A search-based approach for procedurally generating player adapted enemies in real-time

Viktor Olsson

Computer Science
Bachelor
15 Credits
Spring 2019
Supervisor: Steve Dahlskog

A search-based approach for procedurally generating player adapted enemies in real-time

Viktor Olsson

Department of Computer Science and Media Technology
Faculty of Technology and Society
Malmö University
Nordenskiöldsgatan 1, 211 19 Malmö, Sweden

viktor.j.olsson@gmail.com

Abstract

An Evolutionary Algorithm was run in real-time for the procedural generation of enemies in a third-person, wave based *hack and slash* and *shoot 'em up* game. The algorithm evaluates enemies as individuals based on their effectiveness at battling the player character. Every generation is presented as a new wave of enemies whose properties have been adjusted according to the fitness of the last wave. By constantly making new enemies more adept at the task of the defeating the current player, I attempt to automatically and naturally raise the difficulty as the game progresses. The goal is also to improve player satisfaction as a result. By analyzing the response from players and observing the changes of the generated enemies, I determine whether or not this is an appropriate implementation of Evolutionary Algorithms. Results showed that the success of the algorithm varied substantially between tests, giving a number of both failed and successful tests. I go through some of the individual data and draw conclusions on what specific conditions makes the algorithm perform desirably.

1. Introduction

Procedural Content Generation (PCG) refers to generating content algorithmically as an alternative to manually designing it. It has become widely used in modern game development due to its effectiveness at cutting down game design cost [1]. Specifically, the field of search-based PCG uses evolutionary and other metaheuristic search algorithms to evaluate the quality or fitness of the content being generated [9]. This evaluation process usually takes part during the development process of the game until a point is reached where the generation or individual is deemed satisfactory. At this stage only the final iteration is implemented into the release version of the game, and the rest is discarded [7]. Even though Evolutionary Algorithms (EA) are based on the idea of adapting to become the optimal performer at a specific purpose, this purpose is usually static and non-changing [7]. This is unlike adaptation in the real world from which the algorithms received their inspiration.

Since games are inherently focused on the player, manually designed content is made with the players' response in mind and is changed according to player feedback. This response is collected during tests and content is adjusted accordingly, at which point all players receive the same adjusted version made with either the average player in mind or or a specific player-type that fits the games' target group [3]. In this classic game design approach to player feedback, the purpose you adjust your content to is static and doesn't change on a game to game basis. Given that the very nature of EA is to adapt to specific purposes [7], a natural step of this algorithmic area would be to let it adapt to the player itself as they are playing the game; giving a personally unique experience.

2. Previous Research

Reinforcement Learning refers to methods that solve problems where a sequence of actions is associated with positive or negative rewards, but not with a specific target value, i.e. a *correct* action. One method for this is through Evolutionary Algorithms such as Genetic Programming and Neuroevolution [10]. The introduction of EA-based Reinforcement learning to digital games is not a new concept and started with John Koza evolving programs for playing Pac-Man in 1992 [5]. Since 2005, EA, neuroevolution in particular, has been used for evolving agents playing multiple genres of games such as racing, first-person shooters, strategy games and classic arcade games [10]. All of these examples focus strictly on evolving AI behaviour and all of them play the game much faster than real-time, playing thousands of times in order to learn it well.

Minimal research can be found on using EA for generating content to match the player during gameplay. Hastings et al. [4] noted that in every single example the

content is evolved outside the game. At the time there existed no game which evolves novel content based on usage statistics as the game is played.

Content in PCG can refer to characters, levels, maps, game rules, textures, stories, items, quests, music, weapons, vehicles and more. Since non-player character behaviour, NPC AI, is explicitly excluded from this definition; the previously mentioned examples of [9, 10] as well as the work in [2, 8] cannot be strictly regarded as PCG [6].

Since the work of Hastings et al. in 2009 [4], some research have attempted to use EA during a game's runtime to adapt to the current player [2, 8]. It is however focused on adjusting the AI behaviour of the enemy game agents to perform actions differently based on their interactions with the player. The conclusion to the paper by José Font [2] deemed the results to be satisfactory but noted that no *physical* attributes like health, speed, endurance or strength was ever modified; the goal was achieved through pure changes to behaviour.

No research could be found on generating this kind of content using the described approach and the intent of this paper is to attempt to decrease this gap in research by testing how appropriate Evolutionary Algorithms are for this purpose.

Research questions:

- **RQ1: How well suited are Evolutionary Algorithms for increasing enemies' effectiveness against the player during runtime in a game?**

- **RQ2: How much can player satisfaction be increased by adapting enemies to their individual playstyle?**

3. Method

3.1 The Game: Adaptation

Research is carried out through testing on a third-person, wave based *hack and slash* and *shoot 'em up* game called *Adaptation*. Adaptation is developed in Unity for the purposes of this paper. The player creates his character by choosing from a variety of statistical modifiers and then the choice to use either a sword or magic wand and then either a blocking or dashing ability. He uses these capabilities to face of against waves of enemies, all made from the same assortment of options available to the player. The player can attempt to aim the attacks manually by steering the character's rotation themself or by using the targeting system which automatically keeps the player character rotated towards the currently selected target.

Figure 1: *A snapshot of Adaptation*

3.2 The Evolutionary Algorithm

Since the focus of this research is on the *physical* traits of the enemies and not the behaviour itself, all enemies in the game will be controlled by a simple Finite State Machine (FSM). Minimal differences in behaviour appear based on the genetic makeup of the enemy, such as a melee enemy wanting to reduce distance to the player while a ranged one wants to increase it.

Every individual in the algorithm is applied to a single enemy in-game and every generation is a new wave of enemies that have been bred from the previous one. Some terms will now be defined for this paper to ease the communication of game elements. The genotype of every individual will be represented as a string of chars where every singular char is referred to as a *trait*; thus every enemy has a number of *traits*. Every *trait* of the enemy can be defined as belonging to one of two categories:

- ***Attributes***
  These improve the statistical modifiers of the enemy: health, attack damage, attack speed, attack range and movement speed. Every enemy has 12 *attribute traits* in total. One *attribute trait* grants one increase to that statistical modifier.

- ***Features***
  These provide the enemy with the actions they can perform in the game. Every enemy has two *features*: one *offensive* (melee or ranged attack) and one *defensive* (block or dash). One *feature trait* grants one action that can be performed.

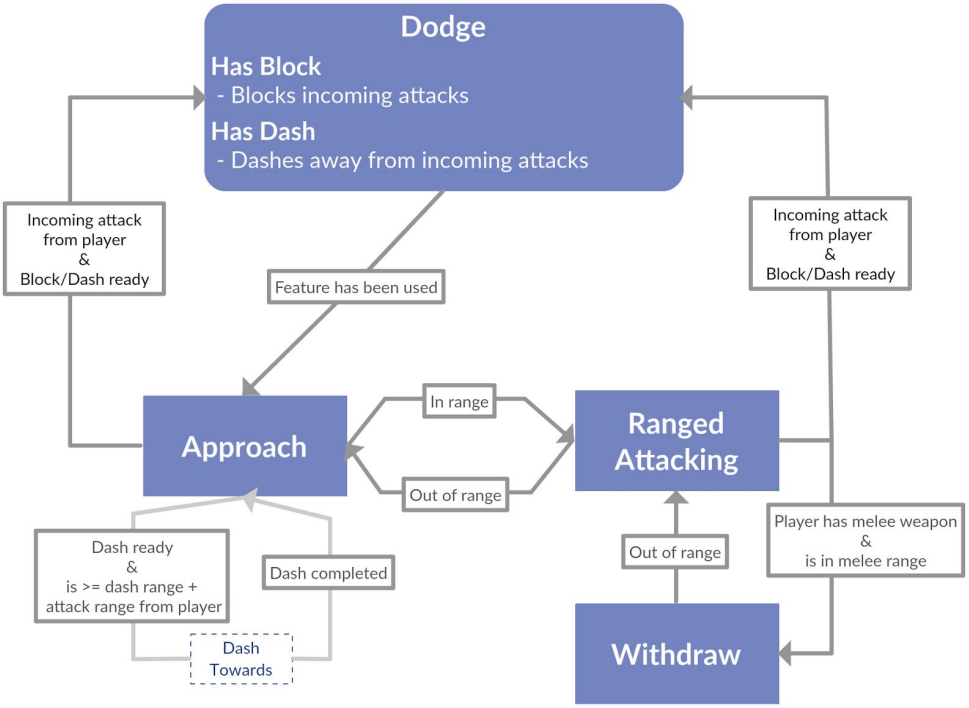


Figure 2a (above): *FSM that controls the behaviour of enemies with Ranged Attack.*

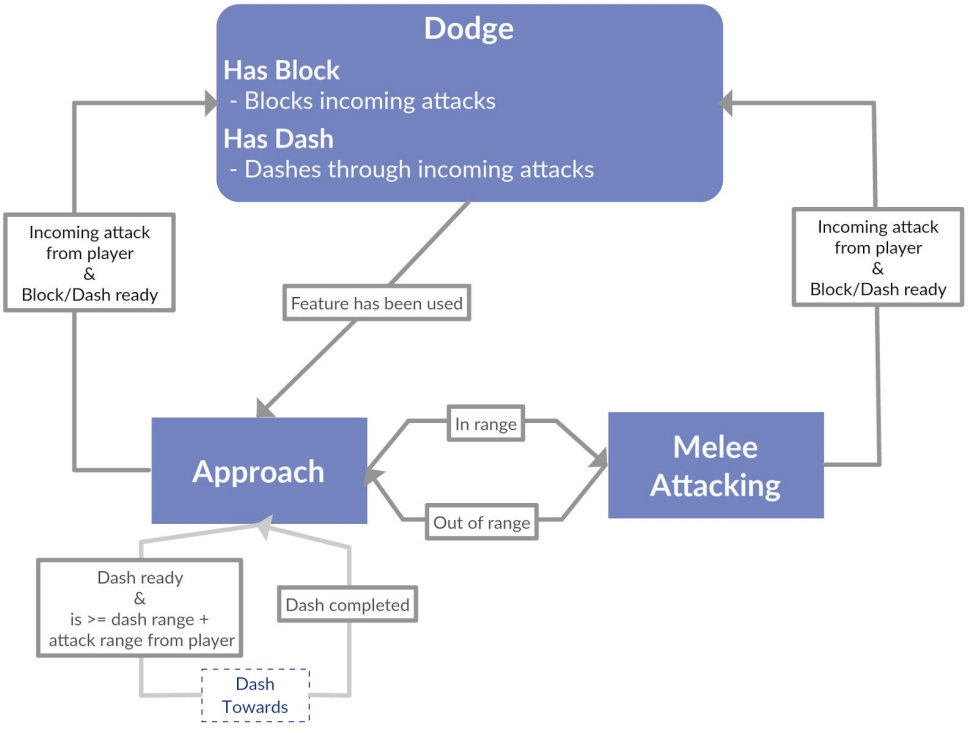


Figure 2b (to the right): *FSM that controls the behaviour of enemies with Melee Attack.*

Every enemy has the same amount of *traits*, their division between *attributes* or *features* is also the same for every enemy. The player character will be constructed as a combination of the same amount of *traits* as chosen by the player. This is to create the potential for variation in play-style so that the modular enemies get different purposes to adapt to. The only difference is the base *health* value for the player which is nine times that of a single enemy since the player faces nine enemies every wave before refilling its health.

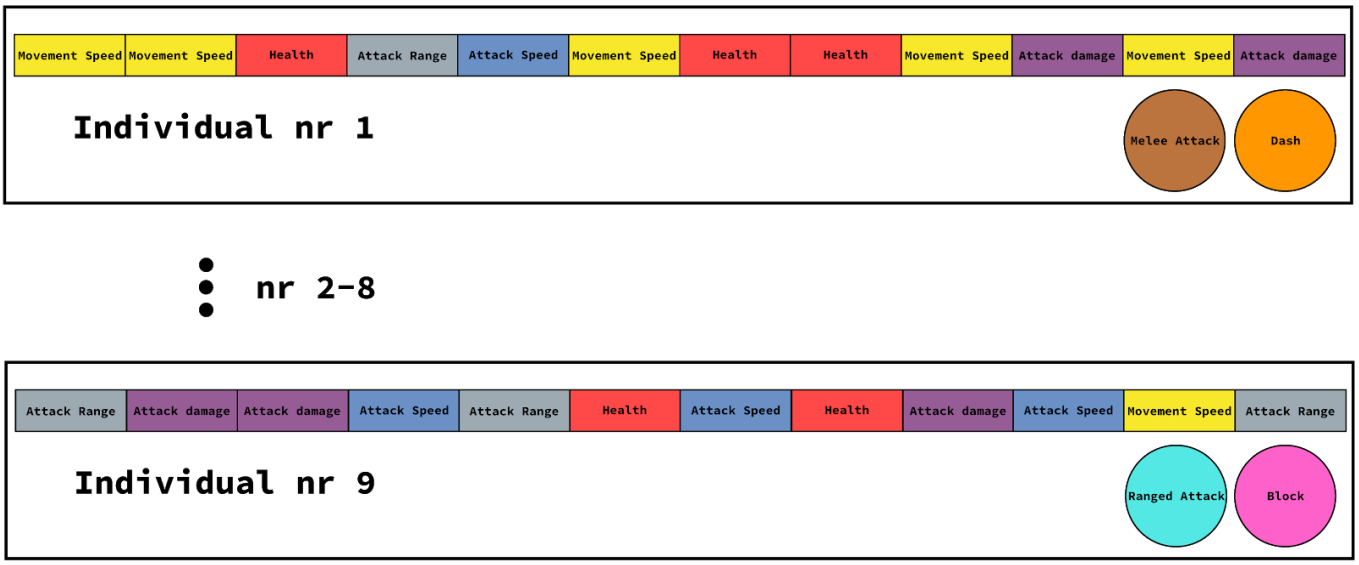


Figure 3: *How one generation is represented in Adaptation.*
Every white box is an individual containing its traits.
Rectangles are the attributes and circles are the features.

After a wave of enemies have been defeated by the player every individual is assigned a fitness score based on their performance against the player. The fitness score evaluates how effective the particular enemy was in their attempts at defeating the player character. Since not every enemy can actually defeat the player, especially if they are somewhat skilled at playing the game, the algorithm instead tries to assess how big of threat the enemy constitutes for the player. This threat level that makes up the fitness function is determined by the following factors:

- **Damage dealt:**
  How much damage did the enemy deal to the player? This is the highest prioritized factor since it is the only value that directly leads to the player character dying. Any time the player registers an attack making contact with their collider the damage dealt to them is added to the fitness score of the responsible enemy.

- **Almost damage dealt:**
  How much damage would the enemy have dealt the player had they not narrowly escaped the attempt? Since not every enemy can have a shot at damaging the player, with enough attempts that result in a close call it is still deemed to have constituted a threat to said player. This is measured by giving the player character an additional, larger collider outside the regular one. Any attack that makes contact with this outside collider but not the regular one does not cause damage to the player. The damage they would have taken from the attack, had it landed, is recorded into the fitness value of the responsible enemy. Since this did not actually cause any harm to the player it

is prioritized much lower in the fitness function than any real damage dealt. However enough of these close calls signifies that the enemy still provide a nuisance to the player and is therefore a threat.

- **Distance:**
  How close was it to the player on average? This information is relevant in two situations:
  - If the player is using ranged attacks and the enemy is using melee attacks. In this case the enemy can only inflict damage by reaching the player's position and as such pose a greater threat the closer they are to said player. Enemies in this situation receive a higher fitness function the lower their average distance to the player was.
  - If the player is using melee attacks and the enemy is using ranged attacks. In this case the opposite holds true i.e. enemies receive a higher fitness function the higher their average distance to the player was. This average distance value cannot be greater than the enemy's maximum attack range.

  All enemies are always spawned the same distance from the player to reduce discrepancies in the average distance that could be caused by static spawn points. These points being at wildly different distances from the moving player at any time would make the average distance of a spawned enemy irrelevant. Note that if the enemy and the player uses the same method of attack the distance provides no useful information. In that instance the value sent to the fitness function is simply the median value one would otherwise receive when being neither close nor far away.

- **Time alive:**
  How much time passed from the point that the enemy spawned until it died? This signifies how long the enemy managed to survive the player and adds a relevant survivability factor to the fitness function. An enemy that takes longer to kill does not necessarily pose a threat to the player if it never managed to inflict any damage. In this context an enemy that is quick to die but inflicts higher amounts of damage is deemed more of a threat to the player. To represent this, the time alive factor is much less prioritized in the fitness function in comparison to damage dealt. Enemies in the same wave will be spawned sequentially as opposed to every individual at once. This is to reduce the chance of an enemy surviving longer as a direct result of the player simply being busy with other enemies.

Normally an EA would run through several hundreds of generations before ending but since this algorithm will be starting and ending during the runtime of one game session the same luxury is not afforded [7]. In order to speed up the optimization of

the EA in this short amount of generations a high degree of elitism is used. Elitism involves copying a proportion of the fittest individuals, unchanged, into the next generation. This can sometimes have a dramatic impact on performance by ensuring that the EA does not waste time rediscovering previously discarded partial solutions [7]. For this reason the EA in *Adaptation* keeps half the individuals between every generation.

The Evolutionary Algorithm used for producing new generations is based in a Genetic Algorithm. After every enemy has received their fitness score, the worst half of the population is deleted and the best half is kept unchanged. This saved half is also crossbred with each others to replace the deleted individuals for the next generation. A uniform crossover is used for this to make sure there is no bias for two chars that are positioned close together in the string to be inherited together [7]. The uniform crossover will run twice for every new child, once for crossbreeding the *attributes* and once again for the *features.* Every newly bred child has a 5% chance for mutation in which a random trait is replaced by a new random one in the same category [7].

Much like in the work of José Font [2] which also tries to explore as many capabilities as possible in a small amount of generations; the genotype of the worst fitted enemy is replaced by a new randomly generated genotype in order to increase the exploration capability of the genetic program.

All *attributes* for the first generation are randomly generated as to not limit the initial search space. Since there are only four possible permutations of *features* an equal amount of every permutation is added to the first generation to decrease the risk that a possible solution is unexplored. One individual with random *features* is also added.

3.3 Evaluating the algorithm

Evaluation of the algorithm's effectiveness is carried out through gameplay sessions with volunteering playtesters. The playtesters are observed going through a number of enemy waves after which they answer a questionnaire regarding their impressions and opinions of the enemies. They go through two different sessions of gameplay in a row, one in which the enemies are generated by the discussed algorithm and one in which every wave is randomly generated. This random session serve as a comparison. If the algorithmically generated enemies cannot outperform the random ones the algorithm is deemed to not be worthwhile. The playtester themselves are given absolutely no information regarding the generation of the enemies, or the difference between the two sessions.

The questionnaire asks how players experienced the difficulty of enemies and which session they experienced to be the most *fun*, since player satisfaction is still a

motivating factor of the paper. Their answers are given in the form of a rating scale of the values *difficulty* and *fun* for the first, middle and last wave of enemies, spawned in during gameplay sessions one and two respectively.

The questionnaire is centered around how players experienced the difficulty and *fun* of the enemies. Their answers are given by selecting which was the most *difficult* and then the most *fun* out of the first, middle and last wave of enemies.They can also select the option of having experienced no difference. These question are asked for both gameplay sessions one and two respectively. When both gameplay sessions are completed the same questions and choices is given but for comparing between the two session instead of individual waves. They are finally given an optional question of describing any other perceived difference they might have experienced between the enemies, if any.

A secondary but less prioritized evaluation of the algorithm is also performed by recording the fitness score and generated *traits* of every enemy from the described playtesting sessions. It's measured if there is an overall increase in the enemies' fitness score from the first to the last wave during a session.This can provide valuable insight regarding if the enemies are actually getting better in their attempts at defeating the player as the game progresses. It is however not guaranteed to be reliable information since the players' skills with the game may improve during the sessions, thus making gameplay progressively easier and negating the enemies' improvements.The recorded *traits* provide the opportunity for observing the difference between waves, to determine if the GA achieved any substantial change in *traits* during the small amount of generations.

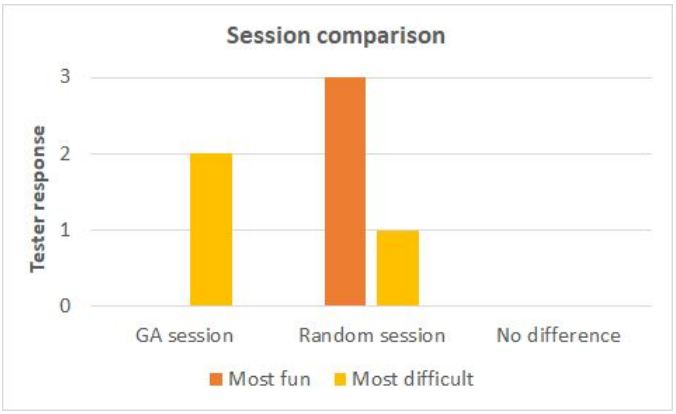
4. Results

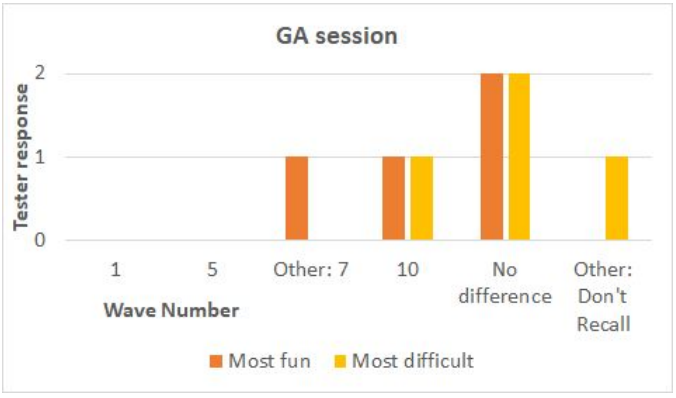
Testers were given an initial tutorial to learn the basic controls of the game. Afterwards they had to defeat a total of 10 waves in both game sessions, each consisting of 9 enemies. The order in which they received the two session was randomly decided for every tester. A total of 11 tests were performed before the results were compiled. To avoid any bias no testers were every used more than once. Every one of the 11 tests used a new player with no previous exposure to the game.

The first two tests resulted in none or minimal perceived difference. Testers expressed that there were heavy balancing issues in the game of *Adaptation* to the point were there was only one viable character build and playing style. This resulted in players being defeated by any enemy due to having a non-viable build. Alternatively players instantly defeated every enemy leading to no individuals having a measurable fitness. Any changes to the *individual* performed by the algorithm were not noticeable by the player due to the already extreme difficulty in either direction.

As a result the tests were deemed inconclusive and *Adaptation* went through a phase of balancing before testing continued.

After this an additional 9 tests were performed resulting in the following data:





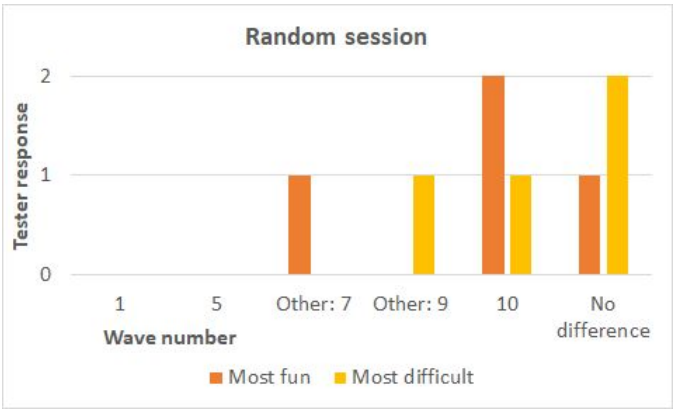
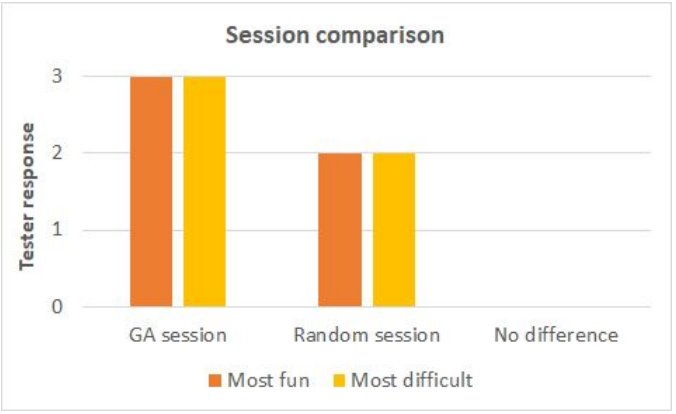
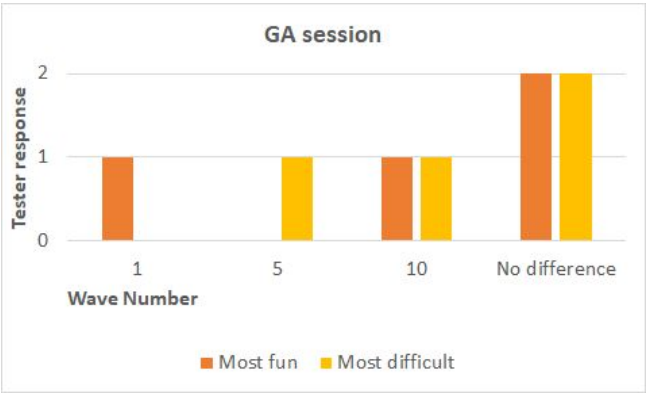


Figure 4a: *Summary of testers' responses to the questionnaire. In this left column are those subjected to GA-generated enemies in the first gameplay session, and random ones in the second session.*





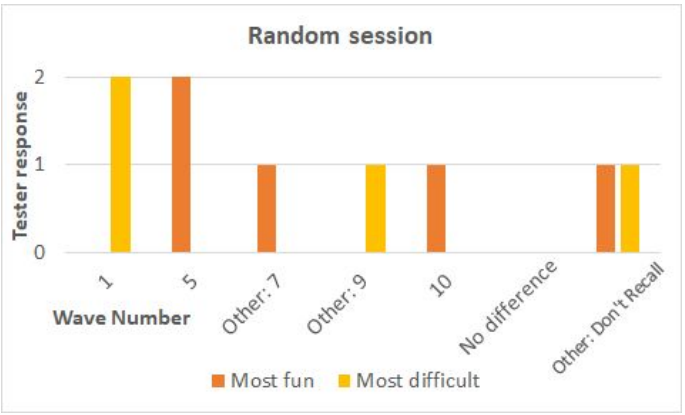


Figure 4b: *Summary of testers' responses to the questionnaire. In this right column are those subjected to randomly generated enemies in the first gameplay session, and GA ones in the second session.*

Figure 4a and 4b summarizes what the testers responded to the questionnaire. The x-axis shows what answers were available to choose from, while the y-axis shows how many testers that chose the specific answer. Testers could choose between *wave 1*, *5*, *10* or *No difference* when answering which specific wave was the most *fun* and *difficult*. If a tester wanted to provide an answer that wasn't part of this selection, they could write it in an additional option called *Other.* An answer in the diagrams marked with *Other* means that a tester provided their answer in this way instead of the pre-existing options.

The summarized data in *Figure 4a and 4b* is useful for determining how consistent the algorithm is in producing the desired results. Since the data shows that users experienced wildly differing results, it becomes important to look at the isolated data per session to determine what conditions made it perform desirably versus what made it fail. Thus changing the approach from a quantitative one to a qualitative one. A test was deemed successful if the following conditions were fulfilled:

- The tester chose waves from the latter half of the GA session (i.e. wave 5-10) as both the most *fun* and *difficult*.
- The tester chose the GA session as both the most *fun* and *difficult* when asked to compare the two sessions.
- **Optional:** The tester gave an answer that reflected the changes the algorithm is trying to implement as an answer to the final optional question. The exact question being "Was there anything noticeably different between the enemies of the two game sessions?".

Three out of the total nine tests were deemed successful and all of the conditions were fulfilled for those tests. The following test stood out as the most successful one due to the tester's extensive answer to the optional question:

Tester's questionnaire response:

First session (Random):

- **Most fun:** wave 10
- **Most difficult:** wave 10

Second session (GA):

- **Most fun:** wave 7
- **Most difficult:** wave 9

Session comparison:

- **Most fun:** second session (GA)
- **Most difficult:** second session (GA)

Answer to the optional question of noticeable difference:
Regarding the second session, the tester expressed that it felt like the game was adjusting its difficulty according to how he was playing. He felt like enemies who managed to damage him were more prevalent when the difficulty was too easy and the opposite when the difficulty was too hard. He also expressed that he felt like the behaviour of the enemies were implementing tactics that made them harder to defeat.
Chosen player *traits:*

***Attributes*:** Health x2, Damage x2, Range x4, Movement Speed x 4
***Features*:** Ranged Attack, Dash

From observing, it was noticeable that this tester had a very consistent playing style, behaving more similarly between the waves when compared to other testers. This stemmed from him learning the game mechanics quickly, having already adopted an optimal playing style during the initial waves of the first session. The result was him taking none or minimal damage during the entire first (randomly generated) session. As a stark contrast; the second session quickly developed a counter-strategy of *trait* combinations that made his strategy less viable. Perceived differences in enemy behaviour that the player experienced seems to have been a result of them performing the same actions but at higher speeds due to allocating a high share of Attack Speed and Movement Speed *traits*. While other tests returned mixed result this one stood out as a clear success.

Out of the other total testers about half verbally expressed that the game became more fun and less difficult during the second session as a direct result of the learning curve of the game. The algorithm did not produce a satisfactory result during any of those tests. On average, testers did not adopt a fully functional playing style until around the last wave of the first session or the initial wave of the second session. Testers' character builds also resulted in severely different learning curves. The steepest of these being the *feature* combination of Melee and Dash which some testers didn't perform fully functionally with until the last waves of the second session.

Chosen *traits* are another noticeable difference between the most successful test, in comparison to tests which failed in providing any perceivable difference. When selecting 4 *attributes* in both Range and Movement Speed they were each given an entire third of the total *attribute* allocation. Even though this is not radically different from the most common choice of 3 or 2 per *attribute* it is still an interesting observation. This observation is given more weight when comparing with the trait composition of a different test that was deemed successful.

During this other test the player was subjected to the GA session first and the random one second. Even though this tester responded that he perceived no difference in *fun* between the waves within either session, he answered that the GA session was both more fun and difficult than the random one. The tester noted that some waves in the GA session contained more ranged enemies with a higher attack speed making them harder to damage and avoid. He also expressed that the enemy behaviour felt more responsive in the first session while being slower and worse at defending itself in the second session. *Attributes chosen by* this player was also unusually uneven with 6 in Damage, 2 in Attack Speed and 4 in Movement Speed.

As suspected, no conclusive data could be found for the secondary evaluation by simply comparing the fitness values of the first to last generation within a session. The total, average and worst fitness varies wildly from generation to generation due to the very nature of trying to adapt to a non-static purpose. No player will act completely identically from one wave to another, preventing a smooth increase in fitness from ever occurring, even during successful sessions. When observing the changes in *traits* between generations, there is however a noticeable increase in the prevalence of those *traits* that seem to be effective against the player. At least in those tests deemed to be successful.

5. Conclusions and Future Work

A large amount of data was inconclusive due to the learning curve not leveling out for some testers until at least halfway through the full test. For future testing this can be amended by expanding the test with an additional section. Placed after the tutorial, but before the two game sessions. The new section would be dedicated solely to getting new players used to the game while no data is recorded. From previous tests we can conclude that this should be approximately one game session in size. It is hard to tell how many of the six tests that produced unsuccessful results were due to the discussed learning curve problem. From personal observations and the expressed opinion of testers it is estimated that most of them, possibly all, were substantially affected by it. For this reason; most of the following conclusions are based on those tests that were deemed successful.

My initial reasoning for giving individuals 12 *attributes* was so the GA had the opportunity of generating as diverse enemies as possible. The design rationale was that for the algorithm to properly adapt there has to be a large number of options for adapting. It instead led to changes between two enemies being too subtle to affect gameplay in a meaningful way, unless they have radically different compositions. Looking at the *traits* of both the player and enemies in tests where the algorithm performed desirably, there is a clear correlation with the amount of *min-maxing* done. When testers chose an uneven *trait* distribution, an optimal playing style for the

player became more apparent, making it easier for the algorithm to adapt. The opposite occurred when players chose a more balanced composition i.e. when they put a more equal amount of *attributes* in every category (health, damage etc.). To summarize; the more niche and unbalanced the *attribute* composition of the player character is, the more the same phenomenon occurs in the generated enemies. And only enemies with a less equalised composition, become perceivably different enough to affect how gameplay is experienced. This conclusion could lead to two different variations of future work depending on what purposes the developer is seeking.

If a developer's intent is to discourage players from using the same play style over and over again, a version of the algorithm could be implemented as it behaves now. A player constantly sticking to a similar play style would be gradually subjected to enemies against which they are ineffective. However, if a player varied their playstyle, the algorithm wouldn't have a consistent purpose to adapt to over several generations. That would essentially make the enemies randomly generated and less apt att defeating the player.

If the purpose instead is the same as the original intentions of this paper, the biggest challenge is keeping the point to adapt around similar in multiple generations. As discussed, the differences between two individuals are currently too subtle. One improvement would be to reduce the amount of *attributes* a single individual gets and instead make every single given *attribute* increase its corresponding modifier drastically more. Currently, everyone of the 12 given *attributes* provides a minor increase to its corresponding modifier. A suggestion is changing this to 3 *attributes* per individual and making the increase that every single *attribute* provides at least four times as big. This would force the chosen *attribute* composition of every player character to be more niche and unbalanced, leading to the same phenomenon occurring in generated enemies. Which in turn leads to the sought after result, as discussed in the second paragraph of this section. To still keep the opportunity for generating as diverse individuals as possible; a suggestion could be to increase the possible selection with additional types of *attributes*, affecting new modifiers on top of the existing ones (health, attack damage, attack speed, attack range and movement speed) .

A second improvement would be to enhance the behaviour of the enemies. Currently, even if an individual has a *trait* composition that makes its statistics better against the player, it will not adopt the optimal playing style for this composition. The only changes in behaviour occur with different *features,* but there are no such changes for different *attributes*. This results in lost potential when a great number of individuals never assumes the tactic it's best suited for. Manually expanding the current system with additional behaviours for specific *attribute* combinations is

possible. Due to the high amount of *attribute* permutations this means a high workload, in addition to balancing for making sure the behaviour is actually optimal. A suggestion would be to instead evolve the behaviour during run-time as well, similar to what was accomplished in the *Previous Research* section. This would hopefully lead to every individual adopting behaviour that is optimal for its unique genotype

I would also be curious to see what differences could be achieved by giving the algorithm a neuroevolutionary approach. The Genetic Algorithm in *Adaptation* currently includes no evolving neural topologies or any form of artificial neural network. As seen in the *Previous Research* section, neuroevolution has proven to be a versatile tool and implementing that approach to *Adaptation* while observing any differences in results could be interesting.

Since tests in which the algorithm performed successfully resulted in the session being rated as more fun it is suggested that adapting enemies leads to an increase in player satisfaction. This serves as an incentive for further improving this algorithmic approach, causing it to more consistently increase player satisfaction.

6. Author's Note

To encourage future research in this area:
Anyone seeking access to the source code for *Adaptation* can contact me using the email address listed at the beginning of this paper and I will provide it. Feel free to fully modify it for your purposes.

7. Sources

1. Amato, Alba. “Procedural Content Generation in the Game Industry.” Game Dynamics, 31 Mar. 2017, pp. 15–25., doi:10.1007/978-3-319-53088-8_2.

2. Font, José M. “Evolving Third-Person Shooter Enemies to Optimize Player Satisfaction in Real-Time.” Applications of Evolutionary Computation Lecture Notes in Computer Science, 2012, pp. 204–213., doi:10.1007/978-3-642-29178-4_21.

3. Fullerton, Tracy. Game Design Workshop: a Playcentric Approach to Creating Innovative Games. AK Peters / CRC Press, 2019.

4. Hastings, Erin J.; Guha, Ratan K.; Stanley, Kenneth O. “Evolving Content in the Galactic Arms Race Video Game.” 2009 IEEE Symposium on Computational Intelligence and Games, 7 Sept. 2009, doi:10.1109/cig.2009.5286468.

5. Koza, John R. Genetic programming: on the programming of computers by means of natural selection. MIT Press, 1992.

6. Noor Shaker, Julian Togelius, and Mark J. Nelson (2016). Procedural Content Generation in Games: A Textbook and an Overview of Current Research. Springer. ISBN 978-3-319-42714-0.

7. Schwab, Brian. AI Game Engine Programming. Course Technology Cengage Learning, 2009.

8. Stanley, K.O.; Bryant, B.D.; Miikkulainen, R. “Real-Time Neuroevolution in the NERO Video Game.” IEEE Transactions on Evolutionary Computation, vol. 9, no. 6, 5 Dec. 2005, pp. 653–668., doi:10.1109/tevc.2005.856210.

9. Togelius, Julian; N. Yannakakis, Georgios; O. Stanley, Kenneth; Browne, Cameron. “Search-Based Procedural Content Generation: A Taxonomy and Survey.” IEEE Transactions on Computational Intelligence and AI in Games, vol. 3, no. 3, 29 Apr. 2011, pp. 172–186., doi:10.1109/tciaig.2011.2148116.

10. Yannakakis, Georgios N., and Julian Togelius. Artificial Intelligence and Games. Springer, 2018.

8. Appendix: Test Questionnaire

Rate the next categories in comparison to the other waves within the same game session:

First (1st) Game Session

Which wave was the most fun?

- Wave 1
- Wave 5
- Wave 10
- No difference
- Other (Provide your own alternative)

Which wave was the most difficult and/or challenging?

- Wave 1
- Wave 5
- Wave 10
- No difference
- Other (Provide your own alternative)

Second (2nd) Game Session

Which wave was the most fun?

- Wave 1
- Wave 5
- Wave 10
- No difference
- Other (Provide your own alternative)

Which wave was the most difficult and/or challenging?

- Wave 1
- Wave 5
- Wave 10
- No difference
- Other (Provide your own alternative)

Rate the next categories in comparison between the two different game sessions

Comparison between both Game Session

Which game session was the most fun between the two?

- First (1st) Game Session
- Second (2nd) Game Session
- No difference

Which game session was the most difficult and/or challenging between the two?

- First (1st) Game Session
- Second (2nd) Game Session
- No difference

(Optional) Was there anything noticeably different between the enemies of the two game sessions?