Team Blockhead Wars: Generating FPS Weapons in a Multiplayer Environment

Eric McDuffee
Department of Computer Science
SUNY Oswego
Oswego, NY 13126
mcduffee@oswego.edu

Alex Pantaleev
Department of Computer Science
SUNY Oswego
Oswego, NY 13126
alex@cs.oswego.edu

ABSTRACT

We present an attempt at exploring the search space of weapons in team-based multiplayer First-Person Shooters (FPS). At the foundation of the experiment is Team Blockhead Wars (TBHW), a game that we developed for the purposes of this project. TBHW allows human players to enjoy classic multiplayer FPS gameplay and uses a genetic algorithm to continuously generate new weapons. A weapon's genome consists of ten real-valued parameters, which together form a vast search space that includes common FPS weapon tropes. The evaluation function scores weapons on the basis of their use by players. The game also generates 3D meshes to visually represent the generated weapons for easy player recognition. While TBHW is work in progress, preliminary results are encouraging.

Categories and Subject Descriptors

K.8.0 [**Personal Computing**]: General—*Games*

General Terms

Design, Experimentation

Keywords

Games, procedural content generation, game design, first-person shooters

1. INTRODUCTION AND BACKGROUND

FPS has continuously been one of the most popular game genres since its inception more than twenty years ago. Its success was boosted by the introduction of team-based networked multiplayer games at the start of the previous decade. Unfortunately, few innovative multiplayer FPS games have been released recently: the multiplayer gameplay experience is nearly identical to that of a decade ago, partly due to the petrification of existing FPS weapon tropes and the lack of desire by developers to risk with innovative weapon designs. This unfortunate state is not helped by the amount of resources required to create a high-quality FPS, not to mention a multiplayer one, meaning that independent developers are unlikely to innovate in the genre.

This paper describes an experiment to explore the search space of weapons in team-based multiplayer FPS games, with the hope of uncovering innovative weapon patterns. To this end we are in the final stages of building Team Blockhead Wars (TBHW), a multiplayer FPS that uses an evolutionary algorithm to search for optimal combinations of weapon parameters. The interactive and implicit evaluation function scores weapons on the basis of their use by players while they fight in one of the several multiplayer arenas. A central server monitors the fitness value of all equipped weapons and procedurally creates new ones, using both an archive of previously used weapons and the ones currently in use. Players collect new weapons as pickups from the arena, and can replace a weapon they are currently using with one they have picked up before they respawn in the game. TBHW also generates 3D meshes that correspond to a generated weapon's parameters to facilitate its easy recognition by players.

1.1 Situating the Algorithm

TBHW generates new weapons procedurally using an evolutionary search technique. Hence, it can be categorized within the field of search-based procedural content generation, of which Togelius et al. [17] published an extensive taxonomy and survey. According to the distinctions the taxonomy makes, the game presented here uses a direct encoding in the genotype-to-phenotype mapping, since the genome variables (the genotype) map to attributes of the created weapons (the phenotype). Furthermore, the content generation algorithm is online, since it is performed during the runtime of the game. The algorithm ranks weapons on the basis of their usage by players, which makes its evaluation function interactive and implicit.

1.2 Search-Based PCG in Games

TBHW allows players to continuously equip their characters with new procedurally generated weapons. Similar attempts at this type of content generation have been made in the past, most prominently in *Galactic Arms Race* (GAR), a two-dimensional multiplayer arcade shooter by Hastings et al. [4]. GAR uses cgNEAT, a search-based algorithm, to evolve the connection topology of a neural network for each weapon's particle system [3]. Our work, which was partly inspired by GAR, expands on procedural generation of weapons within the dissimilar genre of FPS, which im-

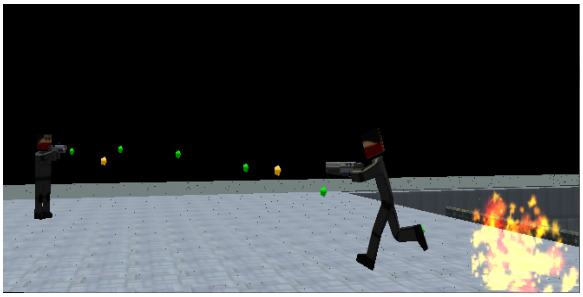

Figure 1: TBHW in action

poses its own constraints on both the algorithms used and the content to be generated.

Borderlands and *Borderlands 2* (Gearbox Software 2009 and 2012, respectively) are commercial FPS games in which weapons are procedurally generated. However, unlike in GAR and our work, in these games weapons are randomly created and then balanced according to the player's level. Weapons in the Borderlands series are also constrained within specified types and mechanics, with predefined meshes for each type, whereas in our work weapon types, mechanics, and their representation are dependent on the evolutionary algorithm.

Search-based PCG for games is a new and promising field. Successful applications include generating tracks and levels [6, 10, 12, 14], terrain and maps [11, 15], buildings [8], camera control [2], unit types and abilities [7, 9], and rule systems and mechanics for games [1, 5, 13, 16].

2. EXPERIMENTAL SETUP

TBHW closely models the tropes of a standard team-based multiplayer FPS. The game allows human players to choose weapons for their avatars before joining a multiplayer FPS arena. Player login information, as well as inventory (which includes favorite weapons), is stored in a centralized information server, which is different from the multiplayer server. Thus several multiplayer battles can take place simultaneously. Each multiplayer server continuously exchanges pertinent data with the central information server. Figure 1 depicts a two-player shootout.

TBHW implicitly keeps a fitness score of each weapon, based on two parameters: the amount of time a weapon is equipped, and the number of non-suicide kills achieved with that weapon. This fitness score is used by the information server to evolve new weapons and send the batches of evolved weapons to any connected multiplayer servers, which then place weapon pickups on the battlefield. The 3D meshes for the pickups are smaller versions of the meshes used in the weapon selection screen, which allows players to recognize the parameters of the weapons they are picking up for later use.

When a player avatar's health points reach zero, the avatar is removed from the game arena, and the player must wait for fifteen seconds to rejoin. A player can also choose to enter this fifteen-second respawn period at any time; a player's avatar effectively dies when that happens, with no kill recorded. During this period the player has the chance to replace any of the two previously equipped weapons with other weapons

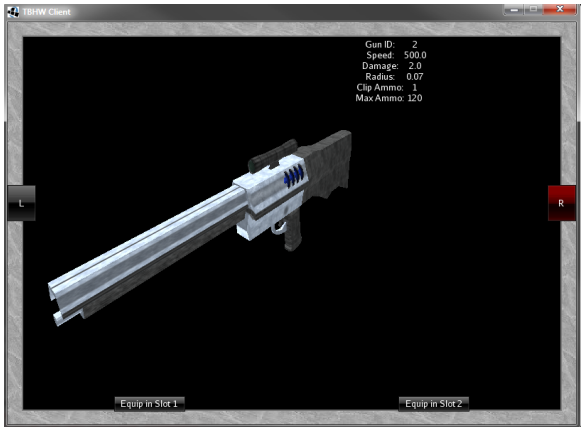

Figure 2: Weapon selection interface

that were evolved by the server and picked up during gameplay. The replacement process is facilitated by a weapon interface that shows an enlarged 3D model of each weapon. A weapon's mesh is generated, stretched and modified to directly represent the weapon's parameters, helping quick user interaction and weapon recognition. For example, the length of a weapon's barrel represents the speed of its projectile. The same weapon meshes, representative of the capabilities of the weapon, are also used in the arena, both for currently equipped weapons and for weapon pickups. Figure 2 shows an example mesh of a sniper-like weapon (long and narrow barrel plus scope and a tiny magazine) as seen in the weapon selection interface.

2.1 Game Mechanics

Each avatar is represented as an in-game 3D character. On the multiplayer server avatars are collections of bounding boxes, defined by their mesh skeletons, giving almost mesh-accurate collision detection. The damage a projectile inflicts on an avatar is multiplied by a given bounding box's damage multiplier (e.g., the head takes twice the normal amount of damage).

When a weapon is fired, a projectile is created within the game world. Every projectile is a sphere with a certain radius. Projectiles are affected by physics and can bounce off of players, other projectiles, and the environment. After a certain number of bounces the projectile detonates and damages any players within a certain radius.

2.2 Content Representation and Evolutionary Algorithm

Weapons and their projectiles are a central concept to the game. Each weapon has a set of parameters, which were chosen based on their ability to create weapons common to the FPS genre. Those are:

- Projectile speed—the initial speed of a projectile exiting the weapon.
- Projectile size—the size of the projectile (sphere radius).
- Projectile gravity—the effect that gravity has on the projectile, as a proportion of normal gravitational pull.

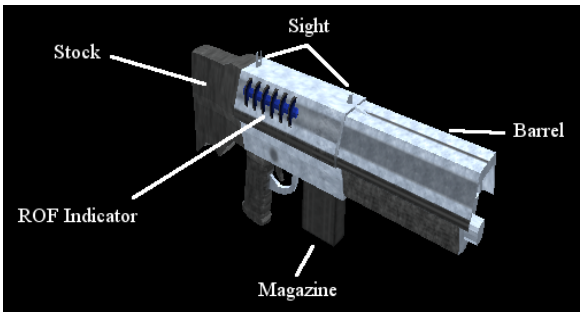

Figure 3: Parts of a weapon mesh

- Projectile damage—the amount of damage a projectile inflicts on other players.
- Projectile damage radius—the radius within which a projectile inflicts damage (i.e., splash damage).
- Projectile bounce amount—a threshold for the number of bounces a projectile can endure before detonating (to model grenade launchers and the like).
- Weapon RoF—the rate of fire of the weapon, as defined by the minimum interval between shots.
- Weapon magazine size—the number of times the weapon can be fired before requiring a reload.
- Weapon maximum ammunition—the amount of ammunition the player can carry for a weapon.
- Weapon reload time—the amount of time necessary to reload a fully empty magazine.

Parameters cannot evolve outside of a preset range for each parameter.

These ten parameters comprise a weapon's genome and are used to evolve additional weapons. In order to ensure a continuously changing and diverse population, TBHW's central information server keeps a population of weapons above a certain fitness value. A weapon's fitness value, as described previously, is calculated as a function of the weapon's usage by players and the number of kills achieved with it. Since this population is stored centrally, it is persistent across game instances and game arenas. New players are given two preset weapons to start with, which do not contribute to evolution. If the central server does not have an archive of previously evolved weapons, it randomly generates a set of weapons for initial evolution and balances them.

The central server creates a new generation of weapons every ten minutes while players are logged in. To achieve that it uses standard fitness proportionate selection and discrete recombination, followed by a low-probability mutation. Then the offspring are balanced, constraining the total damage per second they can achieve. Balancing is an important step in the process to ensure that new generations of weapons do not converge on a hypothetical best weapon, which would have all of its parameters maximized. Due to the direct encoding in the genotype-to-phenotype mapping, the balancing function effectively modifies the genome of the weapon.

As described previously, the offspring are then used by currently running multiplayer servers to place weapon pickups on the arena.

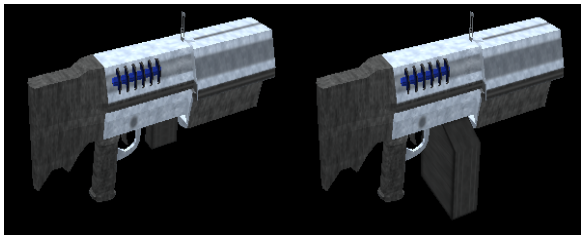

Figure 4: Representations of two similar weapons

2.3 Graphical Representation

Players must be able to quickly differentiate between weapon and projectile traits based on each weapon's visual model in order to effectively serve as the evaluation function of the genetic algorithm. To facilitate easy player recognition of desired traits, we used the parameters of a weapon to define its 3D mesh; parts of a weapon's mesh are depicted in Figure 3. We then use this mesh to represent the weapon visually whenever a player is able to see it (as a pickup, equipped by an avatar, or in the weapon selection screen).

Rather than simply offer discrete mesh changes based on parameter values, such as those found in the *Borderlands* series, we used continuous mesh modifications for most parameters. We also tried to sensibly match weapon parameters to mesh parameters, following real-world associations where possible, in order to maximize our interface's convenience to users. As an example, the weapon's barrel and stock are elongated or shortened to visually represent the speed of the weapon's projectile. The barrel's diameter is modified to reflect the projectile's size. We use the weapon's sight to represent the influence of gravity on a projectile: as the projectile's speed decreases and it becomes more susceptible to gravity, the sight gradually becomes more reminiscent of a grenade launcher's one. Conversely, as the projectile's speed increases, the weapon's sight morphs into iron sights, and finally a scope.

We used the visual size of the magazine to represent a weapon's magazine ammunition, as well as its maximum allowed ammunition. We also display a ROF indicator on the sides of each weapon. Finally, we use the color of projectiles to display their damage as a gradient from green to red, and a simple particle effect surrounding a projectile to represent its damage radius. Figure 4 represents two grenade launcher-like weapons (wide and short barrels and a launcher sight) with different magazine ammunition and maximum ammunition.

3. RESULTS AND DISCUSSION

Even though the game is still in development, it has already shown in preliminary tests that it is capable of generating interesting weapons. As an example, we accidentally evolved a weapon with a high initial projectile speed, a large damage radius, and a bounce amount of two. This allowed the player who had equipped the weapon to stand two feet behind a wall, shoot at the wall from a certain angle, and effectively shell the other side of the map. By the time the projectiles arrived there they would be dropping almost vertically. This was a completely unexpected weapon that our algorithm generated; we started with a combination of parameters that

achieved the effect of a grenade launcher, and in the end had a weapon resembling a mortar, which we had not thought to model before.

Another interesting weapon that the game generated had a projectile speed slower than the running speed of a player's avatar, which made players regularly run into their own projectiles. This, combined with the fact that enemy players could easily outrun the bullets, initially made us dismiss the weapon. However, after a few evolutionary iterations we accidentally uncovered a weapon that combined slow projectiles with a low gravity effect on them, high damage, and a very high magazine capacity. This turned out to be a nearly perfect defensive weapon, since a player could use it to blanket a hallway with slow moving projectiles, then switch to the other equipped (offensive) weapon and move in the opposite direction, knowing that no enemy could sneak from behind.

The emergence of these weapons is very encouraging at this early stage. When the game is finalized and we test it with real (non-developer) users, we anticipate that their fresh perspectives will guide TBHW to generate even more interesting weapons.

4. CONCLUSIONS AND FUTURE WORK

This paper described a project in progress that explores the search space of FPS weapons through procedural content generation. While incomplete, TBHW has already shown some promise towards that end.

After the project is complete and we collect test data from real-world users, we intend to expand significantly upon the search space of the evolutionary algorithm by introducing an additional set of parameters, such as negative damage (allows healing of friendly players), multiple projectiles per shot, and weapon accuracy (bullet spread). In terms of interface improvements, we intend to attempt generating textures that represent weapon parameters in addition to meshes. Another improvement is the inclusion of various kinds of levels or maps (either human-created or procedurally generated), which would allow players to explore new subsets of the search space: a battle in an open arena is very different from a battle in a maze of twisted, narrow hallways.

Finally, in the long term we intend to introduce enumerated variables in the weapon genome to signify optional traits, which will allow us to model various additional FPS tropes such as mines and other traps. We also intend to include traits that impact avatars in non-traditional ways, e.g., increase or reduce the speed of all nearby players (presumably members of the same team).

5. REFERENCES

[1] C. Browne. *Automatic generation and evaluation of recombination games*. PhD thesis, Queensland University of Technology, 2008.

[2] P. Burelli and G. Yannakakis. Combining local and global optimisation for virtual camera control. In *IEEE Symp. Computational Intelligence and Games*, 2010.

[3] E. Hastings, R. Guha, and K. Stanley. Neat particles: Design, representation, and animation of particle system effects. In *IEEE Symp. Computational Intelligence and Games*, 2007.

[4] E. Hastings, R. Guha, and K. Stanley. Automatic content generation in the galactic arms race video game. *IEEE Transactions on Computational Intelligence and AI in Games*, 1(4):245–263, 2009.

[5] V. Hom and J. Marks. Automatic design of balanced board games. In *AAAI Conf. Artificial Intelligence and Interactive Digital Entertainment*, 2007.

[6] R. Lopes, T. Tutenel, and R. Bidarra. Using gameplay semantics to procedurally generate player-matching game worlds. In *The Third Workshop on Procedural Content Generation in Games*, 2012.

[7] T. Mahlmann, J. Togelius, and G. Yannakakis. Towards procedural strategy game generation: Evolving complementary unit types. *Applications of Evolutionary Computation*, pages 93–102, 2011.

[8] A. Martin, A. Lim, S. Colton, and C. Browne. Evolving 3d buildings for the prototype video game subversion. *Applications of Evolutionary Computation*, pages 111–120, 2010.

[9] A. Pantaleev. In search of patterns: Disrupting rpg classes through procedural content generation. In *The Third Workshop on Procedural Content Generation in Games*, 2012.

[10] C. Pedersen, J. Togelius, and G. Yannakakis. Modeling player experience for content creation. *IEEE Transactions on Computational Intelligence and AI in Games*, 2(1):54–67, 2010.

[11] W. L. Raffe, F. Zambetta, and X. Li. Evolving patch-based terrains for use in video games. In *Conf. Genetic and Evolutionary Computation*, pages 363–370, 2011.

[12] N. Shaker, J. Togelius, G. Yannakakis, et al. The 2010 mario ai championship: Level generation track. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(4):332–347, 2011.

[13] A. Smith and M. Mateas. Variations forever: Flexibly generating rulesets from a sculptable design space of mini-games. In *IEEE Symp. Computational Intelligence and Games*, 2010.

[14] N. Sorenson, P. Pasquier, and S. DiPaola. A generic approach to challenge modeling for the procedural creation of video game levels. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):229–244, 2011.

[15] J. Togelius, M. Preuss, N. Beume, S. Wessing, J. Hagelback, and G. Yannakakis. Multiobjective exploration of the starcraft map space. In *IEEE Symp. Computational Intelligence and Games*, 2010.

[16] J. Togelius and J. Schmidhuber. An experiment in automatic game design. In *IEEE Symp. Computational Intelligence and Games*, 2008.

[17] J. Togelius, G. Yannakakis, K. Stanley, and C. Browne. Search-based procedural content generation: A taxonomy and survey. *IEEE Transactions on Computational Intelligence and AI in Games*, 3(3):172–186, 2011.