DOI: http://dx.doi.org/10.7551/978-0-262-31709-2-ch084

GOLEM: Generator Of Life Embedded into MMOs

Andrea Guarneri[1], Dario Maggiorini[1], Laura A. Ripamonti[1] and Marco Trubian[1]

[1]Dept. of Computer Science – Università di Milano - Italy
ripamonti@di.unimi.it

Abstract

Massively Multi-player Online Role-Playing Games and Massively Multiplayer Online games are complex and costly system from both a game design and a technical point of view. Among their many issues, in this paper we tackle the problem of incrementing players thrill by supplying them plenty of freshly produced, unpredictable monsters. To achieve this goal we have designed, developed and tested GOLEM, a genetic-based approach to the evolution of new species of monsters for video games.

1. Introduction

In Massively Multi-player Online Role-Playing Games (MMORPGs) and in Massively Multiplayer Online games (MMOs) players interact among them in an online, persistent, and shared virtual world. This game genre has gained success and diffusion especially thanks to the prosperity of Blizzard's World of Warcraft (WoW) (Taylor, 2006). The huge amount of people that interact on an ongoing basis (e.g., more than 10 million in WoW) in a shared environment raises questions for game designers and computer scientists that have scarcely been investigated until now, but that could nonetheless affect the success and survival of a specific game.

Beside the issues related to the game mechanics and the service supply, there is a number of other – only apparently – minor, "tricky" features that could, nonetheless, impact deeply on players' satisfaction with the game (see e.g., Bartle, 2003; Maggiorini et al., 2012a, 2012b). Among these neglected issues, we can enlist several characteristics that are intrinsic to the paths players have to follow to raise the "level" of their characters and that could rise problems of players' loyalty. In particular, to explore the in-game world, complete quests, and gain "experience points", players are often confronted with different types of monsters (also called "mobs", which stands for "mobiles"), which they are supposed to slain. In spite of the fact that a game world can contain hundreds of different species of monsters, after spending a certain amount of time playing, players become well aware of the characteristics presented by each specie and its related hazard. In the long run, this knowledge has the drawback of generating a certain amount of boredom in players, which lose the thrill of braving unfamiliar dangers (Koster, 2004).

In this work, we tackle this issue by proposing GOLEM – Generator Of Life Embedded into MMOs: an algorithm aimed at increasing variety and unpredictability among mobs in MMOs, by means of an approach rooted into Genetic Algoritms (GAs). The main idea is to represent each monster specie in the population present in the in-game world through its genome, and to generate new species by recombining their chromosomes in a way quite similar to what happens in the natural world. This implies also taking into account aspects like the actual possibility for a mob to survive the habitat in which it was born (e.g., a marine-like animal with fins will unlikely survive in a desert), providing an estimate of the population growth (in order to avoid overpopulation), some means to contain the mobs numerosity when needed, etc. Nonetheless, our aim is not to recreate a complex ecosystem, since this will go far beyond the scopes of a generator of monsters for a video game, whose main goal should simply be to increase the players' fun.

The paper is organized as follows: the following Section 2 briefly analyses related works, from both the academy and the industry, while Section 3 recalls the fundamental concepts related to Genetic Algorithms (GAs). The subsequent Section 4 summarizes how we have designed the chromosomes to describe the most diffused fantasy monsters. Section 5 concentrates on the issues related to managing a population of evolving monsters through several generations. Sections 6 and 7 focuse, respectively, on the implementation of the GOLEM algorithm and on some perspective results derived from several tests. Finally, Section 8 draws conclusions and delineates major future developments.

2. Related works (in video games)

Although scholarly literature on the applications of GAs and Genetic Programming (GP) in video games is quite huge and multi-faceted, in our knowledge, until now it has focused on scopes different from creating diversity among mobs. In particular, the major part of recent works has tackled either the use of GAs to generate or evolve the environment (i.e. the in-game world or game levels), or the evolution of agents (the so-called "bots") behavior in order to produce more challenging opponents to the players.

In the first category we can enlist, e.g., the work of (Frade et al., 2012), which develops a GP-based procedural content technique to generate procedural terrains that do not require parameterization. Taking a different perspective, (Halim & Raif Baig, 2011) propose a set of metrics for measuring entertainment in video games and then use evolutionary algorithms to generate games using the proposed entertainment metrics as the fitness function. (Mourato et al., 2011) propose an approach rooted into GAs to generate automatically levels for a video game. The only work

suggesting the use of GAs in MMORPGs – to the best of our knowledge – is (de Carvalho et al., 2010), which proposes to use GAs for managing the dynamics of geophysics events, asserting that specific events will occur only in areas where they are prone to. The work of (Frade et al., 2010) uses GP to evolve automatically Terrain Programs, which are able to generate terrains procedurally, for a set of desired accessibility parameters. Finally yet importantly (Sorenson & Pasquier, 2010) developed a genetic encoding technique specific to level design used to generate game levels.

In the second category, we enlist works focused on evolving bots' behavior. For example, (Mora et al., 2012) focus on an evolutionary algorithm designed for evolving the decision engine of a program that plays Planet Wars, a game requiring the bot to deal with multiple target, while achieving a certain degree of adaptability in order to defeat different opponents in different scenarios. (Esparcia-Alcázar & Jaroslav, 2012) focus their attention on the problem of estimating the fitness value of individuals in an evolutionary algorithm while containing time and costs. In the field of racing video games, (Onieva et al., 2012) developed a driving system, whose controller for adapting the speed and direction of the vehicle to the track's shape is optimized by means of a GA. The work of (Inführ & Raidl, 2012) showed how GP could be used to create game playing strategies for 2-AntWars, a deterministic turn-based two-player game with local information. (Barros et al., 2011) used GAs to develop a convincing artificial opponent for chess. (Benbassat & Sipper, 2011) apply GP to zero-sum, deterministic, full-knowledge board games. (Alhejali & Lucas, 2010) used GP to evolve a variety of reactive agents for a simulated version of Ms. Pac-Man. The work presented by (Hong & Zhen Liu, 2010) presents a GA based-Evolvable Motivation Model for bots. Both the works of (Mora et al., 2010a) and (Mora et al., 2010b) focus on the adoption of GAs and GP techniques to evolve bots' behavior in Unreal. Last but not least, (Wong & Fang, 2012) study the applications of neural network and GAs techniques for building controllers for automatic players.

2.2 GAs and commercial video games

The use of GAs in commercial video game is still quite limited, at least as far as developers disclose it. The three major titles whose gameplay relates heavily on evolution are Creatures – released in its first version by Millennium Interactive in 1996 (Fig. 1), Spore – released by Maxis in 2008 (Fig. 2), and GAR: Galactic Arms Race – released by Evolutionary Games in 2010 (Fig. 3). None among them uses GAs in a way similar to that envisaged by the present work. In particular, Creatures only seems to use chromosome to generate minor variations (e.g. skin colour) in the different generations of Norns (the fictional creatures populating the game world), while the evolution of the specie is based mainly on learning and reasoning algorithms. Spore exploits GAs to produce automatically animations for the species created by the players, and uses extensively procedural generation for "evolving" content pre-made by developers. Nonethless, the game main focus is on the evolution of a single user-created specie. Galactic Arms Race exploits NEAT – NeuroEvolution of Augmenting Topologies algorithms to develop spaceships' weapons accordingly to the player's play style.

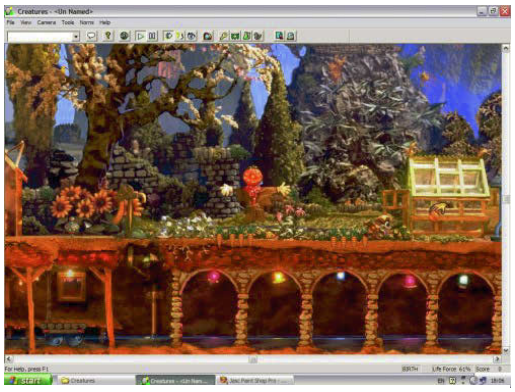

Figure 1- Screenshot from the video game Creatures

Figure 2 - Screenshot from the video game Spore

Figure 3 - Screeshot from the video game GAR

3. Genetic Algorithms

Genetic Algorithms (GAs) are a particular class of algorithms applied to solve many classes of problems, mainly belonging to the Artificial Intelligence (AI) field, but – more in general – they are useful in many optimization problems and in heuristic search processes. They have been inspired by Darwin's evolution theory: the chromosomes of a set of individuals represent a population, and a new generation in the population is produced by recombining, according to specific rules, the genetic material. For each generation, an ad hoc *fitness*

function selects the most "suitable" parents and iterate the reproduction process on them. To produce "children" the algorithm operates genetic recombination techniques (*crossover*) and *mutations* – similarly to what happens in the natural world – on chromosomes represented by bit sequences. Once the new generation has been created, the algorithm verifies whether the population registers any improvement in any relevant feature. If yes, parents will be discarded and their sons will substitute them in the reproduction process. Generally, these steps will be iterated until some optimal solution is reached (Mitchell, 1998; Mode & Sleeman, 2012; Koza, 1998).

3.1 Crossover

GAs use *crossover* to mix the genes of the two parents. Crossover can be achieved by different approaches:

1. *Single point crossover*: the chromosome is split into two parts in a randomly selected point. The chromosome describing the offspring is composed by the first section of the first parent chromosome and by the second part of the second parent chromosome. A second offspring will inherit the remaining two parts;

2. *Two points crossover*: the chromosome is split into three parts, the first and third part of the first parent plus the second part of the second parent become the genetic heritage of the first whelp, while the remaining parts go to the second whelp;

3. *Uniform crossover*: this approach provides a higher genetic variation, since each gene of the whelp is copied – randomly – from one of the corresponding gene belonging to one of the parents. The second whelp will have the genes not chosen for creating its "brother" chromosome;

4. *Arithmetic crossover*: the offspring chromosomes are the results of some arithmetic operations on the parents' genes (e.g. an AND operation).

For our aim is to generate the broadest diversity among monsters, we have opted for the *uniform crossover* approach. Actually, the first two techniques (*single* and *two point(s) crossover*) provide a limited variation in the genetic heritage of the offspring, while the *arithmetic crossover* is not suitable for the structure we have used to represent chromosomes (see §4).

3.2 Mutation

Genetic *mutation* is useful for inserting into the offspring's chromosome some characteristics not inheritable from parents, since not present in their genetic heritage. Similarly to what happens in nature, mutations can introduce a new characteristic or modify/destroy an existing one.

In our work we have consider only the possibility to create/destroy characteristics, since modifying an existing one (e.g. the beak of a bird that increases its length from generation to generation, according to Darwin's theory), besides requiring more complex data structures to be managed, would not have made much sense in a typical MMOs. Moreover, representing such a mutation would create more than one headache to someone in charge of developing graphics 3D models and animations for the monsters.

4. Selecting and describing monsters

To provide a significant population of monsters, we have tried to describe them accordingly to the characteristic and skill structures used in the most popular MMOs and MORPGs (such as World of Warcraft). Since these games can be set in very different periods (ranging from the most remote past, to the far future), also the types of mobs present can vary widely, and may require a different representations of their genes. This implied to make some choices, such as selecting an historical setting and sticking to it.

Since the vast majority of successful video games are based on a *fantasy* setting, we have opted for it. This, inevitably, led to lose some generality in the monsters representation, but on the other side offered the advantage to guarantee coherence and meaningfulness. Moreover, a generative algorithm based on fantasy setting, probably is more likely to be inserted into a video game. Finally yet importantly, such an exploited setting offers a greater variety of ready-to-use monsters.

Figure 4 - Fantasy mobs: an Ent and a Beholder

An accurate description of mobs (e.g. numerical values for their skills) is generally difficult – or quite impossible – to extrapolate from commercial MMOs. Luckily, these games are generally more or less sophisticated derivations of paper-and-pen Role Playing Games (RPGs), such as the renowned Dungeon and Dragons (D&D) (Bartle, 2003). This offered us the opportunity to exploit the huge corpus of information about physical aspect, characteristics and skills present in the manuals of tabletop RPGs games. In particular, the description of the monsters population we have adopted is based on the mobs described in the Dungeons and Dragons (D&D) manuals: (Dungeons and Dragons, 2000, 2003). We have analyzed more than 150 fantasy monsters (see Fig. 4 for an example), and we have created a "candidate" list of monsters that could have been represented and used as input for a GAs. We have not included in the list both immortal (e.g. angels, quasi-divine creatures, etc.) and undead (e.g. zombies, vampires, ghosts, etc.) monsters: the first ones could have caused overpopulation, while the second ones usually do not reproduce (at least in a "natural" way).

Characteristic	Type	Range
Sex	physical	2
Head number	physical	0-7
Arms number	physical	0-8
Legs number	physical	0-8
Eyes number	physical	0-8
Skin colour	physical	0-8
Eyes colour	physical	8
Tail number	physical	0-8
Wings	physical	yes/no
Fins	physical	yes/no
Gills	physical	yes/no
Scales (fish-like)	physical	yes/no
Scales (dragon-like)	physical	yes/no
Feathers	physical	yes/no
Hair/fur	physical	yes/no
Size	physical	8
Type (animal, magical creature, etc.)	nat. ab.	8
Movement: swimming	nat. ab.	yes/no
Movement: flying	nat. ab.	yes/no
Movement: digging	nat. ab.	yes/no
Movement: climbing	nat. ab.	yes/no
Breathing: air	nat. ab.	yes/no
Breathing: water	nat. ab.	yes/no
Breathing: fire	nat. ab.	yes/no
Breath and type (e.g. fire, ice, etc.)	mag. ab.	6
Natural weapons and type (e.g. claws)	mag. ab.	8
Aura and type (e.g. fire, etc.)	mag. ab.	7
Casts spells	mag. ab.	yes/no
Immunity ad type (e.g. to poison)	mag. ab.	6
Lycanthropy	mag. ab.	yes/no
Shapeshifter	mag. ab.	yes/no
Fast healing	mag. ab.	yes/no
Regeneration	mag. ab.	yes/no
Resistance to spells	mag. ab.	yes/no
Resistance to dispel	mag. ab.	yes/no
Damages reduction (e.g. thick skin)	mag. ab.	yes/no
Poisonous	mag. ab.	yes/no

Table 1 - Genes of the main mob's characteristics

Each monster is described by a chromosome, which maps its characteristics and skills. Characteristics can be: *physical* (e.g., number of legs, eyes colour, etc.), *natural abilities* (e.g., breathing water, ability to swim, etc.), and *magical abilities* (e.g., immunity to fire, ability to cast spells, etc.), as detailed in Tab.1. The maximum number of variations for each characteristic has been fixed to 8 (i.e., a monster cannot have 9 or 10 arms), thus constraining the possible mutations, in order to guarantee the algorithm performances.

Monsters' characteristics can be clustered into three main groups, each of which has a different relevance or goal in the reproduction process:

- characteristics that must be different in candidate parents (e.g., *sex*);
- characteristics that must be present in order to create a living monster (e.g., *type of breathing*, *number of legs* and *heads*, etc.);
- optional characteristics, useful to better define the monster (e.g., *resistance to spells*, *lycanthropy*, etc.).

The whole complex of these characteristics can be described using 37 genes, which are the basic building blocks of each monster chromosome. The dimension of each gene depends on how many *disjunctive* variations it can express. When the gene represent a non-disjunctive characteristic (e.g. the mobs breaths both air and water, like in the case of the mermaid), it has been duplicated: this solution allows the offspring to inherit none, one or both the characteristics.

Moreover, 16 more characteristics (*generation*, *current generation*, *challenge rating*, *life*, *strength*, *dexterity*, *constitution*, *intelligence*, *wisdom*, *charisma*, *armour class*, *speed*, *attack*, *reflexes*, *temper*, *will*) are necessary. Each of them is described by a number, since they may assume values too big for being easily managed by binary code: for example, the "*generation*" characteristic (which represent the maximum number of generations to which the monster is allowed to survive – i.e. it is a proxy for its lifespan) may ideally vary between 1 and infinite. As a consequence, any monster used in the GOLEM project is represented by a chromosome composed by 53 genes (37 basic plus 16 special). To notice that not all the characteristics used to describe monsters in D&D have been mapped into the chromosome, since several among them are not relevant for a MMO and/or cannot be properly managed through reproduction (e.g. *horse-riding* ability).

5. Creating and balancing a monsters population

Usually, GAs are used to solve optimization problems. In our work, instead, we have investigated to which extent GAs can be fruitfully exploited to generate a variety of new and unpredictable monsters for MMOs, in order to increase fun for players. To obtain this effect, our algorithm, that we have called GOLEM (Generator Of Life Embedded into MMOs), selects a male and a female candidate parent, mixes up (using *uniform crossover*) their genes, produces some mutations and creates two whelps, each of which possesses a subset of the parents' genome, plus a possible mutation. At this point, the algorithm ends, without applying any optimality criterion.

To implement GOLEM, besides creating a proper structure for representing the chromosome, we had to deal with some other design problems. Namely:

- choosing the number of whelps produced by each couple in each generation;
- defining some parameters to verify whether freshly created species are suitable for survival;
- balancing the population numerosity.

5.1 How many whelps?

Choosing how many whelps each couple of parents should generate in each generation has been a crucial point in the design of GOLEM. We ended up in deciding to generate *two* puppies. Actually, producing only one whelp implied a certain probability that several characteristics would have disappeared (since casually never transmitted to the offspring), thus de facto partially nullifying the effect we wanted to obtain (maximizing *variety*). On the other side, one single puppy would have meant a linear increase in the population numerosity and a better scalability of the software application. At the opposite end, a large number of puppies guarantees that practically every possible characteristic is preserved in the offspring, but also causes a steady and rapid increase in the population, which would reach its upper limit in a bunch of generations. As a result, we would end up with a group of individuals close relatives and presenting quite similar characteristics. For example: a starting population of 30 mobs, producing 8 whelps for couple in two interactions per generation, with an upper bound of 78 individuals for the population, would saturate in 3 generation. As a consequence, only 12 individuals have had the possibility to interact (around the 40% of the starting population).

5.2 Are new species suitable for survival?

Crossover and mutation could produce a mob with characteristics that could diverge significantly from those of its parents. Since the mob should be inserted in a game world, it is necessary to verify its "credibility". In particular, we want to be sure that the newly created specie is able to survive in its *habitat*. Think, for example, to the offspring of a mermaid and a human: it could be born on the mainland, but be suitable only for marine life. In the GOLEM project we have considered four possible habitats: *forests*, *mountains*, *lava flooded areas* and *water*. Whelp's characteristics that must be checked against its habitat to verify its survival probabilities are: *breathing* (e.g. a mob breathing water will not survive in a forest) and *movement type* (e.g. a mob only able to swim, will not survive on a mountain). To notice that several monsters could have more than one respiratory system (e.g. mermaids), thus they suffer some *malus* when outside their primary habitat, but they do not die. Moreover, some other characteristics may provide a *bonus/malus* to the whelp according to the surrounding environment: fins, gills, scales (both fish and dragon-like), feathers and fur.

5.3 How many individuals in the population?

Although some whelp will die due to unsuitability to the environment, it is necessary to keep a control on the population numerosity, in order to avoid overpopulation. This implies defining an upper bound to the number of individuals alive, and some criteria to "kill" several mobs when their number reaches a certain threshold. For this reason, we have introduced both death *by old age* and *by chance*.

5.3.1 Death by old age. In this case, for each generation and mobs, GOLEM verifies whether the mob has reached the end of its lifespan (in terms of number of generations to which it can survive). In particular, each monster has a *longevity* that can assume one value among *short*, *medium* and *long*, to simulate the different lifespans of different monsters.

5.3.2 Death by chance. In a game world, a mob can die also because it has been killed by a player (and generally this is one of the main goal of players!) or, in few cases and under special game design conditions, due to illness. To avoid under populating the world, GOLEM applies casual death only when the population numerosity reaches the 70% of its maximum. This threshold has been defined by trials. To decide whether a mob should die by chance, we have grouped monsters into four categories, according to their "*challenge rating*" (a numerical value representing a proxy of the monster hardiness and stamina, usually adopted in large number of RPGs): stronger mobs get killed with a lower probability. Once the challenge rating level has been (randomly) chosen, an individual in that category is randomly extracted to die.

6. The GOLEM algorithm

The GOLEM algorithm takes as input a starting population (whose numerosity grows constantly, generation after generation), and executes four main functions: *selection*, *crossover*, *mutation* and *evaluation*, after which a new generation is added to the population. All the functions have been implemented using C++, since it is one among the most diffused languages in the video game industry.

6.1 Selection

This function main goal is to select couples of candidate parents. Its first action is verifying whether some mobs should die by old age or by accident. That is to say, firstly GOLEM compares each mob lifespan against the current generation number, and then verifies if the population numerosity has reached the 70% of its maximum value. If yes, death by chance occurs as described in §5.3.2. Once this verification is concluded, the function selects candidate parents: if the "challenge rating" of the two mobs in a couple is too different, they do not mate. This check simulates the fact that, in a fantasy world, powerful mobs (e.g. a dragon) generally are uninterested in mating with weaker creatures (e.g. a kobold). The challenge ratings of our mobs have been grouped in four main classes: *low*, *medium*, *high*, *invincible*.

6.2 Crossover

The function takes as input the chromosomes of the parents selected by the Selection function and operates a *uniform crossover* (§3.1) on them. The chromosomes of the two resulting whelps are randomly filled with their parents' genes.

6.3 Mutation

Once we have the offspring chromosomes, we apply *mutation* in order to increase variety and avoid the possibility of obtaining a population too uniform. GOLEM mutation changes randomly the value of *one* (and only one) randomly chosen characteristic; the mutation happens in the 90% of the cases, in order to add variation in the population. The possibility to generate randomly exactly the starting value for a characteristic is not excluded; hence, obviously, the probability to have an actual mutation increases with the length of the string representing the specific characteristic. Lethal or impairing mutations have not been introduced, since it would not have made any sense in a population of mobs for a video game.

6.4 Evaluation

It is now necessary to evaluate whether whelps can survive the habitat were they are born, as described in §5.2. If yes, the function checks if some *bonus/malus* applies: e.g., a water-born mob devoid of fins and scales can survive in the sea, but is weakened; as a consequence its skills (which are represented by numerical values – see §4) will suffer a 20% *malus*. The evaluation is applied to every whelp created in the current generation.

6.5 Ending condition

Generally GAs end when a certain condition – which determines the "optimum solution" – is met. The GOLEM algorithm is not intended for optimization, nor it has to evaluate the quality of the offspring (our goal is to maximize diversity), hence its ending condition is met when it reaches a predefined number of generations.

6.6 GOLEM fine tuning

In the definition of GOLEM we had to face several criticalities. In particular, we had to fine-tune by hand the value of several parameters in order to obtain useful results (in terms of mobs generated) and good performances. As a matter of fact, the outcomes generated by the algorithm are affected by:

- the number of characteristics represented by the chromosome;
- the probability to have a mutation;
- the probability to have death by chance;
- the probabilities, for a low/medium/high/invincible *challenge rating* mob, to be chosen for death by chance (respectively 0.8, 0.6, 0.3 and 0.1);
- the maximum numerosity of the population;
- the number of generation to produce;
- the number of whelps for each mating.

7. GOLEM primary results

As an example of what can be obtained from GOLEM, let's see what happens when crossing two very different monsters: a griffin and a goblin (see Fig.5). By recombining and mutating their chromosomes, we obtain two whelps, whose genome differs from those of their parents respectively 18.8% and 28.3% (see Tab.2– not binary values in greyed cells).

Figure 5 - A griffin and a Goblin

To estimate the level of diversity among generations, the variation should not be calculated taking as reference the starting couple, since offspring genes, after a certain amount of generations, could – by chance – configure (partially or completely) exactly as those of the ancestors. Consequently, these variations would not be included into the total amount of variations. Hence, the degree of diversity provided by the algorithm should be evaluated taking into consideration the differences between each couple of two subsequent generations. If we adopt this approach, we can notice that, with a starting population composed by 2 individuals, after 10 generations, the average genetic difference among two subsequent generations is of 7.1 genes (that is to say the 13%), while after 20 generations this value increases till the 7.15 (13.4%). This value then decreases in the following generations till it becomes constant, due to an increasing similarity in the genetic heritage (see Fig. 6; note that, since the comparison is made on only 1 puppy, the figure shows only half the generations on *x* axis). After 100 generations, the difference between parent and son is only of 1.9 genes (3.5%).

If the staring population increases to 18 different individuals, the algorithm performs better. As Fig.7 shows, the difference among contiguous generations is of 7.4 genes (13.9%) on average, and, even after 100 generations, the genetic mixing is still substantial.

After a dozen of tests, we have established – by trials and errors – that GOLEM performances are at their best in the following situation:

- population starting numerosity: 18;
- maximum population: 200 individuals;
- number of generation: 100;
- number of whelps per generation: 4 or 8;

in this case, the final numerosity of the population is 170, with a good diversity among mobs.

In the case a larger number of monsters is required, a quite good performance is provided by this configuration:

- population starting numerosity: 18;
- maximum population: 400 individuals;
- number of generation: 150;
- number of whelps per generation: 4 or 8;

in this latter case, the final numerosity of the population is 330, with a quite good diversity among mobs: approximately 30 individuals have a very similar genome (that is to say the difference is no more than 2 genes), thus only the 10% of the population is represented by very similar – or even identical – monsters. A further increase in the population numerosity causes the appearance of too many very similar chromosomes.

Characteristic	Goblin	Griffin	*Whelp 1*	*Whelp 2*
Max. generations	20	30	*30*	*20*
Current generation	1	1	*1*	*1*
Challenge rating	1	4	*1*	*1*
Life (hits)	5	59	*59*	*59*
Strenght	11	18	*11*	*11*
Dexterity	13	15	*15*	*15*
Constitution	12	16	*16*	*16*
Intelligence	10	5	*5*	*10*
Wisdom	9	13	*9*	*13*
Charisma	6	8	*6*	*8*
Armour class	15	17	*15*	*15*
Speed	9	9	*9*	*9*
Attack	1	7	*7*	*1*
Reflexes	1	7	*7*	*7*
Temper	3	8	*3*	*3*
Will	0	5	*5*	*0*
Sex	1	0	*1*	*0*
Breathing: air	1	1	*1*	*1*
Breathing: water	0	0	*0*	*0*
Breathing: fire	0	0	*0*	*0*
Head number	0000	0000	*0000*	*0000*
Arms number	0001	0000	*0001*	*0000*
Legs number	0001	0101	*0001*	*0101*
Eyes number	0001	0001	*0001*	*0001*
Skin colour	100	000	*100*	*000*
Eyes colour	100	101	*101*	*100*
Tail	0000	0001	*0000*	*0001*
Wings	0	1	*1*	*0*
Fins	0	0	*0*	*0*
Gills	0	0	*0*	*0*
Scales (fish-like)	0	0	*0*	*0*
Scales (dragon-like)	0	0	*0*	*0*
Feather	0	1	*1*	*0*
Hair/fur	0	0	*0*	*0*
Spells	0	0	*0*	*0*
Size	100	100	*100*	*100*
Type	111	011	*111*	*011*
Movement: swimming	0	0	*0*	*0*
Movement: flying	0	1	*0*	*1*
Movement: digging	0	0	*0*	*0*
Movement: climbing	1	0	*1*	*0*
Breath and type	000	000	*000*	*000*
Natural weapons	000	001	*000*	*001*
Aura and type	000	000	*000*	*000*
Immunity and type	000	000	*000*	*000*
Lycanthropy	0	0	*0*	*0*
Shapeshifter	0	0	*0*	*0*
Fast healing	0	0	*0*	*0*
Regeneration	0	0	*0*	*0*
Resistance to spells	0	0	*0*	*0*
Resistance to dispel	0	0	*0*	*0*
Damages reduction	0	0	*0*	*0*
Poisonous	0	0	*0*	*0*

Table 2 - Genes of a Goblin, a Griffin and of their offspring

Nonetheless, it is important to underline that a similar chromosome – or even identical – not necessary implies that the mobs are "the same". For example, a population whose maximum is fixed to 250 individuals produces 218 whelps in 100 generation (4 or 8 puppies for each generation): among them only the 20% have similar chromosomes. In fact, genes describing physical traits can described in the same way monsters that have very different skills; e.g. both a panther and a mouse have four legs, fur, a tail, and breathes air, but their strength and hazard is deeply different.

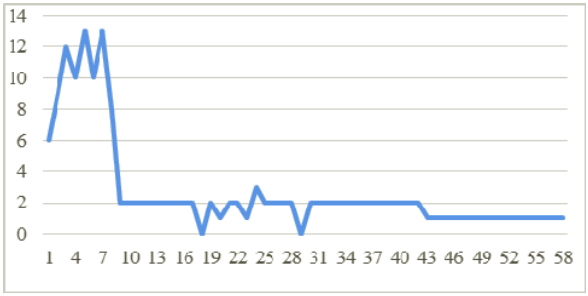

Figure 6 – Genetic difference (starting population: 2)

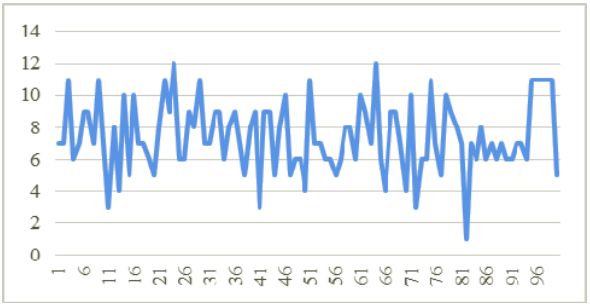

Figure 7 - Genetic difference (starting population: 18)

8. Conclusions and future development

In this work, we have tackled the issue of increasing diversity and unpredictability of mobs by developing an ad-hoc GA, we have called GOLEM – Generator Of Life Embedded into MMOs. In particular, we have provided a detailed general-purpose description of fantasy creatures, based on genes and chromosome, developed functions aimed at generating new species by uniform crossover and mutation on the genome of monsters parents, implementing an overall logical architecture that partially mirrors a biological ecosystem. Only several specific aspects of an actual ecosystem have been modelled: those relevant for mimicking a meaningful virtual environment for a video game. We have then tested and fine-tuned the algorithm in order to obtain outcomes – in terms of monsters population – useful for a MMOs.

The GOLEM project provides only the GA algorithm and the database of the monsters characteristics used to create the chromosomes. To be embedded into an actual MMOs – or,

more in general, in a video game –, it could, and should, be expanded in many directions. In particular, we have envisaged several possible future developments for the GOLEM algorithm:

- adding *dominant* and *recessive* genes: this development needs a careful design and a fine tuning, since it may imply a decrease in the number of characteristics actually expressed (*phenotype*) by the genetic heritage (*genotype*);
- introducing more significant mutations, which, e.g., may increasingly modify in time a specific characteristic;
- adding interdependence among specific genes: e.g. a mobs able to breath water will also have gills;
- refining the death by old age/by chance function, to better simulate monster aging (e.g. older monsters will not reproduce and will have more chances to succumb by illness).

Moreover, several refinements and simulations will also be developed to test GOLEM performances under the following conditions:

- the algorithm is embedded into a client-server architecture, designed to support a MMO, hence real-time interactions;
- the game environment is populated by players, which interact with the mobs (and kill them);
- the number of players varies (hence the population of the monsters should adapt).

Last, but not least, some corollary – but nonetheless of fundamental relevance for using GOLEM in a video game – imply such complex issues to generate several autonomous research areas:

- creating tools able to provide automatically – and in quasi real-time – graphic 3D representations and animations of GOLEM-generated monsters;
- adapting GOLEM as a basis for generating game maps directly related to the monsters characteristics;
- measuring the impact of monsters diversity generated by GOLEM on players' overall satisfaction with the game.

References

Alhejali, A. M., and Lucas, S. M. (2010). Evolving diverse Ms. Pac-Man playing agents using genetic programming. *Computational Intelligence* (UKCI), 2010 UK Workshop on. IEEE, 2010.

Barros, G. A. B., et al. (2011). An Application of Genetic Algorithm to the Game of Checkers. *Games and Digital Entertainment (SBGAMES), 2011 Brazilian Symposium on*. IEEE.

Bartle, R. A. (2003). *Designing Virtual Worlds*. New Riders Publishing, Indianapolis (Indiana).

Benbassat, A., and Sipper, M. (2011). Evolving board-game players with genetic programming. Proceedings of the *13th annual conference companion on Genetic and evolutionary computation*. ACM, 2011.

de Carvalho, L. F. B. Silva, et al. (2010). An application of genetic algorithm based on abstract data type for the problem of generation of scenarios for electronic games. *Intelligent Computing and Intelligent Systems* (ICIS), 2010 IEEE International Conference on. Vol. 2. IEEE, 2010.

Dungeons and Dragons (2003). *Monster Manual: core rulebook III v 3.5*. Wizard of the Coast

Dungeons and Dragons (2000). *Player's handbook: core rulebook I.* Wizard of the Coast

Esparcia-Alcázar, A. I., and Moravec, J. (2012). Fitness approximation for bot evolution in genetic programming. *Soft Computing*: 1-9.

Frade, M., de Vega, F. F., and Cotta, C. (2012). Automatic evolution of programs for procedural generation of terrains for video games. *Soft Computing* 16.11: 1893-1914.

Frade, M., Fernandez de Vega, F., and Cotta, C. (2010). Evolution of artificial terrains for video games based on accessibility. *Applications of Evolutionary Computation*. Springer Berlin Heidelberg. 90-99.

Halim, Z., and Raif Baig, A. (2011). Evolutionary Algorithms towards Generating Entertaining Games. Next Generation *Data Technologies for Collective Computational Intelligence*. Springer Berlin Heidelberg, 383-413.

Hong, Y., and Zhen Liu (2010). A First Study on Genetic Algorithms Based-Evolvable Motivation Model for Virtual Agents. *Multimedia Technology* (ICMT), 2010 International Conference on. IEEE, 2010.

Inführ, J., and Raidl, G. R. (2012). Automatic generation of 2-antwars players with genetic programming. *Computer Aided Systems Theory–EUROCAST 2011*. Springer Berlin Heidelberg. 248-255.

Koster, R. (2004). *A Theory of fun for game design*, Paragliph Press.

Koza, J.R. (1998). *Genetic programming: on the programming of computers by means of natural selection*, MIT Press.

Maggiorini D., Nigro A., Ripamonti L.A., Trubian M. (2012a). Loot Distribution in Massive Online Games: foreseeing impacts on the player base. 8th *International Workshop on Networking Issues in Multimedia Entertainment* (NIME'12) Int. Conf. on Computer Communication Networks ICCCN 2012 Munich, Germany.

Maggiorini D., Nigro A., Ripamonti L.A., Trubian M. (2012b). The Perfect Looting System: looking for a Phoenix? IEEE *Conference on Computational Intelligence and Games* (CIG), Granada (Spain).

Mitchell, M. (1998). *An Introduction to Genetic Algorithms*, MIT Press.

Mode, C.J. and Sleeman, C.K. (2012) *Stochastic Processes in Genetic and Evolution: Computer Experiments in the Quantification of Mutation and Selection*, World Scientific Publishing Company.

Mora, A. M., et al. (2012). Effect of Noisy Fitness in Real-Time Strategy Games Player Behaviour Optimisation Using Evolutionary Algorithms. *Journal of Computer Science and Technology* 27.5: 1007-1023.

Mora, A. M., et al. (2010a). Evolving bot AI in Unreal. *Applications of Evolutionary Computation*. Springer Berlin Heidelberg, 2010. 171-180.

Mora, A. M., et al. (2010b). Evolving the cooperative behaviour in Unreal™ bots. *Computational Intelligence and Games* (CIG), 2010 IEEE Symposium on. IEEE.

Mourato, F., dos Santos, M. P., and Birra, F. (2011). Automatic level generation for platform videogames using genetic algorithms. Proceedings of the 8th *International Conference on Advances in Computer Entertainment Technology*. ACM.

Onieva, E., et al. (2012). An evolutionary tuned driving system for virtual car racing games: The AUTOPIA driver. *International Journal of Intelligent Systems.* 27.3: 217-241.

Sorenson, N., and Pasquier P. (2010). Towards a generic framework for automated video game level creation. *Applications of Evolutionary Computation*. Springer Berlin Heidelberg. 131-140.

Taylor, T.L. (2006) *Play between worlds: exploring online game culture*. MIT Press.

Wong, S., and Fang, S. (2012). A Study on Genetic Algorithm and Neural Network for Mini-Games. *Journal of Information Science and Engineering*, 28.1: 145-159.