Evolving Content in the Galactic Arms Race Video Game

In: Proceedings of the IEEE Symposium on Computational Intelligence and Games (CIG09). Piscataway, NJ:IEEE
Winner of the Best Paper award at CIG-2009

Erin J. Hastings, Ratan K. Guha, and Kenneth O. Stanley

***Abstract*— Video game content includes the levels, models, items, weapons, and other objects encountered and wielded by players during the game. In most modern video games, the set of content shipped with the game is static and unchanging, or at best, randomized within a narrow set of parameters. However, ideally, if game content could be constantly renewed, players would remain engaged longer in the evolving stream of novel content. To realize this ambition, this paper introduces the *content-generating NeuroEvolution of Augmenting Topologies* (cgNEAT) algorithm, which automatically evolves game content based on player preferences, as the game is played. To demonstrate this approach, the Galactic Arms Race (GAR) video game is also introduced. In GAR, players pilot space ships and fight enemies to acquire unique particle system weapons that are evolved by the game. As shown in this paper, players can discover a wide variety of content that is not only novel, but also based on and extended from previous content that they preferred in the past. The implication is that it is now possible to create games that generate their own content to satisfy players, potentially significantly reducing the cost of content creation and increasing the replay value of games.**

I. Introduction

Creating the models, levels, textures, and other content that players encounter and possess in games is time-consuming and expensive [1], [2]. In part to address this problem and to provide additional replay value, it is increasingly popular for developers to distribute tools that enable players to create their own content [3], [4] or to randomize content (e.g. random map generators). However, content creation tools usually require significant effort to master and specialized knowledge beyond that of most players. Moreover, randomization only works if it is tightly constrained to avoid generating undesirable content, and provides no means to deduce the kind of content that players prefer. Thus a more intriguing potential solution is to automatically generate content during the game, as it is played, based on actual player behavior.

To make such content generation possible, this paper introduces the *content-generating NeuroEvolution of Augmenting Topologies* (cgNEAT) algorithm, which automatically generates content for video games and simulations. This approach creates new content in real time through an evolutionary algorithm based on the content players liked in the past.

To show that automatic content generation is genuinely possible, cgNEAT is implemented in this paper in a novel video game called Galactic Arms Race (GAR). In GAR, *compositional pattern producing networks* (CPPNs), which are a variant of artificial neural network (ANNs), genetically encode and control particle system weapons. The CPPNs evolve and increase in complexity though cgNEAT, which tracks which weapons players fire the most. That way, during the game, weapon behavior becomes increasingly sophisticated while consistently evolving to suit player tastes. In this way, it is the *player* rather than the designer who ultimately implicitly determines what kind of content will populate the game.

The main result is that players discover a wide variety of content that is not only novel, but also based on and extended from previous content that they liked in the past, which makes this approach superior to simple content randomization. Moreover, for developers, it means that it is possible to produce games and simulations that create their own content to satisfy users, impacting both the production cost and longevity of future such games. While the evolved content in GAR is the weapons, in principle cgNEAT can evolve any class of content in the same way, opening up an exciting new direction in video game design.

II. Background

This section first reviews existing video games that rely in part on machine learning. Next, related research on automatic content generation for games is presented. Third, NEAT and CPPNs, which are major components of cgNEAT, are detailed. Finally, because particle system weapons are automatically evolved by cgNEAT in GAR, prior work in evolving particle systems is discussed.

A. Machine Learning in Commercial Games

The impact of machine learning so far on the video game industry has been limited, although some games are beginning to incorporate learning techniques. However, content generation continues to be absent from applications of machine learning in commercial games. The most common application of machine learning is to optimize the policy that controls non-player characters (NPCs) (figure 1). For example, the ANN race car controllers Colin McRae Rally 2.0[1] and Forza Motorsport 2[2] and the creature brains in Creatures 3[3] and Black and White 2[4] are learned. Generally,

Erin J. Hastings, Ratan K. Guha, and Kenneth O. Stanley are with the School of Electrical Engineering and Computer Science. University of Central Florida, Orlando, FL 32816. email: {hastings, guha, kstanley}@eecs.ucf.edu.

[1]Copyright 2001 Codemasters, http://www.codemasters.com/
[2]Copyright 2007 Microsoft Game Studios, http://forzamotorsport.net/
[3]Copyright 2004 Creature Labs, http://www.gamewaredevelopment.co.uk/
[4]Copyright 2005 Lionhead Studios, http://www.lionhead.com/

(a) (b) (c) (d) (e)

Fig. 1. **Evolving NPC Behavior in Existing Games**. Learned policies enable race car controllers to navigate tracks with complex physics in (a) Colin McRae Rally 2.0 and (b) Forza Motorsport 2. Learned policies also control decision making for a variety of characters in (c) Creatures 3 and (d) Black and White 2. In (e) NERO [5], players evolve a squad of virtual soldiers to fight other players. NERO introduced rtNEAT, which demonstrated the viability of NPC evolution in real time. Building on the success of these games, cgNEAT aims to evolve other forms of game content, outside of NPC behavior.

the NPC behavior in such games is trained by developers before release. Recently, although it is not a commercial game, NeuroEvolving Robotic Operatives (NERO [5]; http://nerogame.org/) enabled players to evolve the tactics for a squad of virtual soldiers in real-time, while the game is played, demonstrating the potential viability of evolution to commercial gaming.

The success of learning algorithms in these games suggests the potential to apply learning to create content beyond NPC behavior, as discussed in the next section. In fact, automatically generating content could further open the video game industry to the possibilities created by machine learning.

B. Evolving Game Content

Evolving game content is an emerging research area with great potential to contribute to the mainstream gaming industry. Two of the few current examples of evolved game content include race tracks [6] and even the rules of the game itself [7]. These investigations thus represent the cutting edge of an exciting new research direction. However, in these examples content is evolved *outside* the game; there currently exists no game (and thus no comparable algorithm to cgNEAT) that evolves novel content based on usage statistics as the game is played. The aim of cgNEAT is thus to evolve such content in real time, based on tracked player preferences, and seamlessly introduce the newly generated content into the game. The cgNEAT method represents content with CPPNs evolved by NEAT, both of which are discussed next.

C. NeuroEvolution of Augmenting Topologies (NEAT)

The NEAT method was originally developed to solve control and sequential decision tasks. The ANNs evolved with NEAT control agents that select actions based on their sensory inputs. While previous methods that evolved ANNs (i.e. neuroevolution methods) evolved either fixed topology networks [8], [9], or arbitrary random-topology networks [10], [11], [12], NEAT begins evolution with a population of small, simple networks and *complexifies* the network topology into diverse species over generations, leading to increasingly sophisticated behavior. A similar process of gradually adding new genes has been confirmed in natural evolution [13], [14] and shown to improve adaptation in a few prior evolutionary [15] and neuroevolutionary [16] approaches. This section briefly reviews the NEAT method; Stanley and Miikkulainen [5], [17] provide complete introductions.

To keep track of which gene is which while new genes are added, a *historical marking* is uniquely assigned to each new structural component. During crossover, genes with the same historical markings are aligned, producing meaningful offspring efficiently. Traditionally, speciation in NEAT protects new structural innovations by reducing competition between differing structures and network complexities. However, in this work, because a human performs selection rather than an automated process, the usual speciation procedure in NEAT is unnecessary.

Most importantly, complexification, which resembles how genes are added over the course of natural evolution [13], allows NEAT to establish high-level features early in evolution and then later elaborate on them. For evolving content, complexification means that content can become more elaborate and intricate over generations.

In this paper, particle system weapons are controlled by ANNs evolved by NEAT. NEAT is chosen because (1) it is proven effective for evolving ANNs in a diversity of domains [17], [18], [19], [20], and (2) it is fast enough to run in real time (in the NERO video game [5]), which is required for an interactive system. Because NEAT is a strong method for evolving controllers for dynamic physical systems, it can naturally be extended to evolve the motion of particle effects as well, such as those featured in GAR.

The next section explains CPPNs, which are the special variant of ANNs evolved by NEAT in GAR.

D. Compositional Pattern Producing Networks (CPPNs)

Compositional pattern-producing networks (CPPNs) are a variation of artificial neural networks (ANNs) that differ in their set of activation functions and how they are applied [21], [22]. While CPPNs are similar to ANNs, the different terminology originated because CPPNs were introduced as pattern-generators rather than as controllers. This section explains the difference in implementation and application between CPPNs and ANNs.

Whereas ANNs often contain only sigmoid or Gaussian activation functions, CPPNs can include both such functions and many others. The choice of CPPN functions can be biased toward specific patterns or regularities. For example, periodic functions such as sine produce segmented patterns

with repetitions, while symmetric functions such as Gaussian produce symmetric patterns. Linear functions can be employed to produce patterns with straight lines. In this way, CPPN-based systems can be biased toward desired types of patterns by carefully selecting the set of available activation functions.

Additionally, unlike typical ANNs, CPPNs are usually applied across a broad space of possible inputs so that they can represent a complete image or pattern. Because they are compositions of functions, CPPNs in effect encode patterns at infinite resolution and can be sampled at whatever resolution is desired.

Successful CPPN-based applications such as Picbreeder [19], in which users from around the Internet collaborate to evolve pictures, and NEAT Drummer [23], which evolves drum track patterns to accompany songs, demonstrate that CPPNs can evolve diverse content. The approach in this paper evolves particle systems encoded by CPPNs, as discussed next.

E. Evolving Particle Systems

Particle systems are ubiquitous in computer graphics for producing non-solid, or "fuzzy," phenomena such as fire, smoke, water, cloth, explosions, magic, electricity, and many others [24], [25]. Particle systems are usually defined by (1) a set of points in space and (2) a set of rules guiding their behavior and appearance, e.g. velocity, color, size, shape, transparency, rotation, etc. Such rule sets can be complex and are usually hand-coded. Therefore, evolving particle system behavior is a suitable domain for investigating content generation technology.

The approach in this paper is built upon NEAT Particles, a general-purpose particle effect evolver, and NEAT Projectiles, which is specialized for evolving particle weapon effects intended for video games [26]. Both systems evolve CPPNs with NEAT to control the motion and appearance of particles. GAR embeds this technique into the game through cgNEAT, an automatic content generation algorithm, introduced in the next section.

III. Content-Generating NEAT (cgNEAT)

The aim of the cgNEAT algorithm is to automatically generate computer graphics and video game content based on user behavior as the game is played. It represents a step beyond constrained randomization. While there are technologies for evolving content like pictures [19] or limited types of three-dimensional models [27], these technologies are not designed to work in real-time during a game; rather they require users to explicitly designate which items are the best, which is something that a user playing a game should not have to do. That is, constantly answering questions about what they like and what should be produced in the future would disrupt players' experience. In contrast, the cgNEAT method makes these decisions automatically based on implicit information within usage statistics.

The main principles of cgNEAT are as follows:

1) Each content item is represented by a CPPN. Different types of content can be represented by different CPPN input/output configurations (the specific representation for particle weapons is described later). In principle, a different representation than CPPNs can also be evolved.
2) During the game, each content item is assigned a fitness that is computed based on how often players actually *use* the content. That way, the system knows the relative popularity of each content item currently in the game.
3) Content is spawned in the game world, which means that it is placed in parts of the world where users can get it. However, unlike in most evolutionary systems, spawned content is not eligible for reproduction until players pick it up.
4) Content is reproduced in cgNEAT as follows: The algorithm selects content items from among content that players in the world *already possess* as parents that reproduce to form new content, which is spawned as described in step 3. The content items that are chosen as parents are selected probabilistically based on a roulette wheel scheme in which the chance of being chosen as a parent is proportional to the popularity (i.e. fitness) of the item. Reproduction, including mutation and crossover, is performed in accordance with the NEAT algorithm. Thus, there is a chance that CPPNs may become more complex than their parents.
5) For any new content that is spawned, there is a probability (selected by the designer) that it will be chosen from a *spawning pool*, which is a collection of pre-evolved content, instead of being reproduced from parents. This pool ensures that diversity is not lost and that good types of content from the past (i.e. those that users liked) might reappear. Additionally, it ensures an initial seed of good content when the game first starts and players' preferences are unknown. The game designers initially select content, which may be pre-evolved before the game is released, to put into the spawning pool.

The cgNEAT algorithm incorporates some mechanics of NEAT and standard evolutionary computation (EC), yet exhibits several major differences. Unlike in normal EC, the population size (i.e. those items that are eligible at any given time to reproduce) is variable and depends entirely on the number of users in the system. Furthermore, when an offspring is produced, unlike in normal evolutionary computation, it is not immediately placed into the population eligible to reproduce. Instead, it is in a special temporary state (placed somewhere in game world) in which it may join the population only if a player chooses to acquire it. Also unlike normal evolutionary computation, instead of fitness determining which items are eliminated from the population, users entirely determine which items leave the population simply by discarding them.

Unlike standard interactive evolutionary computation (IEC

Fig. 2. **Galactic Arms Race**. Players in GAR pilot their space ship (screen center) from a third person perspective. This picture demonstrates a player destroying enemies with an evolved corkscrew-shaped weapon. Left of the player ship is a weapon pickup dropped from a destroyed enemy base. A particle system preview emits from the weapon pickup (i.e. "neuralium isotope," left of player) to visually indicate how the weapon will function before the player picks it up. GAR is designed to look and feel like a near-commercial quality video game to effectively demonstrate the potential of automatic content generation in mainstream games.

[28]), users never explicitly communicate to the system which content they like. Instead, the preferred content is deduced by the system implicitly from natural human behavior. That is, users do not need to know that they are interacting with an evolutionary algorithm yet evolution still works anyway. Unlike regular NEAT, speciation is not necessary because users determine what is popular and the diversity of the population reflects the diversity of user preferences. Finally, every step of the cgNEAT algorithm is asynchronous. At any time players may cause content to join the population or be eliminated.

The next section details how cgNEAT is applied in practice to evolving weapons in the Galactic Arms Race video game.

IV. Galactic Arms Race (GAR)

In GAR (figure 2), the goal is to pilot a space ship to defeat enemies, gain experience, earn money, and most importantly, to find advantageous new weapons that are automatically generated by cgNEAT. GAR is intentionally designed to look and feel like a near-commercial quality video game so that it can convincingly demonstrate the promise of automatic content generation for mainstream games. To reach that level of quality, it took over a year to build by a nine-member mostly student team.

GAR is available online http://gar.eecs.ucf.edu. The game is a full multiplayer Internet platform in which servers evolve weapons based on the aggregate usage of all players online. However, this initial paper focuses on GAR's single-player mode, in which evolution is directed by the actions of a single player battling NPC aliens in the game, which are controlled by scripted steering behaviors [29] and the BOIDS algorithm [30].

Every weapon found in GAR is unique and players can continually find novel weapons with characteristics evolved from those weapons players favored in the past. It is important to note that weapons evolved in GAR all fire particle bursts with the same strength and number. Thus it is not sheer power that is evolving, but rather the pattern in which particles spray from the gun, which has complex tactical implications. Therefore, the space of weapons is not a total order from worst to best, but rather a complex multi-objective coevolutionary landscape.

Players are limited to three *weapon slots*, each of which holds a single weapon. Destroyed enemies and enemy bases may drop a *weapon pickup* that contains a novel weapon evolved by cgNEAT. Players choose in which weapon slot to place the new weapon, but doing so discards the existing weapon in that slot. Thus players must be selective about which weapons to keep. In this context, an important goal for any game that generates unpredictable content is to indicate what that content will be like before it is taken. To give players an idea of how a weapon functions before picking it up, weapon pickups emit a miniature particle system preview that behaves exactly as the actual weapon does. In the game this preview is called a *neuralium isotope* (figure 2, left side).

The remainder of this section details the integration of cgNEAT in GAR, including (1) CPPN representation, (2) calculating weapon fitness, (3) evolving new weapons, and (4) starter weapons and the spawning pool.

A. Particle System Weapon CPPNs

Particle system CPPNs in GAR are based on the techniques developed in NEAT Particles and NEAT Projectiles [26]. Each player weapon contains a single evolved CPPN (figure 3). Every frame of animation, each particle issued from the weapon inputs its current position relative to the ship (p_x, p_z) and distance from the ship (d_c) into the CPPN. There are two, rather than three, spatial inputs because the game is entirely situated on the $y = 0$ plane. The CPPN is activated and outputs the particle's velocity (v_x, v_z) and color (r, g, b) for that animation frame. Representing particle velocity and color in this manner produces a wide of variety of vivid patterns [26].

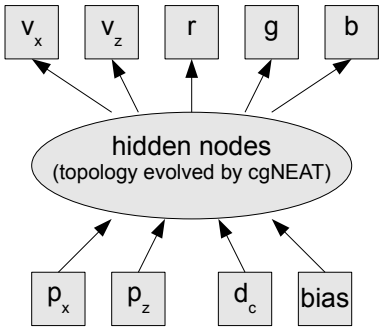
B. Calculating Weapon Fitness

Because it would disrupt the gameplay experience to query the player's opinion of every piece of content, weapon fitness is automatically calculated based on usage statistics. Players possess up to three weapons at one time. When a player fires a weapon, that weapon (which is a unique member of the population) gains fitness at a constant rate and the other weapons in that player's arsenal lose fitness at the same rate. There is no maximum fitness and the minimum fitness is 1.0. This *fitness decay* mechanism for unused weapons emphasizes emerging new weapon trends.

C. Evolving New Weapons

When players destroy an enemy base or a boss enemy, a new weapon is spawned either through reproduction within the current population or from the spawning pool. Any novel weapon created by cgNEAT is evolved from the current

(a)



(b)

Fig. 3. **How CPPNs Represent Particle Weapons**. (a) Each frame of animation, each particle separately inputs the position (p_x, p_z) and distance (d_c) from where it was *initially* fired into the CPPN (p_y is ignored because the game is situated entirely on the $y = 0$ plane). (b) The CPPN is activated and particle velocity (v_x, v_z) and color (r, g, b) are obtained from CPPN outputs. This method provides GAR with smooth particle animations and a wide variety of possible evolved weapons.

weapon population. In single-player GAR, the weapon population is only the three weapons the player currently holds. In multi-player GAR, the weapon population includes the weapons currently held by all players. Thus single-player evolution is to some extent greedy; however, it is not equivalent to a normal evolutionary algorithm with a population of three because the player encounters a significant number of weapon previews *in addition* to the weapons in the ship's current arsenal. Therefore, the player is in effect judging such previews by taking them or not. Furthermore, the spawning pool ensures a diverse set of jumping-off points are injected at regular intervals. As results in this paper show, the net effect is that a single player can genuinely discover a diverse array of highly specialized and effective weapons.

The roulette method, based on weapon fitness, decides which weapon reproduces. Figure 4 illustrates weapon evolution in GAR with two genealogies of related weapons.

D. Starter Weapons and the Spawning Pool

When the game begins players have no history of weapon preference. One possible policy is to initially give players three random weapons. However, such randomization could cause new players to receive three undesirable weapons. A better solution is for players to begin the game with a predefined set of *starter weapons*. The starter weapons in GAR (1) shoot only in a straight line, and (2) are not eligible to reproduce during evolution. Thus, new players are guaranteed to begin with viable weapons.

Because starter weapons cannot reproduce and players begin the game with only starter weapons, a method is needed to start evolution. For this purpose, the *spawning pool* is a diverse collection of good weapons evolved by the game developers. If cgNEAT selects a starter weapon to reproduce because it is fired often, a random spawning pool weapon is spawned instead. The advantages of the spawning pool are (1) it jump starts evolution at the beginning of the game and (2) it enables developers to influence what weapons players will see early on, which is a critical time to make a good first impression on players. The spawning pool can also serve as a *hall of fame*, to which popular weapons are retired, possibly reappearing later in game.

The next section describes the experience of weapon evolution in the game and presents examples of weapons evolved by players.

V. Playing GAR

The aim of the experiment in this section is to determine whether GAR can produce a convincing variety of weapons both tactically and aesthetically. To investigate the creativity of GAR in single-player mode, a group of ten test players piloted space ships in separate games for at least one hour each. The results in this section (including figure 5) are from these test sessions. The main result is that players indeed discovered a variety of genuinely unique weapons with compelling tactical implications and aesthetics.

As the weapons showcased in this section will show, the gameplay implications of evolved content sometimes seem intentional, as if designed purposely to create a specific capability. Thus it is important to keep in mind that *all* the weapons are entirely invented by the game itself with no forethought by the game creators. In many cases powerful guns were invented that were unlike anything the developers had seen or imagined before. They often exhibit both appealing tactical and aesthetic (through changing color patterns) qualities. Yet of course these guns are not the result of random luck either; just as in other evolutionary algorithms, they result from selection pressure, which is wrought by the preferences of the player in GAR. In this way, GAR is a credible demonstration of the potential of this approach.

In GAR it is possible for player projectiles to intercept enemy projectiles. Therefore, several key tactical trade-offs are explored by evolution. Slow projectiles make it easier to block incoming fire whereas fast projectiles are easier to aim at distant enemies. Weapons with a wide spread are more effective at blocking incoming projectiles; however, concentrated patterns more effectively destroy distant targets quickly. Hybrid weapons with variable spread pattern and speed over time evolve as well. Yet these tactical principles are only the beginning. In fact, figure 5 presents a sample of the wide range of generated weapons and describes some

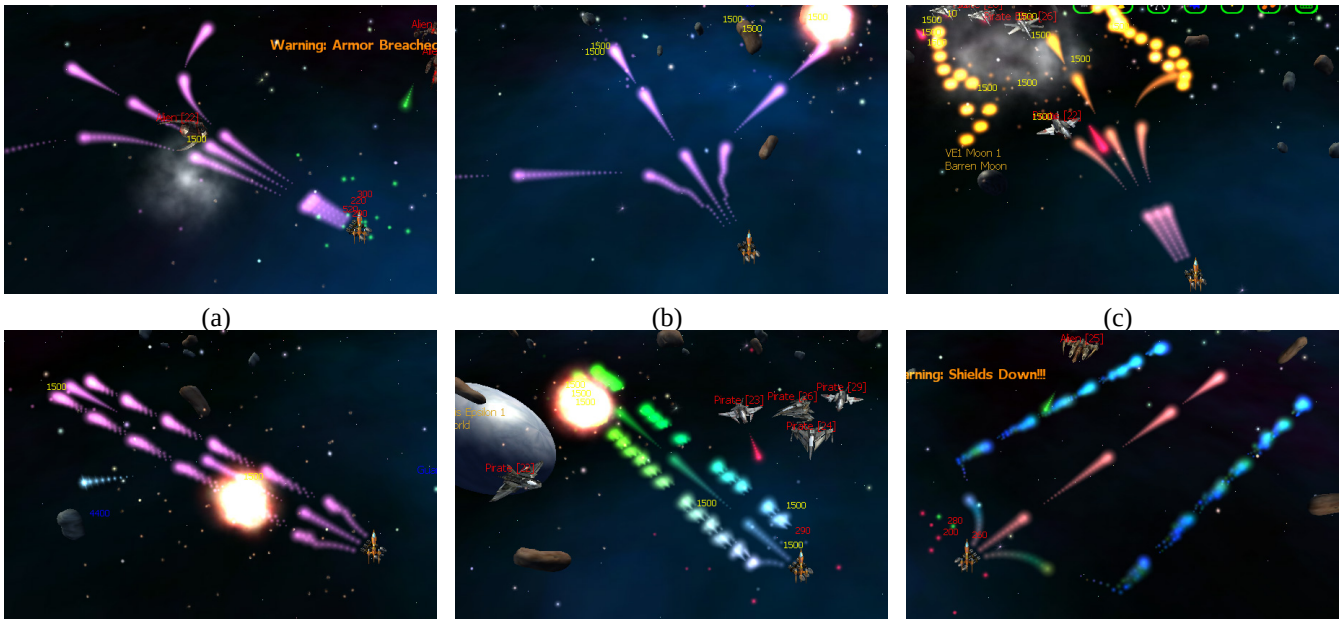


Fig. 4. **Weapon Evolution Examples**. As weapons evolve over the course of the game, players are likely to find weapons with qualities similar to those they favored in the past. In this example from actual gameplay, the player often fired a spread weapon (a). Later in the game, new spread gun variations (b,c) evolved. Another interesting spread gun (d) fires two slower-firing outer projectiles and a fast inner projectile. Later descendants of this weapon (e,f) exaggerated the speed difference between the inner and outer projectiles, diversified the color pattern, and modified the spread width. These examples illustrate how cgNEAT evolves novel content based on past user preferences.

of their tactical implications. To highlight the creativity of cgNEAT, we have assigned descriptive names to each such gun to help to more easily appreciate their concept. Two especially interesting evolved weapon types are *wallmakers* (figure 5j,k), which literally create a wall of particles in front of the player, and *tunnelmakers* (figure 5h,l), which create a line of particles on either side of the player. Both weapon types are defense-oriented, enabling players to switch between them and more offense-oriented weapons, as the tactical situation dictates. Most importantly, the authors had never conceived of such guns, yet cgNEAT invented them. These examples demonstrate that cgNEAT evolves unique and tactically diverse weapons as the game is played.

Finally, it is important to point out that it does not take long for players to begin to find effective weapons. As figure 5 shows, compelling weapons often arise within the first ten generations (e.g. the *tunnelmaker* is from generation two). Furthermore, weapons continue to evolve into novel forms over dozens of generations, such as the *blue ladder* (figure 5f) from generation 42.

The next section discusses implications and other possible applications of cgNEAT.

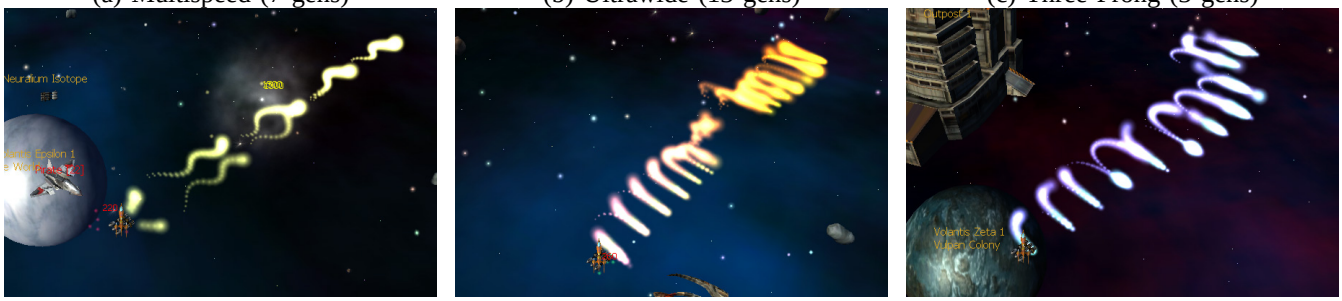
VI. Discussion and Future Work

GAR demonstrates that automatic content generation is a viable new technology. The main application is in simulations and games wherein the designers want users to be able to discover and experience a continual stream of new content beyond what the original artists and programmers are able to provide. For players the main implication is a new kind of experience in which not only is novelty a constant, but the pursuit of novelty itself is an integral part of the game. In fact, players informally indicated enjoying the consistent satisfaction of novel discovery. For some game designers, this loss of control will be viewed as a risky sacrifice; yet others will see it for its potential, just as any new frontier opens up an unknown world of possibilities. In fact, the interactive evolutionary dynamic automatically creates a kind of implicit game balance because, as soon as a player acquires a weapon that tips the equilibrium, variants of that weapon become available to other players in proportion to its use, thereby continually balancing the game.

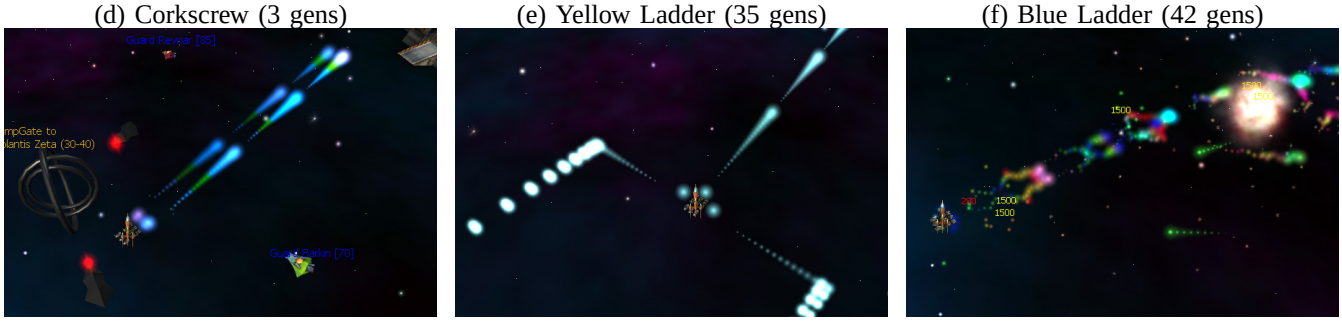
In addition to weapons, a wide variety of other game content could potentially be generated by cgNEAT including two-dimensional textures, three-dimensional models, other types of particle effects, and programmable shader effects. Video games that automatically generate their own content (e.g. characters, clothing, weapons, houses, vehicles, music, special effects, etc.) could keep players engaged much longer in such a constantly evolving game world than in a static one. Thus the potential future applications are broad.

At the time of this publication, GAR has recently been released as a multiplayer Internet game. While the single player results in this paper demonstrate the promise of the idea, full-blown multiplayer evolution on persistent servers can yield a significantly broader explosion of content. Furthermore, the tactical implications of human players fighting *each other* (instead of robotic enemies) with a constantly changing arsenal promises to produce a coevolutionary effect heretofore never

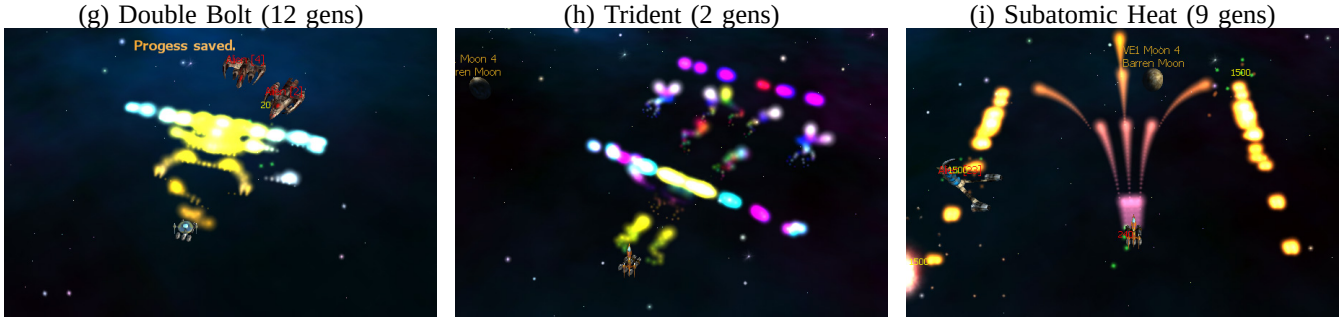
(a) Multispeed (7 gens) (b) Ultrawide (13 gens) (c) Three Prong (3 gens)



(d) Corkscrew (3 gens) (e) Yellow Ladder (35 gens) (f) Blue Ladder (42 gens)



(g) Double Bolt (12 gens) (h) Trident (2 gens) (i) Subatomic Heat (9 gens)



(j) Wallmaker (14 gens) (k) Double Wallmaker (15 gens) (l) Tunnelmaker (2 gens)

Fig. 5. **Weapons Evolved During Gameplay**. GAR players discovered many useful and aesthetically pleasing weapons. The number of generations of reproduction taken to evolve each weapon is shown next to its name. The *multispeed* (a) fires two slow outer projectiles, which are useful for blocking incoming enemy fire, and a fast center projectile for quickly striking distant targets. The *ultrawide* (b) and *three prong* (c) emit wide particle patterns that are effective for fighting many enemies at once. The *corkscrew* (d) emits a pattern that is initially wide, for blocking, but later converges for concentrated damage at a distance. Two version of the *ladder gun* (e,f) fire a wide wave-like pattern that can swivel around obstacles like asteroids. The *double bolt* (g) demonstrates that weapons similar to those in typical space shooters can evolve. The *trident* (h) launches a single projectile forward and two perpendicular projectiles that can block enemy fire from the sides. *Subatomic heat* (i) fires a chaotic multi-colored stream resembling bouncing subatomic particles. Two types of *wallmaker* (j,k) literally create defensive walls of particles in front the player. The *tunnelmaker* (l) creates a defensive line of particles as well, but on both sides of the player, yielding a defensive sheath. These results demonstrate the ability of cgNEAT to generate a tactically and aesthetically diverse and genuinely useful array of weapons. Furthermore, useful weapons appear in early generation and continue to elaborate over successive generations.

experienced in video games. Already, the GAR client has been downloaded over 8,500 times, appeared on several Internet news sites including *Slashdot* (http://games.slashdot.org/story/09/07/08/1419242/Experimental-Video-Game-Evolves-Its-Own-Content), and over 1,000 players have registered with the 32-player official server run by our research group. A broad array of evolutionary statistics are being collected from the server and will be reported in future publications.

VII. Conclusion

This paper presented cgNEAT, an algorithm that automatically generates game content based on perceived user preferences, as the game is played. In cgNEAT, unlike standard evolutionary algorithms, selection for reproduction is controlled implicitly by player behavior within the game. That is, content players utilize often is more likely to reproduce. The result is a constant stream of novel content suited to players' tastes. The first implementation of cgNEAT was demonstrated in the single-player mode of Galactic Arms Race, a game in which particle system weapons evolve. The success of GAR suggests the potential of cgNEAT, and automatic content generation in general, to generate a myriad of other types of content. For players, such a novel content stream can potentially significantly increase game replay value, keeping players engaged in the evolving world. For the game industry, it means that it is possible to build games that automatically create their own content to satisfy users, possibly impacting the way games are made.

Acknowledgments

Special thanks to the GAR volunteer team: Nathan Sriboonlue (NPC control), Jaruwan Mesit (soft-body models), Fabian Moncada (music and sound effects), John Martin (additional design and testing), Derrick Janssen (additional design), Kristen Martin (additional database programming), Eric Isles (additional music and sound effects), Gordon Hart (additional music), Jonathan "Zarcath" Chan (additional design), FourTwoOmega (additional music), and JRWR (webstats). GAR was developed with the Microsoft XNA Game Studio SDK at the Evolutionary Complexity Research Group (Eplex) at University of Central Florida (http://eplex.cs.ucf.edu). GAR is available at http://gar.eecs.ucf.edu and the project's official email address is gar@eecs.ucf.edu.

References

[1] R. Edwards, "The economics of game publishing," *IGN Entertainment*, 2006. [Online]. Available: http://games.ign.com/articles/708/708972p1.html

[2] M. J. Irwin, "Game developers' trade off," *Forbes.com*, 2008. [Online]. Available: http://www.forbes.com/2008/05/27/videogame-art-money-tech-personal-cx_mji_0528vgames.html

[3] V. Software, "Source engine SDK," 2007. [Online]. Available: http://developer.valvesoftware.com

[4] E. Games, "Unreal engine SDK," 2007. [Online]. Available: http://www.unrealtechnology.com/

[5] K. O. Stanley, B. D. Bryant, and R. Miikkulainen, "Real-time neuroevolution in the NERO video game," *IEEE Transactions on Evolutionary Computation Special Issue on Evolutionary Computation and Games*, vol. 9, no. 6, pp. 653–668, 2005.

[6] J. Togelius, R. D. Nardi, and S. M. Lucas, "Towards automatic personalised content creation for racing games," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2007.

[7] ——, "An experiment in automatic game design," in *Proceedings of the IEEE Symposium on Computational Intelligence and Games*, 2008.

[8] F. Gomez and R. Miikkulainen, "Solving non-Markovian control tasks with neuroevolution," in *Proceedings of the 16th International Joint Conference on Artificial Intelligence*. San Francisco: Kaufmann, 1999, pp. 1356–1361.

[9] N. Saravanan and D. B. Fogel, "Evolving neural control systems," *IEEE Expert*, pp. 23–27, June 1995.

[10] F. Gruau, D. Whitley, and L. Pyeatt, "A comparison between cellular encoding and direct encoding for genetic neural networks," in *Genetic Programming 1996: Proceedings of the First Annual Conference*, J. R. Koza, D. E. Goldberg, D. B. Fogel, and R. L. Riolo, Eds. Cambridge, MA: MIT Press, 1996, pp. 81–89.

[11] B.-T. Zhang and H. Muhlenbein, "Evolving optimal neural networks using genetic algorithms with Occam's razor," *CplxSys*, vol. 7, pp. 199–220, 1993.

[12] X. Yao, "Evolving artificial neural networks," *Proceedings of the IEEE*, vol. 87, no. 9, pp. 1423–1447, 1999.

[13] A. P. Martin, "Increasing genomic complexity by gene duplication and the origin of vertebrates," *The American Naturalist*, vol. 154, no. 2, pp. 111–128, 1999.

[14] J. D. Watson, N. H. Hopkins, J. W. Roberts, J. A. Steitz, and A. M. Weiner, *Molecular Biology of the Gene Fourth Edition*. Menlo Park, CA: The Benjamin Cummings Publishing Company, Inc., 1987.

[15] L. Altenberg, "Evolving better representations through selective genome growth," in *Proceedings of the IEEE World Congress on Computational Intelligence*. Piscataway, NJ: IEEE Press, 1994, pp. 182–187.

[16] I. Harvey, "The artificial evolution of adaptive behavior," Ph.D. dissertation, School of Cognitive and Computing Sciences, University of Sussex, Sussex, 1993.

[17] K. O. Stanley and R. Miikkulainen, "Evolving neural networks through augmenting topologies," *Evolutionary Computation*, vol. 10, pp. 99–127, 2002.

[18] M. E. Taylor, S. Whiteson, and P. Stone, "Comparing evolutionary and temporal difference methods in a reinforcement learning domain," in *GECCO 2006: Proceedings of the Genetic and Evolutionary Computation Conference*, July 2006, pp. 1321–1328.

[19] J. Secretan, N. Beato, D. B. D'Ambrosio, A. Rodriguez, A. Campbell, and K. O. Stanley, "Picbreeder: Evolving pictures collaboratively online," in *Proceedings of the Computer Human Interaction Conference*, 2008.

[20] T. Aaltonen *et al.*, "Measurement of the top quark mass with dilepton events selected using neuroevolution at CDF," *Physical Review Letters*, 2009, to appear.

[21] K. O. Stanley, "Exploiting regularity without development," in *Proceedings of the AAAI Fall Symposium on Developmental Systems*. Menlo Park, CA: AAAI Press, 2006.

[22] ——, "Compositional pattern producing networks: A novel abstraction of development," *Genetic Programming and Evolvable Machines Special Issue on Developmental Systems*, pp. 131 – 162, 2007.

[23] A. Hoover, , and K. O. Stanley, "Exploiting functional relationships in musical composition," *Connection Science Special Issue on Music, Brain, and Cognition*, 2009, to appear.

[24] J. Lander, "The ocean spray in your face," *Game Developer Magazine*, pp. 13–20, July 1997.

[25] J. V. der Berg, "Building an advanced particle system," *Game Developer Magazine*, pp. 44–50, March 2000.

[26] E. Hastings, R. Guha, and K. O. Stanley, "Interactive evolution of particle systems for computer graphics and animation," *IEEE Transactions on Evolutionary Computation*, 2009.

[27] K. Sims, "Evolving virtual creatures," in *Proceedings of the ACM Special Interest Group on Graphics and Interactive Techniques*, 1994, pp. 50–62.

[28] H. Takagi, "Interactive evolutionary computation: Fusion of the capacities of EC optimization and human evaluation," *Proceedings of the IEEE*, vol. 89, no. 9, pp. 1275–1296, 2001.

[29] C. Reynolds, "Steering behaviors of autonomous characters," in *Proceedings of the Game Developers Conference*, 1999, pp. 763–782.

[30] ——, "Flocks, herds, and schools: A distributed behavioral model," in *Proceedings of the 14th Annual Conference on Computer Graphics and Interactive Techniques*, 1987, pp. 25 – 34.