Pac-Man: Predator v. Prey Evolutionary Algorithms

Jacob Page
University of Tennessee
Knoxville, Tennessee
jpage21@vols.utk.edu

Matthew Walters
University of Tennessee
Knoxville, Tennessee
mwalte21@vols.utk.edu

***Abstract*—In this paper, we attempt to use the well-known game Pac-Man to demonstrate Predator v. Prey Evolution, showing the algorithm to improve both Pac-Man and the Ghosts. We hope to create a Pac-Man that will complete the initial level from the original Pac-Man game, which means he will have to visit every space in the level. While Pac-Man learns to complete the level, the Ghosts will learn his tactics in order to catch Pac-Man as fast as possible. A key question is how hyper-parameters in the evolutionary process will impact the run time and results of the program. Overall results from this experiment were satisfying, but there is a lot of work to be done in order to improve them further. The fitness scores in this project are designed to be minimized rather than maximized. Fitness scores for Pac-Man improve from nearly two hundred to around ninety-five, which is a major improvement. The Ghosts' fitness scores never show a general trend of improvement, ranging from around twenty, which is a good result; to 500, which is nearly the worst possible score.**

***Index Terms*—Evolutionary Algorithms, Predator/Prey Evolution, Machine Learning, Video Game Visualization**

I. Introduction

For this project, we looked into how we could implement a Predator v. Prey simulation into a game of Pac-Man. The game Pac-Man was chosen because of how well-known the game is, and how the premise of the game is built around the concept of Pac-Man(Prey) running away from four Ghosts(Predators) while Pac-Man has to try and collect all of the pellets scattered throughout the map.

The goal of this project was to use random genomes to model the movement behavior of Pac-Man and the Ghosts to create the best versions of each party. Ultimately, we wanted Pac-Man to cover the entire map before time ran out, or the Ghosts caught him. This goal was used to then implement a Predator v. Prey evolutionary algorithm to see whether or not this simple concept could be used to create better versions of both Pac-Man and/or the Ghosts.

This topic was chosen because we wanted to try implementing Predator-Prey Evolution within some sort of game that was well-known. We knew that with a well-known game, it would be easy for people to see how the game interacted with the biologically-inspired algorithm by showing them the gradual improvements made throughout the entire process.

II. Related Work

There have been quite a few attempts at trying to replicate similar results to our project, and a few of these projects will be referenced at the end of this paper. They attempted to create a more interactive game such as Pac-Man, by using biologically-inspired algorithms, such as Predator v. Prey, to create enemies that learn as you play. These enemies make the game more interactive by creating more of a challenge depending on the skill level of the player. This creates a one size fits all difficulty that could be implemented in many different types of games. Most other work in this field differs from our project since they never try to tackle both sides of the equation. Instead of working with a player, we work to develop a better version of Pac-Man himself, while trying to generate the best Ghosts possible in order to catch him.

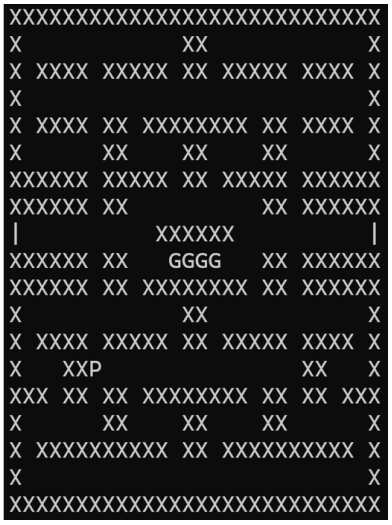

Fig. 1. The Pac-Man arena used in this simulation.

III. Approach

A. Simulating the Game

Our program to simulate the game was built from scratch in order for all the features we needed to be implemented properly. The arena (shown in Fig. 1) was built using a two-dimensional array that stores all characters in it. Within the array, an 'X' represents a wall or an object that neither Pac-Man nor the Ghosts can travel through, a '|' serves as an entrance to a portal that takes the user to the opposite side of the map, a '0' or ' ' stands for any open location that both Pac-Man and the Ghosts can travel to, a 'P' stands for the current location of Pac-Man, and a 'G' stand for the current location of any of the Ghosts. This basic implementation of

the map is then used to simulate the locations of everything on the map to determine where Pac-Man and the Ghosts can go and whether or not they have collided with each other.

B. Approach Methodology

We began with randomly generated genomes to tell the characters what moves they should make. The genomes are a set-length, the number of time steps that will be taken in any generation, and consist of a zero, one, two, or three, these numbers indicate the direction the character will move. Zero indicates a move upwards, one indicates a move to the left, two indicates a move down, and three indicates a move to the right. The better genomes would then be taken and used in the next generation with mutation and crossover in order to find the best genomes for both Pac-Man and the Ghosts. The evolution would occur in stages where each stage is defined as a certain number of generations of both Pac-Man and the Ghosts. In each stage, Pac-Man first learns on the best-scoring member from the previous Ghost generation, evolving for some number of generations (In our best cases, fifty generations). Then the Ghosts will evolve for the same number of generations, learning from the best-scoring Pac-Man is the newest generation that was made in the current stage.

IV. Implementation

A. Rules of the Game

As mentioned earlier, everything was built from scratch in order to have all the features needed to simulate the interactions between Pac-Man and the Ghosts. A run of the game consists of five-hundred steps where each step consists of every character making one move, or failing to move because they tried to move into a wall. In order to prevent Pac-Man and the Ghosts from moving right past each other, Pac-Man moves first, and cannot actually move into a space containing a Ghost, the Ghost can then approach him and Pac-Man loses. The game runs on the ASCII representation of the first Pac-Man level shown in Fig. 1, but the simulator can work on any map made in the same format as the map we use in this experiment.

B. Evolving Pac-Man and the Ghosts

Both Pac-Man and the Ghosts go through the same process of evolution. The process is rather simple but contains quite a few steps in order to try and produce better children from the parents. The first step is to initialize a population size of five hundred for both Pac-Man and each Ghost. The second step is to evaluate Pac-Man or the Ghosts with their respective fitness function in order to determine how well the current iteration is performing. The fitness scores for this project are different than most fitness scores as we are trying to minimize the fitness score rather than maximize them, so in our project, a perfect fitness score would be zero. We made this decision because it would be easier to evaluate whether Pac-Man or the Ghosts succeeded in their goal, if we saw a zero, it would be a perfect success, and we would not have to compare the score to an arbitrary high value that might change if we used a different map or modified the fitness scores.

Pac-Man's fitness is the number of unique spaces that Pac-Man did not visit during the simulation, plus the number of remaining steps that he could not take in the case that Pac-Man was caught by a Ghost. The Ghost's fitness score is how many steps it took for any of them to catch Pac-Man, plus the sum of each Ghost's minimum distance from Pac-Man during the run. This evaluation is made as one combined score; in essence, the four Ghosts in any game are a single "individual" in the population of Ghosts of any given generation. The third step is then using tournament selection to choose some of the best running cases from the population. Then there is a chance of mutation and crossover will occur on these selected individuals in order to produce the next generation of characters. The best average improvement we saw in hyper-parameter testing set the chance for mutation at 1%, and crossover also at 1%. If a mutation is to occur on a section of the genome, then one is added to that bit with rollover, so zero becomes one, one becomes two, two becomes three, and three becomes zero. Crossover works to create a child from two parents rather than just creating copies of the parents, so the two children generated are a random combination of the parents' genomes. Finally, the new generation is run through the simulation, and the entire process begins again.

C. Hyper-Parameter Tuning

Hours of hyper-parameter tuning were run in order to determine the best values for four different parameters. The four parameters were population size, probability of crossover, probability of mutation, and the number of generations that either side evolved per stage.

These tests only consisted of two stages for the sake of time, and we evaluated them by best improvement from the last generation of the first stage's best scores to the last generation of the second stage's best scores. Each combination got twenty tests, and their improvements were averaged, to find the best and most consistent improvement from each combination of parameters.

With more time, we may have also included different options for the tournament size of our selection process, but we left it at a default value of six. This means that regardless of the size of the population, for each member of the next generation, six members from the previous generation are selected and the member with the best fitness is copied to the next generation, before being mutated or crossed with one of their neighbors. This tournament size may have been too high for the smaller population sizes, causing them to lose genetic diversity over time.

For population size, the values tested were one hundred, two hundred, and five hundred. Testing showed that larger population sizes perform better, but they also dramatically increase the run time of the simulation. So, we had to make a decision on whether a higher run time was worth the better results we were seeing in the testing. Ultimately, we decided

on going with the population size of five hundred as we saw enough of an increase in order to justify the longer run times.

For the probability of crossover, the values tested were 1%, 5%, and 10%. In the end, tests that ran with a 1% crossover rate on average performed better than those with lower rates, so we went with this value for our final crossover rate.

For the probability of mutation, the values tested were the same as the values tested with crossover. Testing revealed that a lower mutation rate generally performed better, so we went with a mutation rate of 1%.

Finally, for the number of generations, the values tested were twenty-five, fifty, and one hundred. Testing revealed that the lower generation counts performed better than the highest counts, which confirmed what we thought from some of our graphs from earlier tests (Fig. 2). These tests show dramatic improvement within the first three to ten generations, then there is a plateau in improvement, where improvement is either small or the genomes get worse. The best average improvement actually had 50 generations, but in the top three were all three options, first 50, then 25, then 100. With this in mind we decided to go with 50 generations in most of our testing. We believe the smaller values perform better because there is more interaction between the two parties trying to learn, and less chance of over-fitting.

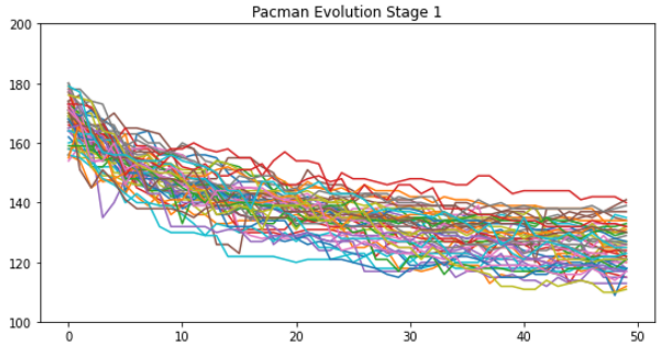

Fig. 2. The image above shows the first stage of Pac-Man's evolution before hyper-parameter tuning.

V. Results

A. General

Before hyper-parameter tuning, the default values we ran were one hundred for population size, 10% for the probability of crossover, 1% for the probability of mutation, and fifty generations. After hyper-parameter tuning, we ran the best combination (500 population size, 1% mutation and 1% crossover, with 50 generations) for 50 stages of evolution. In general, results were better after hyper-parameter tuning, but the best average improvement for Pac-Man was a fitness score 4.65 points better than the end of the first stage.

B. Creating the Ultimate Pac-Man

Before using hyper-parameter tuning, Pac-Man's initial fitness score started somewhere around 200. It would gradually improve on average somewhere between 120 and 140 with his best scores being about 100 as shown in Fig.2 and Fig. 3. These scores indicate that Pac-Man was covering anywhere from 1/3 to 1/2 of the 242 available spaces.

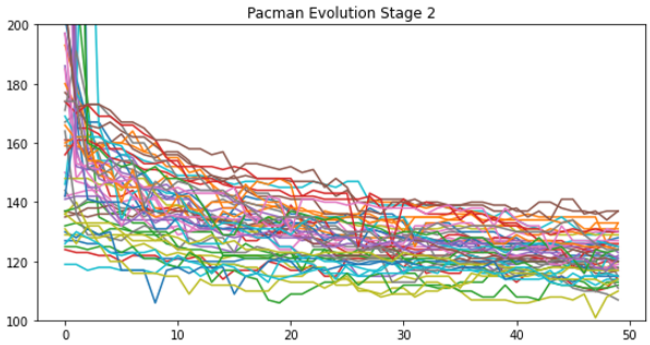

Fig. 3. The image above shows the second stage of Pac-Man's evolution before hyper-parameter tuning.

With optimized hyper-parameters, we ran our evolution simulation for 50 stages, hoping to see Pac-Man eventually cover the entire map, considering each stage could have improved his score by a few points. Unfortunately, the Ghosts were either too much for Pac-Man, or we were experiencing a cyclical evolution. Pac-Man's scores over time did not continuously improve, and even tended to get worse in the last 30 stages. However, there were at least 2 runs where the best score was actually less than 80, which means Pac-Man touched two-thirds of the map! Unfortunately, Pac-Man never actually won the game, in the sense that he would have scored a 0.

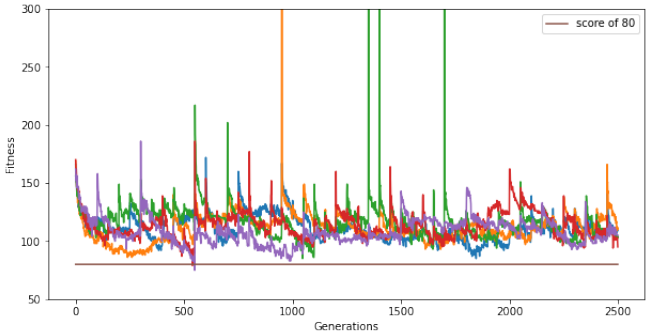

Fig. 4. The image above shows all of Pac-Man's best scores from every generation over all 50 stages of our tests.

In conclusion, we see a lot of improvement in Pac-Man that we did not expect initially, because prior to evolution, Pac-Man rarely touched even a quarter of the board, and because our methods for evolution are quite simplistic. Our Pac-Man evolved from a random string of numbers that performs poorly, to something, through evolution, that is actually quite close to completing the task that we have given him. This goes to show the power of something as simple as our program and how small amounts of evolution can affect real-world systems and programs.

C. Creating the Ultimate Ghosts

In the beginning, unlike Pac-Man who saw some decent improvement the Ghost's scores seemed more random and they were not progressing, shown in Fig. 5.

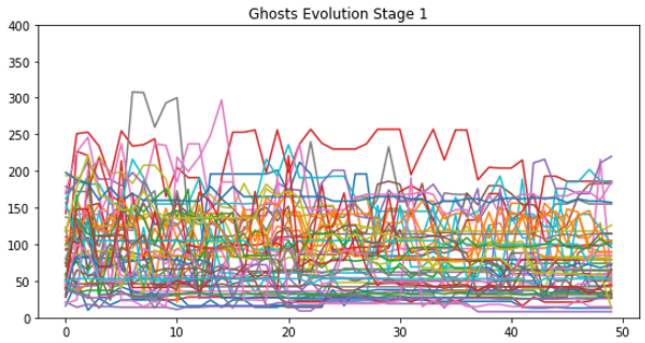

Fig. 5. The image above shows the first stage of the Ghosts evolution before hyper-parameter tuning.

Although no real improvement in the Ghosts occurs, in general, the Ghosts perform much better than Pac-Man does, which leads to many other questions about their fitness function. In many cases, the Ghost score is very low, often below 20, which is very good. This low score means the Ghosts are catching Pac-Man within the first 20 steps of the game.

The theory on why they are reaching much lower scores without actual improvement is that it is all attributed to the randomness of the experiment. Here in some cases, the pathing strings are pretty consistently leading the Ghosts into Pac-Man since Pac-Man can not actually see or react to the Ghosts. Part of this consistency could be attributed to the fact that there are four Ghosts and only one Pac-Man. The four Ghosts probably have pretty similar movement strings in this case, so the four Ghosts all move toward Pac-Man's starting location and end up catching him rather quickly.

Our theory on why the Ghosts generally see little improvement has to do with how their fitness score is determined. Unlike Pac-Man who has his own fitness score, the Ghosts' fitness score is calculated on them as a whole. With the Ghosts' fitness being calculated like this, only the collective score of the four Ghosts is being reflected, so the Ghosts that are performing very well could be brought down by Ghosts that aren't performing very well. Alternatively, one high-achieving Ghost on a team with three useless Ghosts can carry his team to the next generation again and again. This means that the genomes that are performing the best aren't properly being reflected in the mutation and reproduction phase. To combat this, one could perform crossover more heavily on the Ghosts, or score the Ghosts individually, and combine the best of each team in the next generation.

In conclusion, the Ghosts performed better than we expected and we have achieved the level of performance that we expected from them. Unfortunately, this good performance seems to be attributed to randomness, and that they simply outnumber Pac-Man, which makes their job a lot easier to accomplish than Pac-Man's job, leaving little room for improvement.

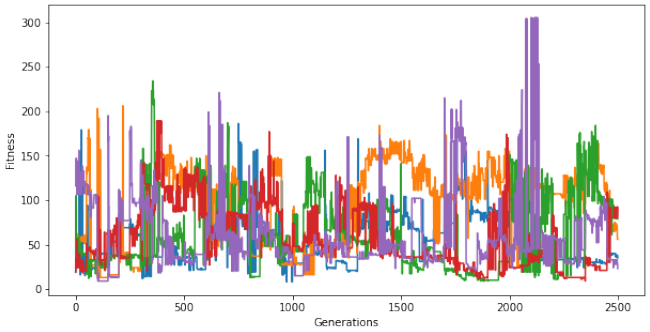

Fig. 6. Even with many stages of Evolution, the Ghosts cannot consistently improve.

VI. Conclusion

A. Discussion

Even though the process of developing these genomes is rather simple, there were a lot of small meticulous tasks that went into creating this simulator. The scores were never near perfect, and there is definitely a lot of work that could go into refining and perfecting the simulation. There are still a lot of questions to be answered, but overall we achieved what we wanted from this experiment by showing a simple simulation of a Predator v. Prey Evolution in a gaming medium that everyone would be able to recognize.

Pac-Man was given the task to try and visit as many unique locations on the map as possible without running into four different Ghosts all within five hundred steps. Given that he was given a pretty complicated task and the randomness of the initial generation, he performed the task rather well. He has a long way to go before we can test him out in an actual game of Pac-Man, but a lot of future work would be trying to implement a sort of neural network where Pac-Man will gain inputs and have to make decisions based on those inputs. Since he does not use any sort of neural network or even inputs, he works about as well as we think we can get him.

Even though on average the Ghosts perform better than Pac-Man, the results seem to be kind of disappointing. The Ghosts' behavior throughout the experiment does not change very much as there is no real trend of improvement, but instead, just some random genomes seem to perform better than others. Although the results were not what was expected, it just shows where much of the future work that we would put into this project would go. The Ghosts have a lot of potential for a fun game, and by themselves could be implemented within a game where a person controls Pac-Man, and the Ghosts learn to play against the player. This would create a fun interactive game that would increase in difficulty with the level of a skill that the player has.

One parallel we would like to draw with our simulation compared to the main game is kind of how similar our experiment turned out to the actual game. Those parallels are how the ghost in the actual game seem to randomly make their way around the map, while Pac-Man has to go through many

different tests to try and learn how to outsmart the Ghosts. The player more than likely will not beat the first level on the first try, so the player either has to go through iterations of trying to learn to best the Ghosts or they have to get lucky and randomly complete the course in the right way. Of course, the Ghosts differ in the game in that they can see where the player is, but that raises the question of whether or not the Ghosts would behave exactly the same should we tell them where Pac-Man is at any given time.

Overall, the results of the experiments were about as good as we could have expected although not as good as we had hoped. We produced a simulation that takes randomly generated genomes and evolves them so that they perform the tasks they have been given. Although Pac-Man's final genomes do not fully complete the complex task they have been given, they are able to complete a sizeable amount of the task considering how simple they are. This shows the power of trying to replicate biological processes within computer science and the power of algorithms that are based in biology.

B. Future Work

Given more time, the biggest change we could make would be modifying our genome representation. Implementing vision inputs and neural networks for both Pac-Man and the Ghosts, so that Pac-Man can make active decisions to stay away from the Ghosts while the Ghosts could work together to try and catch Pac-Man, would be an interesting next step in creating the Ultimate Evolutionary Pac-Man game. The question would be whether we give them a global or local vision. Global vision would involve everyone knowing where all the other characters in the arena were at any given time. Local vision could be two different concepts. One concept is that they can see so many squares around them (i.e. they could see anything within a 3-5 square radius). The second concept for local vision would be that they can see anything in direct line of sight as illustrated by the red line in Fig. 7.

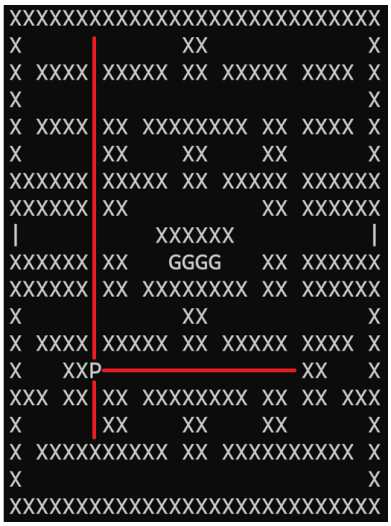

Fig. 7. Illustration of how direct line of sight could work if we were to implement it for Pac-Man.

As briefly mentioned earlier and to go along with adding vision to the characters would be implementing some sort of neural network. This neural network could have more knowledge about itself and have input such as vision that it would use to make decisions, since our random string genomes have no knowledge of the map whatsoever. A neural network would probably produce the best-performing Pac-Man since there would be decisions made based on current factors rather than simply following a set path to see how well he does.

Some further improvements that could be made to Pac-Man, that are not implementing a neural network, would be adding weights to spaces. This would involve changing up how our simulation works and allowing Pac-Man the ability to modify his genome in the middle of a run. Spaces that Pac-Man hasn't visited yet would be weighted higher than places he has previously visited, so he would be encouraged to mutate to move towards locations that he hasn't been to yet. This would more than likely increase the number of spaces that he would visit in the run, but this also could cause him to get caught quicker if vision were also not implemented. This probably seems like a departure from our evolutionary environment, but we could possibly evolve how these weights are determined and how likely Pac-Man might be to choose less weighted option.

The Ghosts are probably the largest area of improvement that could be made as they currently don't do anything special. One of the biggest changes that could be made would be separating out their fitness scores, so the Ghosts that perform better are not overshadowed by Ghosts that perform poorly. We could also look into implementing more defined rules, like adding in weighted movements similar to Pac-Man and encouraging the Ghosts to spread out and visit more unique spaces. Along with separating the fitness scores of the Ghosts, we could also implement some sort of communication between the Ghosts. For example, if one Ghost had local vision and could currently see Pac-Man, then all the Ghosts would know where Pac-Man was as long as that one Ghost could directly see him.

Given all the effort that had been put into this experiment a few new research questions have come to mind that could create more interesting results. The first thing that could be worked on would be implementing the different types of vision previously talked about and then comparing how they each affect the fitness scores. Then we could look to use whatever type of vision works best if any of them, and implement them in a more complete version of the project. Another major research question would be how our results would have differed if there had been only one Ghost and one Pac-Man. Would having one Ghost cause their learning behavior to change and would they have more of an incentive to learn? Then after training the Ghosts individually what would happen if we added the four trained Ghosts onto one map with Pac-Man? Even further we could try implementing a more even playing field and have some tests where there is a bigger arena and four Pac-Men and four Ghosts, almost making our own game.

Finally, something that we wanted to do that we were unable to do because of time constraints was to implement different maps of various sizes. Would Pac-Man perform better on a smaller map or would a larger map benefit him more since it would be more difficult for the Ghosts to find him? Could a Pac-Man trained on one map succeed on another map from the original game? Another idea that we have found interesting is what would happen if all the walls were removed and it was just an open arena. Would the Ghost be able to find Pac-Man sooner or would the open arena be to their detriment?

REFERENCES

[1] Schembri, Keith, and Georgios N. Yannakakis. "Evolving opponents for interesting interactive computer games." International Conference on Computational Intelligence and Games, 2007, pp. 99-106.

[2] Schembri, Keith, et al. "A generic approach for obtaining higher entertainment in predator-prey computer games." International Conference on Advances in Computer Entertainment Technology, 2009, pp. 279-282.

[3] R. J. Schalkoff and J. H. Gislason, "Text feature extraction for legal documents," in Proceedings of the 37th Annual Hawaii International Conference on System Sciences, 2004, pp. 1-10.

[4] T. Umedachi, Y. Kimura, and M. Ishida, "Evolving opponents by learning their personalities in fighting games," in Adaptive and Natural Computing Algorithms, vol. 4431, Springer, Berlin, Heidelberg, 2007, pp. 367-376.

[5] Khaled, Rilla, et al. "Model-based evaluation: A methodology for game usability research." International Journal of Human-Computer Interaction, vol. 23, no. 1, 2007, pp. 143-164.

[6] Yannakakis, Georgios N. "Game AI revisited." PhD thesis, University of Edinburgh, 2005.

[7] Halim, Z., et al. "Cultural diversity and game design." International Conference on Entertainment Computing, 2010, pp. 259-270.

[8] Shaker, Noor, et al. "Toward automatic personalized content generation for platform games." International Conference on Intelligent Technologies for Interactive Entertainment, 2016, pp. 421-425.

[9] Togelius, Julian, et al. "Search-based procedural content generation: A taxonomy and survey." Computational Intelligence and AI in Games, IEEE Transactions on, vol. 3, no. 3, 2011, pp. 172-186.

[10] Liapis, Antonios, et al. "Towards mixed-initiative procedural level design." International Conference on Computational Creativity, 2011, pp. 104-111.