actionVerb} \rangle \langle \text{helpingAdj} \rangle \langle \text{sprite} \rangle$

$\langle \text{lose} \rangle ::= \text{'To lose'} \langle \text{actionVerb} \rangle \langle \text{helpingAdj} \rangle \langle \text{sprite} \rangle \mid \epsilon$

$\langle \text{negative} \rangle ::= \text{'Avoid'} \langle \text{actionVerb} \rangle \langle \text{sprite} \rangle \langle \text{negative} \rangle \mid \epsilon$

$\langle \text{positive} \rangle ::= \langle \text{actionVerb} \rangle \langle \text{sprite} \rangle \langle \text{positive} \rangle \mid \epsilon$

$\langle \text{mechanics} \rangle ::= \langle \text{mech} \rangle \langle \text{mechanics} \rangle \mid \langle \text{mech} \rangle$

$\langle \text{mech} \rangle ::= \langle \text{sprite} \rangle \quad \langle \text{helpingVerb} \rangle \quad \langle \text{actionVerb} \rangle$
 $\quad \langle \text{helpingAdjective} \rangle \langle \text{sprite} \rangle$

$\langle \text{controls} \rangle ::= \langle \text{cont} \rangle \langle \text{controls} \rangle \mid \langle \text{cont} \rangle$

$\langle \text{cont} \rangle ::= \langle \text{controlVerb} \rangle \langle \text{button} \rangle \text{'to'} \langle \text{actionVerb} \rangle \langle \text{sprite} \rangle$

$\langle \text{helpingAdj} \rangle ::= \text{'all'} \mid \text{'every'} \mid \text{'one'} \mid \text{'some'} \mid \text{'none'} \mid \epsilon$

$\langle \text{helpingVerb} \rangle ::= \text{'can'} \mid \text{'can not'} \mid \text{'will'} \mid \text{'will not'} \mid \epsilon$

$\langle \text{actionVerb} \rangle ::= \text{'move'} \mid \text{'shoot'} \mid \text{'dodge'} \mid \text{'kill'} \mid \text{'destroy'}$
 $\quad \mid \text{'collide with'} \mid \text{'beat'} \mid \text{'lose'} \mid \text{'win'}$

$\langle \text{controlVerb} \rangle ::= \text{'press'} \mid \text{'hold'} \mid \text{'release'}$

$\langle \text{button} \rangle ::= \text{'arrow keys'} \mid \text{'left and right'} \mid \text{'space bar'}$

Grammar 1: An example grammar for GVG-AI games.

Thus the system would determine that a Control Sentence would be most applicable to this node, and form the sentence "Press arrow keys to move player". The second bottom-most red node ("Aliens shoot missiles") would be determined to be a Mechanic Sentence, as it references behaviors pertaining to Alien and Alien Missile (two Sprites) and "shoot" (an Action Verb). Thus the engine would build a sentence "Alien shoots alien missile". Of important note, $\langle \text{sprites} \rangle$ are not defined in the grammar, as they differ between games in GVG-AI. Space Invaders has sprites such as "Alien" and "Missile" whereas Solarfox contains "Blib" and "PowerBlib".

Conclusion

Ideas behind designing game tutorials have evolved over time. It is ironic that, while so much effort has been put into generating levels, textures and stories, little to no effort has been made into automatically generating what is the first interaction between players and the gameplay. This paper proposes the problem of automatically generating game tutorials through the lenses of an AI problem. It expands on the four dimensions defined by Andersen et al (Andersen et al. 2012) for classifying different types of tutorials, and highlights three mainstream tutorial types: Teaching using instructions, using examples and using a carefully designed

```

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

```

Figure 4: Part of the rules for Aliens in VGDL

experience. It is our belief that the GVG-AI framework can be a useful testbed for experimenting with tutorial generation. Finally, we propose a graph-based rule-set interpretation of an agent playthrough, presented to the player as a grammar-based text tutorial. As it is, that proposal is our first future step.

References

- Alexander, R., and Martens, C. 2017. Deriving quests from open world mechanics. *arXiv preprint arXiv:1705.00341*.
- Andersen, E.; O'Rourke, E.; Liu, Y.-E.; Snider, R.; Lowdermilk, J.; Truong, D.; Cooper, S.; and Popovic, Z. 2012. The impact of tutorials on games of varying complexity. In *Proceedings of the SIGCHI Conference on Human Factors in Computing Systems*, 59–68. ACM.
- Callaway, C. B., and Lester, J. C. 2002. Narrative prose generation. *Artificial Intelligence* 139(2):213–252.
- Charnley, J. W.; Pease, A.; and Colton, S. 2012. On the notion of framing in computational creativity. In *ICCC*, 77–81.
- Compton, K.; Filstrup, B.; et al. 2014. Tracery: Approachable story grammar authoring for casual users. In *Seventh Intelligent Narrative Technologies Workshop*.
- Credits, E. 2014. Design club - super mario bros: Level 1-1 - how super mario mastered level design. <https://www.youtube.com/watch?v=ZH2wGpEZVgE>.
- Dahlskog, S., and Togelius, J. 2012. Patterns and procedural content generation: revisiting mario in world 1 level 1. In *Proceedings of the First Workshop on Design Patterns in Games*, 1. ACM.
- Dart, I. M.; De Rossi, G.; and Togelius, J. 2011. Speedrock: procedural rocks through grammars and evolution. In *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*, 8. ACM.
- de Mesentier Silva, F.; Isaksen, A.; Togelius, J.; and Nealen, A. 2016. Generating heuristics for novice players. In *Computational Intelligence and Games (CIG), 2016 IEEE Conference on*, 1–8. IEEE.
- Dormans, J., and Bakkes, S. 2011. Generating missions and spaces for adaptable play experiences. *IEEE Transactions*

- on *Computational Intelligence and AI in Games* 3(3):216–228.
- Dormans, J. 2010. Adventures in level design: generating missions and spaces for action adventure games. In *Proceedings of the 2010 workshop on procedural content generation in games*, 1. ACM.
- Dormans, J. 2011. Level design as model transformation: a strategy for automated content generation. In *Proceedings of the 2nd International Workshop on Procedural Content Generation in Games*, 2. ACM.
- Egoraptor. 2011. Sequelitis - mega man classic vs. mega man x. <https://www.youtube.com/watch?v=8FpigqfcvIM>.
- Gaina, R. D.; Pérez-Liébaná, D.; and Lucas, S. M. 2016. General video game for 2 players: framework and competition. In *Computer Science and Electronic Engineering (CEECE), 2016 8th*, 186–191. IEEE.
- Khalifa, A.; Isaksen, A.; Togelius, J.; and Nealen, A. 2016a. Modifying mcts for human-like general video game playing. In *IJCAI*, 2514–2520.
- Khalifa, A.; Perez-Liebaná, D.; Lucas, S. M.; and Togelius, J. 2016b. General video game level generation. In *Proceedings of the 2016 on Genetic and Evolutionary Computation Conference*, 253–259. ACM.
- Lang, R. 1999. A declarative model for simple narratives. In *Proceedings of the AAAI fall symposium on narrative intelligence*, 134–141.
- Li, W.; Zhang, Y.; and Fitzmaurice, G. 2013. Tutorialplan: automated tutorial generation from cad drawings. In *Twenty-Third International Joint Conference on Artificial Intelligence*.
- Pemberton, L. 1989. A modular approach to story generation. In *Proceedings of the fourth conference on European chapter of the Association for Computational Linguistics*, 217–224. Association for Computational Linguistics.
- Perez-Liebaná, D.; Samothrakís, S.; Togelius, J.; Schaul, T.; Lucas, S. M.; Couëtoux, A.; Lee, J.; Lim, C.-U.; and Thompson, T. 2016. The 2014 general video game playing competition. *IEEE Transactions on Computational Intelligence and AI in Games* 8(3):229–243.
- Ray, S. G. 2010. Tutorials: learning to play. http://www.gamasutra.com/view/feature/134531/tutorials_learning_to_play.php?print=1.
- Rumelhart, D. E. 1975. Notes on a schema for stories. *Representation and understanding: Studies in cognitive science* 211(236):45.
- Schrodt, F.; Röhm, Y.; and Butz, M. V. 2017. An event-schematic, cooperative, cognitive architecture plays super mario. *Cognitive Robot Architectures* 10.
- Shaker, N.; Togelius, J.; and Nelson, M. J. 2016. *Procedural Content Generation in Games*. Springer.
- Suddaby, P. 2012. The many ways to show the player how it's done with in-game tutorials. <https://gamedevelopment.tutsplus.com/tutorials/the-many-ways-to-show-the-player-how-its-done-with-in-game-tutorials-gamedev-400>.
- Therrien, C. 2011. "to get help, please press x" the rise of the assistance paradigm in video game design. In *DiGRA Conference*.
- Togelius, J.; Shaker, N.; and Dormans, J. 2016. Grammars and l-systems with applications to vegetation and levels. In *Procedural Content Generation in Games*. Springer. 73–98.
- Treanor, M.; Blackford, B.; Mateas, M.; and Bogost, I. 2012. Game-o-matic: Generating videogames that represent ideas. In *PCG@ FDG*, 11–1.
- Treanor, M.; Zook, A.; Eladhari, M. P.; Togelius, J.; Smith, G.; Cook, M.; Thompson, T.; Magerko, B.; Levine, J.; and Smith, A. 2015. Ai-based game design patterns.
- Williams, G. C. 2009. the pedagogy of the game tutorial. <http://www.popmatters.com/post/111485-active-learning-the-pedagogy-of-the-game-tutorial/>.
- Yannakakis, G. N., and Togelius, J. 2017. *Artificial Intelligence and Games*. Springer. <http://gameaibook.org>.
- ℋ. 2014. *AUTOMATIC GENERATION OF TUTORIAL FROM UNIT TESTS*. Ph.D. Dissertation, University of Tokyo.