DRAGON: diversity regulated adaptive generator online

Laura Anna Ripamonti[1] · Federico Distefano[1] · Marco Trubian[1] · Dario Maggiorini[1] · Davide Gadia[1]

Abstract
Approaches based on Procedural Content Generation (PCG) are more and more diffused among video game developers. They offer many advantages, among which two of the most notables are the opportunity to lighten the burden of level designers and the possibility to produce personalized experiences for the players. In the present work we focus especially on the second aspect, while the first one is addressed as a side effect. In particular, we present DRAGON (Diversity Regulated Adaptive Generator Online), an algorithm for procedurally generating "monster" archetypes for multiplayer games basing also on the players' preferences. The generation process exploits the genetic algorithm paradigm, opportunely adapted, and modified in order to guarantee enough flexibility to the game or level designers. Ideally, the archetypes produced by DRAGON can be employed for any game genre and setting. DRAGON has been implemented as a plugin for one of the state-of-the-art game engines and tested with game developers. Moreover, a simulation has been conducted for the end-users.

Keywords Genetic algorithms (GAs) · Procedural content generation (PCG) · Personalized user-experience · Video games design

1 Introduction

Born in the 50s in US Universities (Tennis for Two by William Higinbotham, dates back to 1958), video games registered a skyrocketing diffusion, boosted by the Internet pervasive diffusion and by the increased performances of PCs hardware and gaming consoles. In time, video games have become very complex artefacts, requiring the skills and competences of highly qualified and diversified members in the development team. Therefore, the importance of video games in the

✉ Laura Anna Ripamonti
ripamonti@di.unimi.it; laura.ripamonti@unimi.it

[1] Department of Computer Science, Università di Milano, Milan, Italy

global economy has grown to the point of competing with cinema, both for development costs and for income produced. Also, the borderline between the technologies underpinning the two entertainment media tends now to blur. More and more the importance of moving the player emotionally is underlined, and for this reason modern video game development teams often include - beside computer scientists and game designers -, artists, musicians, historians, psychologists, writers, actors, etc. In parallel, the development and diffusion of reliable and powerful internet connection has pushed players and developers toward multiplayer games, whose success is testified by the recent introduction of the so-called e-sports. Actually, if we examine the history of games (which dates back several millennia), it is evident that they have always been though for multiple players, the single-player video game being a paradigm forced on people by technological constraints till few decades ago [34]. In this panorama, the expectations of players have risen a lot, as well as their knowledge of video game mechanics and dynamics. These expectations imply hard times for game and level designers, who are called to figure out always new, intriguing ways to entertain the players with their products.

The burden of this work is particularly hard for complex and sophisticated games and for Massive(ly) Multiplayer Online games (MMOs). Actually, among the many different genres in which video games have flourished, one of the most successful is the huge zoo of MMOs: these games are played simultaneously by great numbers of players and – generally –, feature a persistent world (that is to say the game "story" evolves independently from the fact that each single player has or not logged on). Massive(ly) Multiplayer Online Role-Playing Games (MMORPGs) are equally successful and combine MMOs with features typical of Role-Playing Games (RPGs) (World of Warcraft – WoW by Blizzard being one of the most eminent example of this game type). MMORPGs base their attractiveness mainly on the possibility to develop strong social bonds with other players. This category of games is particularly tricky: on one hand, having a lot of players that interact online seems a way to simplify the work of the designers (after all, players entertain one another), but on the other it requires a lot of extra work. This extra work is due mainly to two reasons: the necessity to be "social architects" able to design all the features necessary to foster social interaction among players, and the need to provide plenty of new, fresh and exciting content on an ongoing basis. This second issue is particularly heavy and costly on the medium-long term, when many players are playing a game since a good amount of time and are at risk of getting bored and of giving up (this is especially dangerous for games which base their business model on subscriptions or in-game transactions). To tackle these problems, a certain amount of research (both by practitioners and scholars) has been done, especially focusing on the (semi)automatic generation of contents. Anyway, due to the level of engagement and sophistication reached by contemporary games, a keen attention should be applied to take into the necessary consideration the emotional impact that the generated contents have on the player.

In the vast panorama of massive multiplayer online games, many relevant aspects exist for which some automatized support to the designers and developers would be more than welcomed. In MMORPGs – and in many MMOs – players interact among them and with the game environment using customizable avatars, which skills improve by collecting in-game "experience", that can be gained typically by solving puzzles, by collecting objects and by killing monsters, often with the help of other players. In the majority of games, specific types of "mobs" (as monsters and – more in general – enemies of the player are called) can be met in specific world areas; each time players destroy them, the system automatically re-spawns new instances of the same monster to re-populate the area. This means that players are generally doomed to meet always the same monsters in the same areas. This implies that – after a certain

amount of time – overcoming and killing mobs in combat is no longer a challenging and thrilling activity, but it becomes a task done "on autopilot". Moreover, the mobs' population is "fixed" and does not vary dynamically with the preferences of the players or their playstyle. Such repetitiveness may bore more experienced players. Currently this problem is an open issue, generally poorly addressed, mainly due to the cumbersome work of designing manually possible alternatives for increasing variety in the type of encounters prepared for the players (which number is, at least, forcefully limited due to constraints on development costs and time). Obviously, the risk of boring players can stem also from other features (such as, e.g. poor storytelling, badly designed features, unfairness, etc.), but, in this work, we will focus on the issue of too repetitive encounters with enemies, since, in our opinion, some kind of "technical solution" to help solving this issue could be devised. In the present work we address the problem of supporting the work of game and level designers by devising, designing, and testing a prototypal tool for procedurally generating "adaptive" mobs for video games. This tool should be enough flexible and easy to use to be easily forged by the game designer into a factory of mobs tailored around the specific game she is working on. Also, the generation process must take into consideration to some extent the preferences of the players, and dynamically adapt the new generations of monsters to match them.

The remaining of this work is organized in seven sections. In Section 2 we briefly summarize how Procedural Content Generation (PGC) has been exploited in the field of video game design and development, and we examine a bit more in detail the GOLEM (Generator Of Life Embedded into MMOs) project [18] and the Experience-Driven Procedural Content Generation (EDPCG) approach [69] that we adopted as our design methodology. The following Section 3 is devoted to describing the principles that guided the design of our Diversity Regulated Adaptive Generator Online (DRAGON). In Section 4 we describe the DRAGON algorithm and in Section 5 its implementation. Section 6 relates about the result of several testing sessions we performed to verify the effectiveness of DRAGON, while Section 7 draws some conclusions and gives several hints for future developments.

2 The long-time relation between video games and PCG

Procedural Content Generation aims at producing contents algorithmically instead of manually. It has been widely adopted in Computer Graphics to generate textures, complex models and scenes containing huge quantities of objects and props (see e.g. [7, 13, 46, 49, 50, 54]). Anyway, this approach has raised a lot of interest in many fields, among which the Human-Computer Interaction research area, and, currently, in the neighbouring *game design* field. We could define PCG for video games as "the algorithmic creation of game content with limited or indirect user input" [29, 60, 66]. The first game to exploit PCG has been Rogue (AI Design, 1980) [67], where the dungeons – and all their contents – representing the game world were generated on the fly each time the player entered a new level. Notably, Rogue became the founder of a whole game genre – the so called "roguelike" games –, that shares the characteristic of exploiting PCG to supply always fresh contents to the players. Among the most successful recent roguelikes, we can list, for example: The Binding of Isaac (developed by E. McMillen and F. Himsl in 2011) and FTL: Faster Than Light (Subset Games, 2012). Anyway, all those games exploit PCG per se, without taking into consideration the player experience during the game. As a result, the levels may also be perceived as frustrating (that was particularly true for Rogue) or too easy by the player.

As already mentioned, development time and costs of video games are constantly rising due to the complexity of contemporary titles (that more and more include high definition graphics, complex storytelling and accurate historical reconstructions, like e.g. in the case of Ubisoft's Assassin's Creed series). Thus, the exploitation of tools able to (semi)automatically produce contents is highly welcomed and can even become a critical asset for a specific game, as in the case of No Man's Sky (Hello Games, 2016), an exploration-survival game whose contents – countless intergalactic planets and their biomes – are completely generated by an appropriate algorithm. Moreover, PCG can be exploited, as in the previously mentioned case of roguelike games, to provide non-repetitive experiences to the player by presenting slightly different contents or gameplay each time she plays the same game (such as in the case of Left for Dead by Valve). Last but not least, PCG can be used, in quite simple games, to dynamically adapt level generation to the skill of each player (for example, this approach has been employed in the platform game Cloudberry Kingdom by Pwnee Studios, published in 2013).

PCG can be applied according to two different modes: *offline* and *online*. In the first case, contents are generated *before* the game is shipped (hence the game contents are "fixed"), while in the latter the generation takes place *while* the game is actually played. In both cases, PCG can be applied to the production of widely different contents, ranging from simple textures, models or sounds to very complex artefacts like entire levels, maps or missions (e.g. X-Com: UFO Defense - Mythos Games and MicroProse, 1994; Left4Dead - Valve Corporation, 2008; Minecraft - Mojang, 2011). In any case, PCG generally uses algorithms and/or heuristics that are specifically engineered for the game under development [51], thus limiting the portability of the software tools developed. Anyway, creating a general-purpose procedural generator for game is an impossible task since the structure and the elements of the game vary broadly with the genre and the specific settings [51]. Interestingly enough, several commercial video games have built their PCG based heavily on evolutive techniques, namely on the Genetic Algorithms (GAs) paradigm. Galactic Arms Race (GAR) [21] is a shooter game set in space that exploits online PCG using a personalized version of NeuroEvolution of Augmenting Topologies (NEAT – see e.g. [63]) to evolve spaceship's weapons according to the player's playstyle. Creatures (by Steve Grand, 1996) is a game that simulates the life of small creatures able to learn how to survive and to reproduce. In the same vein, Spore (Maxis, 2008) combines artificial life with elements of strategy and action games: the player acts as the creator of new life-forms, and how she interacts with the new specie has an impact on its future evolution.

From an academic perspective, a good amount of research has focused on PCG for video games (see, e.g. [11, 37, 57, 60]). Earlier studies investigated mainly the impact of non-player characters (NPCs) on the gameplay [2, 3, 40, 62, 70], the dynamic game balancing [35], and the adaptive PCG to produce personalized contents (see e.g. [22–24, 26, 64, 65]). A quite limited – both in numerosity and in scope – number of works have tackled the generation of coherent game segments – i.e., game levels (see e.g. [9, 36, 38, 52, 56, 58, 59]). On the contrary, scholarly literature on the applications of Genetic Algorithms and Genetic Programming (GP) in video games is quite abundant: the major part of it has tackled either the use of GAs to generate or evolve the environment (i.e., the in-game world or the game levels), or the evolution of agents' behavior, with the scope to produce more challenging enemies for the players. In the first category we can include, e.g., the works of Frade et al. [15, 16] and de Carvalho et al. [55], which focused on the procedural generation of terrains and on the dynamics of geophysics events; that of Halim and Raif Baig [20], which proposed a set of

metrics for measuring entertainment to be used as inputs for evolutionary algorithms, and the studies by Mourato et al. [45] and Sorenson & Pasquier [61] that exploited GAs to create levels for a video game. In the second category, we enlist works focused on evolving bots' behavior. For example, Mora et al. [44] and Esparcia-Alcázar and Jaroslav [14] both addressing the topic of evolving agents able to survive in hostile environments populated by enemies. In the same vein, Guarneri et al. [18] exploited GAs to support the automatic creation of monsters' archetypes for massive online games, while Norton et al. [47] expanded and modified the approach proposed in [18] to support some game dynamics in strategic video games for mobile devices. Onieva et al. [48] tested a driving system for racing games. Several other works tackled the use of GAs to produce strategies for strategic games, such as the work by Inführ and Raidl [30] which focused on 2-AntWars, a deterministic turn-based two-players game with local information, and that by Barros et al. [4] which used GAs to develop an artificial player for chess. Other interesting studies are, for example, those by Benbassat and Sipper [6] which tackled the development of strategies for zero-sum, deterministic, full-knowledge board games, and by Alhejali and Lucas [1] which evolved reactive agents for Ms. Pac-Man. A slightly different approach is that by Hong and Zhen Liu [27]: they developed a GA-based Evolvable Motivation Model for bots. In the same vein, Mora et al. [42, 43] adopted GAs and GP techniques to evolve bots' behavior. Last but not least, Wong and Fang [68] studied the applications of neural networks and GAs for creating controllers for artificial players.

2.1 EDPCG: Experience-driven procedural content generation

In 2011, Yannakakis and Togelius published an intriguing work [69], in which they presented the Experience-Driven Procedural Content Generation (EDPCG) paradigm. Basically, it is an approach to the design that can be useful in many fields, but it seems to suit particularly well the design of video games, since it focuses on the "optimization" of the user experience. They have the remarkable merit to underline that, to produce the same satisfactory gameplay experience for each possible player, it is necessary to include game contents that may vary from player to player, basing on her playstyle, but also on her emotional state. Anyway, as already stated, the manual creation of game contents is a huge time-consumer, hence their contribution traces a path toward the production of an optimal experience while saving development time. According to their idea, the game should be able to "model" the player experience basing on her interaction pattern with some appropriately selected and monitored gameplay elements. This knowledge should enable the system to foresee the player's preferences and/or emotions during gameplay. This "guess" is obtained through an appropriate function, basing upon which dynamic "fitness values" are given to the different gameplay elements. This means that, for each player, it should be possible to define the suitability of each game content and consequently to use this information to deliver personalized experience through some PCG/GA approach. Also, in their vision, the fitness value of each relevant element in the game should not be "fixed", but it should dynamically adapt according to what the player is doing in the game. Even if this approach is promising, quite few field tests have been done till now, especially when massive games are involved. Moreover, in our opinion and to the best of our knowledge, EDPCG cannot be applied alone: it should be coupled with some principles of "good" game design, other way the risk is to generate content that will adapt dynamically, but that not necessarily is fun and entertaining for the players.

Since, as we stated, MMOs/MMORPGs are prone to a quite high degree of repetitiveness, in this work we adopt the EDPCG approach as a guideline to design an algorithm to produce

archetypes of monsters aimed at increasing variety and unpredictability of the enemies encountered by the players in that game genres, while, at the same time, adapting to some extent the content produced to the players' preferences.

2.2 GOLEM – Generator of life embedded into MMOs

Probably the nearest work to our point of view is that published in 2013 by Guarneri et al. [18] and applied by [47]. It described a promising first approach to the automatic generation of monsters for massive games. To guarantee diversity and unpredictability, the authors exploited GAs to partially emulate what would happen in a real environment populated by fantasy beasts and magic living beings. The simulation was not intended to reproduce exactly what happens in a real biome, but it mimicked only those aspects that the authors judged relevant to produce potentially interesting effects in the gameplay. For example, no optimization is performed, since "perfect" (i.e., more suitable for survival) monsters would be too difficult (or impossible) to beat, hence they would disrupt the fun for players. Also, no fitness function is used, consequently, Generator Of Life Embedded into MMOs – GOLEM is an *offline* PCG algorithm only loosely based on GAs. Its goal is to generate new monsters from a starting population defined by the game/level designer; the produced archetypes are then used to populate the game world. The starting population is composed by several beings selected among those listed in the Dungeon & Dragons' Monster Manual [19]. Dungeons & Dragons (D&D) is probably the most famous paper-and-pen role playing game, and the founder of the genre. To represent D&D monsters, GOLEM exploits chromosomes that use both binary and integer coding. Each chromosome identifies one monster, and it has 53 genes, 37 of which represents the "basic" characteristics (such as, e.g., the number of heads or the ability to fly) and the remaining 16 codify "special" features, such as the ability to cast spells. The algorithm performs a predefined number of cycles before ending. During each cycle, the following steps are performed:

- *Selections of parents*: GOLEM selects randomly two monsters of opposite sexes for reproduction.
- *Recombination*: 2 offspring (chromosomes) are generated by applying uniform crossover on the parents' chromosomes.
- *Mutation*: newly generated chromosomes have a non-null probability p to mutate. If the mutation takes place, one randomly picked basic characteristic mutates.
- *Evaluation*: in this step, the "whelps" genome may vary again, basing on the compatibility between the specific monster and the biome in which it is born (i.e., the biome to which it has been randomly assigned by GOLEM). For example, a cross between a siren and an orc may be born in the sea, but without having inherited the capability to survive in a water-based environment. Hence it is doomed to die or to suffer some malus on several of its special characteristics (the value of the affected characteristics/special features in the chromosome is modified). If the new monster survives, it is added to the current population.
- *Environmental selection*: in this last step all monsters too "old" (i.e., which have reached a predefined threshold of generations) are discarded. Then the total number of monsters is evaluated against the maximum dimension of the population: if monsters exceed the maximum possible number, several monsters are discarded according to a probability that increases inversely to their "dangerousness" for the player.

To note that, as already pointed out, GOLEM *does not employ any fitness function* (thus diverging from GAs). The design and implementation choices of GOLEM have some inherent implications and drawbacks that undermine its efficacy. For example, it focuses on producing new monsters without taking into any consideration the players' experience and playstyle. This problem is worsened by the fact that it is not possible to define specific difficulty levels for mobs, aimed at mirroring the skills of the players (Section 4.9). Moreover, GOLEM is strongly linked to D&D, hence scarcely flexible. Similarly, the generation parameters are mainly hardcoded in the application, hence the game designer has practically no possibility to personalize and fine tune them. Last but not least, GOLEM is an *offline* generator, consequently monsters do not "evolve" with the game (e.g., there is no "natural selection" deriving from the interactions with the players), but they are simply generated and then inserted into the game. In the same vein, no "adaptative" evolution can affect the monsters, thus the mutation rate cannot change if needed. No dependent traits can be present in the chromosome (for example, it is not possible to force the presence of fins or gills in an aquatic creature). Newly generated and existing mobs can disappear only because randomly selected in the last algorithm steps, regardless to the fact that they could be appreciated by the players. It is practically impossible for the game designer to keep track of the genealogical tree of a monster. Then, the fact that no fitness function is used leads to the inevitable convergence of the population toward very similar mobs: this is probably one of the biggest limits of GOLEM. Last but not least, from the implementation point of view, GOLEM is a stand-alone application, developed in C++, hence it cannot dialogue directly with any game engine, and requires a good knowledge of C++ to be modified and maintained.

As a starting point of our work we decided to couple the EDPCG paradigm to GOLEM basic principles, and to build on this concept to design, develop and test a general-purpose (i.e., not linked to a certain game) tool aimed at procedurally generating mobs instances for MMOs that overcomes the intrinsic limits of GOLEM.

3 The design of DRAGON

To overcome the limits of GOLEM, we wanted, as suggested by the EDPCG paradigm, to put the player experience at the centre of the design process of our tool – that we called DRAGON: Diversity Regulated Adaptive Generator ONline. To reach this goal, we have selected several well-established principles of "good" game design as our guidelines in the process of devising the requirements of DRAGON. In particular, we have tried to use as our landmarks the concept of *flow*, the fundamental relation that links *play and learning* and the relevance of *dramatic elements* to engage the player emotionally in a game.

3.1 The concept of "flow"

Mihaly Csikszentmihalyi is the psychologist that firstly introduced the concept of flow [10]. The flow is a peculiar mental state that a person may enter while she is performing specific activities that lead her to the "optimal experience". Several preconditions must be met to enter the flow; those – among the list created by [10] – relevant in the field of video games can be summarized as follows [34]:

- the effort put into performing the task must be voluntary.
- the task requires the use of some specific skills.

– the task should be difficult to perform, but nonetheless it must be doable.
– the task goal must be clear and perfectly understood, as well as feedbacks about player's actions.
– the outcome of the task must be unknown, and it must be directly linked to the player's actions.
– by performing the task, the player should discover or improve her abilities and skills.
– failure should not be punished too harshly.

When these conditions are met, the attention of the player is completely absorbed by the task, and she feels a sense of accomplishment and of being detached from what happens outside the activity at hand [34]. From a game design perspective, applying the concept of flow means, among other things, that *the game difficulty should evolve with the skill of the player*, and should eventually adapt to it [34]. When this situation is met, the player lives in the "optimal experience", where her abilities and the game difficulty perfectly balance.

We have tried to give to DRAGON several features that should enable a game designer to introduce monsters that will help to keep the player in the flow. Monsters are generated gradually, thus allowing a step by step introduction in game of always new enemies. The presence of new mobs requires a learning effort to the player, that is called to discover the best strategy to overcome them. A gradual introduction of new monsters has also the side effect of increasing the unpredictability of the combat outcome. Moreover, the algorithm guarantees that enough monsters are present for each difficulty level (so to avoid too skilled players or newbies to be bored or frustrated).

3.2 The relation between play and learning

Raph Koster [34], basing on the work of neuroscientists and psychologists [8, 31–33, 39], describes the learning process that lays at the basis of fun in video games. In a nutshell, we could say that the human brain is a devourer of patterns, that it uses to store away knowledge and recall it when needed: during the process, it releases endorphins, which have the function to stimulate humans to increase their knowledge. Games are designed to teach to the player to find the correct pattern to reach a certain goal, for this reason a well-designed game is so fun to play. Also, in a certain sense, we could say that games are iconic representation of human experience, with which we can practice and improve our skills in a safe environment (think, for example, to middle age tournaments). These two aspects imply, by the way, that once the player has extracted all the underlying patterns, the game is no longer fun to play, on the contrary: it becomes boring. Therefore, the more predictable the game is, the easier it is to find the resolutive pattern, the quicker it becomes boring (see, e.g., Tic-tac-toe).

We have tried to embed this aspect in DRAGON, by forcing the algorithm to generate – at a predefined frequency –, new monsters to add to a hypothetical game, while several among those already present in-game are eliminated by "natural selection". Each new monster is a descendant (to a various degree) of one already existing and already met by the players. This means that the player is stimulated to discover the new pattern necessary to overcome this new enemy, and – at the same time – is confronted with a being that is not completely new. This second aspect should also help in enforcing the coherence of the game environment, thus avoiding disrupting the illusion of being in an alternative world. As a matter of fact, DRAGON is an online PCG algorithm that produces mobs that evolve gradually, together with the player

skills. Last but not least, the current population of mobs is not erased and substituted by a new one, but monsters disappear progressively by dying of "old age".

3.3 The importance of dramatic elements in games

Fullerton [17], Schell [53] and Johan Huzinga [28], all underline the importance of maintaining the player "immersed" into the game world, a fantastic alternative reality enclosed in a kind of "magic circle". To create and preserve this illusion, designers exploit several dramatic elements (i.e., challenge, play, premise, story, and characters [17]). These elements "hide" the formal structure of the game (i.e., "the maths") with a *metaphor*, that moves the player emotionally. They should harmonize with formal elements, and they should foster the conflict the player is confronted with. A critical factor from this perspective is the coherence that each game element should maintain with the whole picture and the story driving the game.

DRAGON tries to address this issue by generating monsters that are coherent with the environment in which the player is expected to meet them. This goal is obtained by the introduction of a phase – in the procedural generation – in which dependent traits are evolved and adapted gradually to the monster's environment, and by the fact that players can observe the different types of monster evolving.

4 Structure of DRAGON algorithm

Basing on the design guidelines and principles described above, we have created DRAGON, an *online* EDPCG algorithm, and its related software tool, that has been implemented as a plugin for Unity 3D, one of the most diffused "free" game engines. This choice stems mainly from the consideration that a tool like DRAGON could be potentially useful for small, indie studios, which – in the majority of cases – do not have the resources to develop proprietary solutions.

As already stated, the outcomes of DRAGON are archetypes of mobs that can be instantiated in any hypothetical MMO/MMORPG game, and that adapt to the players behaviour according to the principle of flow, while at the same time trying to respect the relation between learning and having fun and trying to stick to the metaphor of the specific game. Since DRAGON is inspired by GOLEM, we have maintained the use of GAs to generate monsters, that are represented by chromosomes. Anyway, we have adhered more strictly to the GAs paradigm, and – as required also by EDPCG –, we have introduced a *fitness function* to evaluate whether a certain mob archetype should be maintained or eliminated from the current population. Obviously, in our case too, no optimization in the "classical" sense is performed: the fitness function is used to adapt dynamically the monster population to the behaviour of the players, not to obtain some kind of unbeatable "super-monster".

DRAGON functioning is based on two subsequent macro-phases: in the first one the designer personalizes several parameters in order to adapt the generation process to the peculiarities of the specific game, while in the latter one the generation of mob archetypes takes place. During the first – initialization – phase, the designer should: define the structure of the chromosomes of the mobs (see Section 4.1 and Table 1), select the starting monsters pool (see Section 4.1 and Table 2), select the difficulty levels for monsters and define the function to calculate the difficulty level of the mobs generated (generally these two information are "copied" from the project documentation of the game), select the biomes (see Section 4.8 and Table 4), select genetic dependencies (e.g., a mob able to breath water is expected to have

Table 1 Example of chromosome with 9 genes (20 elements binary array)

Gene structure	Starting bit	Length (bits)	Possible values
Life points (LPs)	0	6	[1, 63]
Attack (AT)	6	4	[1, 15]
Armour class (AC)	10	4	[1, 15]
Wings (W)	14	1	[0,1]
Claws (C)	15	1	[0,1]
Fins (F)	16	1	[0,1]
Flight (Fl)	17	1	[0,1]
Dig (Di)	18	1	[0,1]
Swim (Sw)	19	1	[0,1]

gills, see Section 4.7) and set several other parameters that will be introduced in the following. Once this phase ends, the algorithm automatically calculates difficulty levels and biomes for the starting pool of monsters.

When the setup phase has ended, DRAGON can start to evolve new mobs. The call to the genetic algorithm can be done in different ways (this is one of the parameters set by the designer): it can be a synchronous call or an asynchronous one. In the first case the algorithm is called at fixed time intervals, while in the latter one it is called basing on specific in-game events (e.g. when a certain number of mobs has been killed). In both cases no ending condition is defined.

For each cycle, the algorithm (see Fig. 1):

- evaluates the fitness of the monsters in the current population
- eliminates monsters no longer suitable to survive (too "old")
- selects a couple of parents
- performs a crossover operation and generates two new mobs
- performs (adaptive) mutation on offspring
- develops dependent traits on offspring
- choses native biome for offspring
- calculates the difficulty level of the offspring
- inserts the offspring in the new current population

4.1 Phase 1 – Setup: Chromosome definition and population management

The chromosome of each mob was initially codified as an array of bits (a binary vector): we had excluded more exotic codifies to simplify crossover and mutation operations, but, after the testing phase with users, we have added the possibility to use also integers (see Section 6.2).

Table 2 Description of several monsters according to the chromosome structure in Table 1

Monster	LPs	AT	AC	W	C	F	Fl	Di	Sw
Siren	8	2	1	0	0	1	0	0	1
Skeleton	12	4	3	0	1	0	0	0	0
Harpy	16	6	2	1	1	0	1	0	0
Kraken	50	14	6	0	0	0	0	0	1

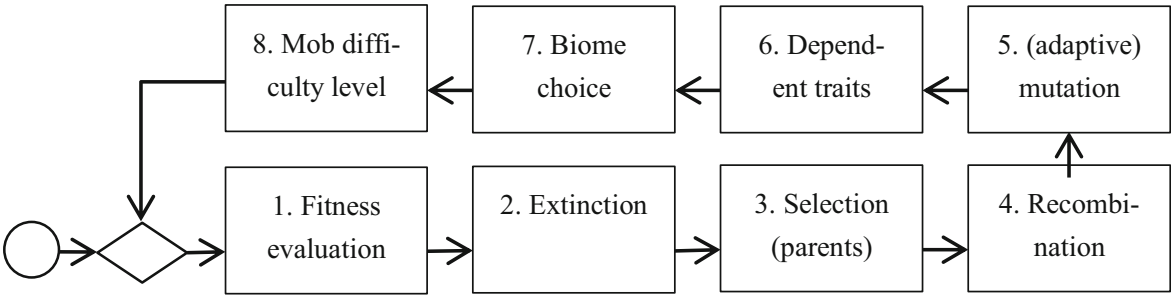

Fig. 1 DRAGON algorithm cycle (phase 2)

Both the array length and characteristics represented by each gene in the chromosome are defined by the game designer during the setup phase, this should guarantee enough flexibility and consequently the possibility to re-use the same tool for many different games and settings. Also, the possibility to define a different type of chromosome for each game, if appropriately used, could help the game designer in designing a coherent game world, thus contributing in the creation of beings that are convincing also from the point of view of the dramatic elements of the game. In the following tables we show an example of how monster archetypes could be defined. Table 1 shows the structure of a chromosome with 9 genes – 6 for the physical aspect and 3 for the abilities of each monster. In Table 2 we list the chromosome of several monsters' archetype described according to the definition of genes shown in Table 1. Table 3 reports the same chromosomes in their binary form.

The numerosity of the *starting population*, too, is defined by the game designer during the setup phase. We decided to let the designer be free of defining the number and type of mobs in the starting population to give her the maximum freedom in setting up a convincing and coherent environment since the very beginning. The numerosity of the population, however, is not fixed, it can vary in time, due to the intrinsic characteristics of the algorithm; this is due to the fact that fitness evaluation and crossover take place in different moments and according to different criteria, thus producing some small fluctuations in the mobs number. Moreover, during the extinction phase (step 2 in Fig. 1), a threshold fixing the minimum and maximum number of monsters in the current population will keep the mobs' number balanced (see Section 4.3), thus producing further fluctuations.

For each "mating" only two whelps are produced, thus helping the players to adapt to the new enemies gradually and to keep track of the relationships among monsters. It is also important to recall that, by applying a uniform crossover, the complete genetic heritage of both parents is passed on (except for some random mutations) to the whelps.

4.2 Phase 2 – Step 1: Fitness evaluation

During the fitness evaluation (step 1 in Fig. 1), every monster in the current population is evaluated. The fitness value of each monster is used to calculate the possibility to be chosen for

Table 3 Binary representation of the monsters listed in Table 2

Monster	Gene
Siren	00100000100001001001
Skeleton	00110001000011010000
Harpy	01000001100010110100
Kraken	11001011100110000001

reproduction. The fitness function we have defined diverges from the classical definition of fitness function "à la Holland" [25], since it includes and balances the effect of two parameters: *diversity* and *death rate*.

The *diversity* value measures how much "different" from the "average" of the current population a mob is. Actually, monsters with a high diversity value should have a higher probability to reproduce, thus increasing the diversity in the population. This, incidentally, may also mean introducing new combat patterns for the player to learn, hence helping to keep her in the flow.

To calculate the *diversity* value of a mob *D(c)*, we use the following formula:

$$D(c) = \frac{\sum_{\forall x \in P \setminus \{c\}} \sum_{i=1}^{L} \frac{d(c_i, x_i)}{L}}{|P|-1} \tag{1}$$

P set of all the chromosome in the current population
L length (bits) of the chromosome
c current chromosome
x all the remaining chromosomes in the current population
x_i i-th bit in the x chromosome
d binary function that compares two bits and produces 0 if they are equal, 1 if not

To note that diversity varies in [0;1], where 0 means that the mob is identical to all the other monsters in the current population, and 1 that it has nothing in common with all the others.

In a similar vein, we have defined a way to measure the monsters' *death rate*. The purpose of this criteria is to introduce an "experience-driven" component in the procedural generation (as we said, DRAGON is an EDPCG tool). The tool continuously collects information about the players encounters outcomes, with the purpose of keeping track of which mob types they preferably engage in combat. This data will help to guarantee that the next – procedurally generated – mob that the players will meet is a descendant of some monster they had already compared with. This will ideally help to maintain the gaming world coherent and help the player to re-use in different ways the knowledge and skills she has already developed in the previous encounters with the current mob's ancestors. To keep track of this parameter, for each mob a counter is increased each time the mob is killed by the players. The most killed mobs are those that the players know better, hence they will have more probabilities to be selected for reproduction. To calculate the death rate of a monster, we use the following function:

$$M(c) = \frac{m(c)}{\sum_{\forall x \in P} m(x)} \tag{2}$$

P set of all the chromosome in the current population.
c current chromosome.
x all the remaining chromosomes in the current population.
$m(x)$ number of deaths of the x chromosome.

We can now define the *fitness function* as a linear combination of diversity and death rate:

$$F(c) = max\{\alpha D(c) + (1-\alpha) M(c), \beta\} \tag{3}$$

D diversity
M death rate

c current chromosome
α varies in [0,1] and its value can be set by the designer to define the relative relevance of diversity and mortality
β varies in [0,1] and its value can be set by the designer to define the minimum fitness value for each chromosome, thus its probability to be selected for reproduction

Including in the fitness function both diversity and death rate values should help to maintain enough difference among the monsters in the population and, at the same time, to avoid puzzling players by introducing mobs that are totally unrelated to those they are already used to.

4.3 Phase 2 – Step 2: Extinction

The goal of this phase is to maintain quite constant the numerosity of the population by eliminating several mobs' archetype. The types of monster to be removed are selected based on their age. This choice has been made to avoid boring the player by forcing them to always combat the same types of monster. We keep track of the "age" (number of generations – or DRAGON cycles – for which the monster has survived) of each mob type. If the age is major than the value of the parameter *E* (that is set by the designer during the setup phase) that mob type has more probabilities to be selected for elimination.

Anyway, since players in a game play at different skill and experience levels, it is also important to make sure that enough monster types are present in each "difficulty level". To obtain this effect, we offer to the game designer the possibility to define her own function to let DRAGON calculate the difficulty level of each generated monster on the basis of its chromosome. Moreover, the game designer can define any number (*NF*) of difficulty levels she desires and their related minimum (*SMIN*) and maximum (*SMAX*) number of monsters. DRAGON will use these inputs to guarantee that the required number of mob types are present for each difficulty level.

To sum up, the selection based on the age works as follows: for each monster type in the current population it is verified whether it is eligible for elimination. A mob type could be removed if and only if: its age is > *E*, it has reproduced at least once, the difficulty level to which it belongs contains at least *SMIN* mob types. Then, for each difficulty level it is verified whether *SMAX* has been exceeded. If yes, several types of that level must be removed (the older types, possibly that have already reproduced at least once). Therefore, *SMAX* can be exceeded at most by 2 units (the last two mobs that have been generated in the current cycle). The same holds true for the total number of monster types present in game, that cannot exceed *SMAX***NF* + *2*.

4.4 Phase 2 – Step 3: Selection of parents

To make sure that also the monster types with a poor fitness score have some small possibility to be chosen for reproduction, we use the "roulette" method [41] for selecting parents. Hence the probability to be chosen as parent is:

$$p(c) = \frac{F(c)}{\sum_{\forall x \in P} F(x)} \tag{4}$$

P set of all the chromosome in the current population
c current chromosome

x all the remaining chromosomes in the current population
F fitness function

To note that, after the first parent has been selected, its chromosome is removed from the mating pool to avoid that a certain mob type reproduces itself (but for some incidental mutation).

4.5 Phase 2 – Step 4: Recombination

As already pointed out, we use the *uniform crossover* method [41] to recombine the genes of the two parents' chromosome. In particular, the second whelp inherits the traits not included in the gene set of its "sibling". Thus, we try to make sure that the whole genetic heritage is passed on to the next generation so to increase the diversity of the future mob types and not to exclude accidentally some desired feature. Actually, making some random feature "disappear" may undermine the believability of the game world, thus negatively affecting the immersion of the player.

4.6 Phase 2 – Step 5: Mutation and adaptive mutation

The chromosome of newly created monster types can undergo two types of mutation (to increase diversity): an *ordinary* mutation and an *adaptive* one, if needed.

In the first case, each bit in the chromosome "mutates" (e.g. inverts its value in the binary case) with probability T (defined by the designer in the setup phase). The latter case (adaptive mutation) takes place when the population tends to converge toward too similar individuals. This situation is verified when the *degree of diversity G(P)* is equal or under a diversity threshold value $S \in [0,1]$, set by the game designer during the setup phase. *G(P)*, in fact, measures how much different the mobs in the population are, while S represents how much (in percentage) the individuals belonging to the population of mobs *must* be different from each other: for example, in an alien setting a wide range of very different "animals" may be needed, while in fish tank the animals must all share some similar features. Hence S value must be chosen by the designer according to the needs of the specific game, and in the setup phase it is entered in the algorithm by using the appropriate slider in the setup interface (see Fig. 2). *G(P)* is then calculated as follows:

$$G(P) = \frac{\sum_{\forall x \in P} D(x)}{|P|} \tag{5}$$

P set of all the chromosome in the current population
x chromosome of the mob x belonging to P
D diversity

In this case, the probability to have a mutation in one or more genes increases: instead of T – which is used for the ordinary mutation –, each gene has a probability U (inversely proportional to *G(P)*) to undergo an adaptive mutation, thus helping to obtain more diversified mobs. U is calculated as follows:

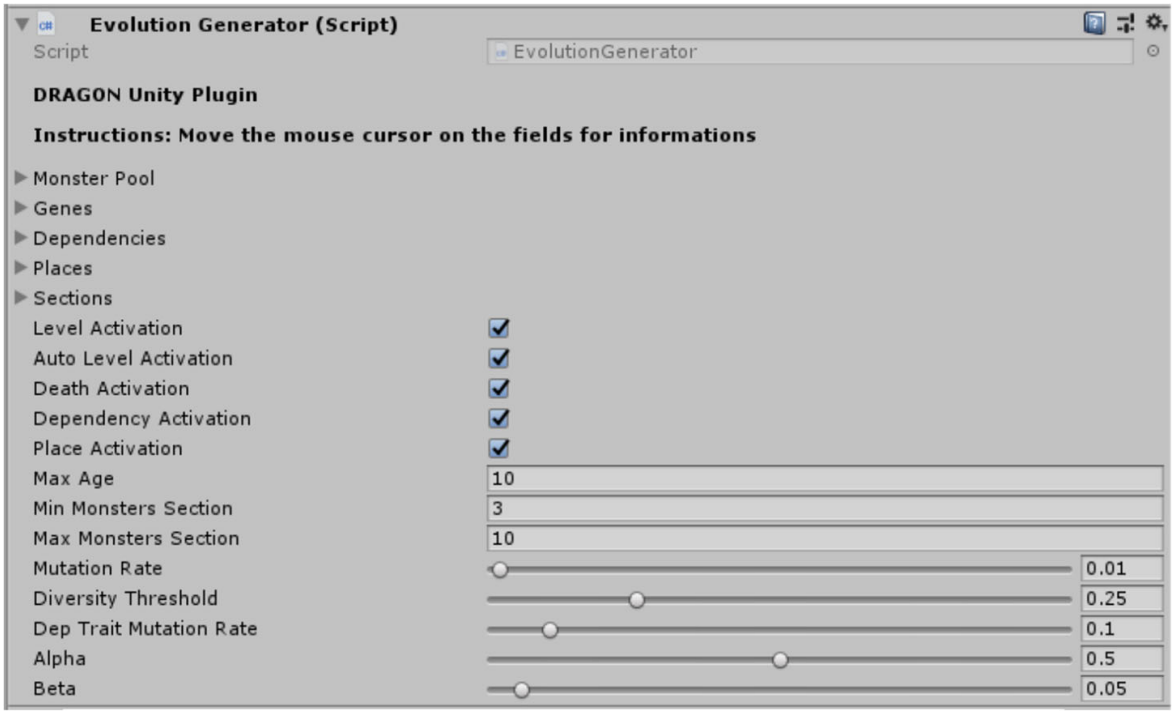

Fig. 2 Example of the monsters' creation interface

$$U = \begin{cases} 0.5 \;\; if \;\; G(P) = 0 \; or \; \dfrac{T}{2G(P)} > 0.5 \\ \dfrac{T}{2G(P)} \quad in \; all \; the \; other \; cases \end{cases} \tag{6}$$

The probability U to have a mutation is 50% when the monsters are all the same ($G(P) = 0$). In a similar vein, the probability is trimmed down to 50% when the value obtained by calculating the ratio among T and $2G(P)$ would exceed 50% (this would produce too many mutations in each chromosome). In all the remaining cases U assumes the value produced by the ratio. For example, if $T = 0.01$, $S = 0.25$ and the value of $G(P)$ is 0.25 an adaptive mutation is triggered ($G(P)$ is equal to S). Consequently $U = 0.01/2*0.25 = 0.01*0.5 = 0.02$, and the probability to have a mutation doubles up.

4.7 Phase 2 – Step 6: Development of dependent traits

After the mutation phase, the chromosomes could be further modified to guarantee the respect of dependencies among genes subgroups. For example, a monster that has wings may also not have the ability to fly (e.g. an ostrich vs a dragon). During the setup phase, the game designer can define dependencies among genes, also specifying a manifestation parameter $M \in [0,1]$, that indicates the probability that the dependent gene(s) will manifest its effect provided that the gene(s) from which it depends is present. In particular, this probability applies if and only if the dependent gene is disabled (all its bits are 0) and at least one of the genes from which it depends is enabled (at least one bit is 1). The presence of interdependencies among the genes helps to create convincing monsters (e.g., a water-based creature like a siren should also be able to swim and breath water, other ways it would be perceived by the players like an oddity) and thus to keep the player immersed in the game world.

4.8 Phase 2 – Step 7: Biome choice

To preserve the coherence (and believability) of the game world, in this step DRAGON automatically determines the biome to which the new mob belongs, hence the appropriate in-game areas where to spawn its instances. During the setup phase, the game designer defines several biomes, assigning them priorities, and – for each of them – which are the necessary and optional traits that a monster must/should have to survive (Table 4). At least one biome must be set as default, with priority 0. DRAGON assigns a biome to a monster on the basis of its characteristics and of the priorities set by the designer. For example (see Table 4), a monster that resists fire, is not a parasite nor able to breath water, but has scales and "breaths lava" would be suitable for living in plains, deserts and volcanoes, but since volcano has the highest priority, this will be the chosen biome.

4.9 Phase 2 – Step 8: Monster difficulty level

In its last step, DRAGON assigns to each mob a difficulty level, calculated on the basis of an appropriate function, defined by the game designer in the setup phase. The algorithm used to calculate the difficulty level is defined by the game designer in the game documentation, as stated in Section 4, and it is one of the component of the formal system of any game (at least for RPGs including different types of mobs). Actually, the formal (mathematical) system underlying each game (tabletop game included) is different and its definition is one of the key aspects of a game designer work, consequently no generalization is possible, since it would mean to put heavy limitations on the creativity of game designers. Hence, we have decided to let the designer chose how to calculate the difficulty level in order to maximize the flexibility of the tool: different games (and game genres) may require very different definitions of the difficulty level for the enemies that they contain (see Section 6.1 for an example). After this last step, the two new monsters are added to the list of the monster's archetypes for the current population, and they can be instantiated in game.

4.10 DRAGON, flow, learning and dramatic elements

As we stated in Section 3, we have based the design of the DRAGON algorithm on the concepts of flow, on the relation between play and learning and on the presence of dramatic elements. In the following Table 5, we summarize how those principles have been mirrored in each step of the algorithm. In the table, a + stands for a relation between the specific element of the algorithm step and the concept (FL = flow, PL = play and learning, DE = dramatic elements), while a + + indicates a strong relation. If no symbol is present, there is no relevant relation. As it can be perceived from the table, the design of DRAGON is rooted into the basic principle of good game design we have described in Section 3.

Table 4 Example of biomes requirements and priorities

Biome	Priority	Necessary traits	Optional traits
Plains	0	–	–
Desert	1	Fire resistance	Parasite
Water (ocean, lake, river, etc.)	2	Breathe water	
Volcano	2	Fire resistance, breath lava	Scales

Table 5 Relations among DRAGON steps and game design principles

Steps of the DRAGON algorithm	FL	PL	DE
Setup			
Structure of the chromosome			+ +
Starting population (mobs' type and number)	+ +	+	+ +
1. Fitness evaluation			
Diversity D(c)	+ +	+ +	+
Death rate M(c)	+ +	+ +	+ +
Fitness function F(c)	+ +	+ +	
2. Extinction			
Elimination based on age	+	+	+
Distribution of remaining monsters across the difficulty levels	+ +	+ +	+
3. Selection of parents			
Roulette method	+	+	
4. Recombination			
Uniform crossover			+ +
5. Mutation			
Diversity threshold S	+	+ +	+
Diversity degree G(P)	+	+ +	+
Ordinary mutation probability T	+	+ +	+ +
Adaptive mutation probability U	+	+ +	+ +
6. Dependent traits			
Development of dependent traits			+ +
Manifestation M			+ +
7. Biome choice			
Biome type			+ +
Biome priority			+ +
Necessary traits			+ +
8. Difficulty level			
Difficulty of each monster	+ +	+ +	
Distribution of newly created monsters across the different difficulty levels	+ +	+ +	

5 DRAGON plugin

DRAGON has been implemented as a plugin for Unity 3D (using its elective programming language C#), one of the most diffused game engines. This choice allows for a possible easy distribution of the plugin in the Unity asset store: game developers using Unity have in time developed a strong community that shares knowledge and plugins, both for free and not. How the software architecture has been structured goes far beyond the scope of this paper, that focuses on the use of EDPCG to support game designers' work; for this reason we will only briefly describe the interfaces of the application, whose usability and effectiveness have been verified by testing them with real users. Two main interfaces have been developed: one to set the parameters for the generation (Fig. 2) and another one to visualise the outcome of the generation and the genealogical tree of each monster archetype (see Fig. 3 for an example). Both interfaces have been tested with a sample of real users in order to make sure they are understandable and easy to use (see Section 6.2).

5.1 Monsters creation interface

Through this interface (Fig. 2) the user can set all the parameters required to initialize the procedural generation. Hence, she has the possibility to obtain highly personalized mobs, that

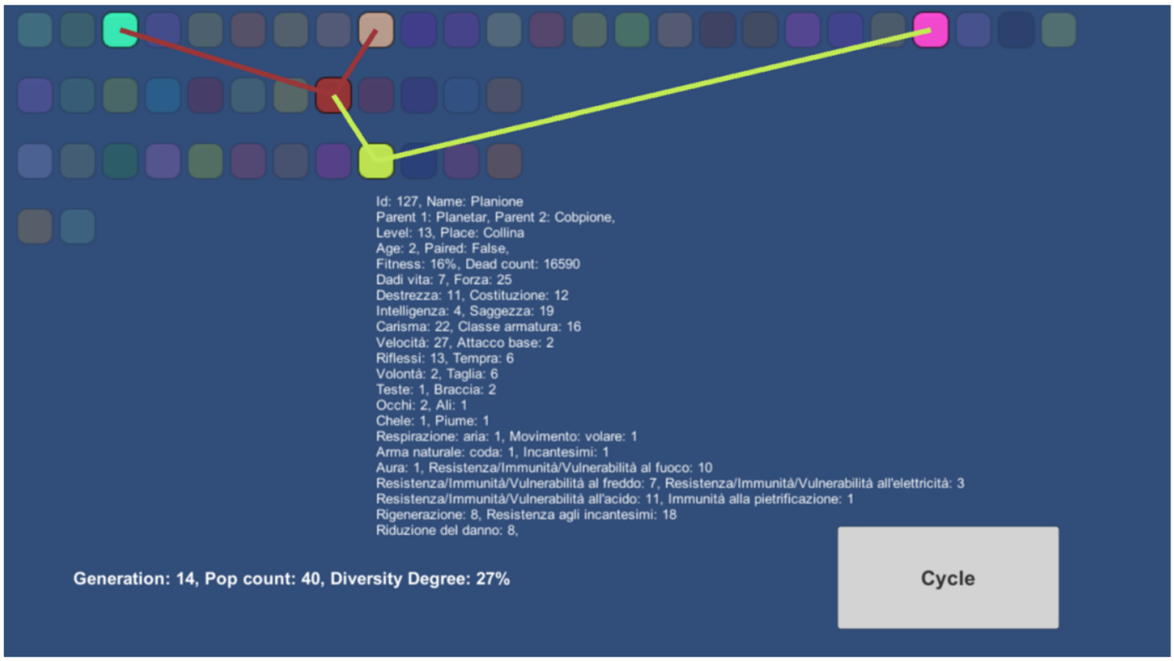

Fig. 3 Monsters' genealogical tree interface

should ideally suit well the type of game she is working on. To push further the flexibility of the tool, several features of the GA are excludable. Using the monster's creation interface, the game designer can define ((*) means that the feature can be excluded):

- the length of the chromosome, how many genes it contains and what each gene represents. For each gene she must enter a name, the starting bit in the chromosome and the length (in bits)
- the set of monsters in the starting population (name and chromosome). Chromosomes can be entered as binary strings or as a list of integers. Each integer represents the value of a corresponding gene in the chromosome
- (*) dependant traits. If this feature is deactivated, the corresponding step in the algorithm will be skipped
- (*) biomes of the game world, setting also their priority and the necessary and optional traits for the mobs to be able to survive in each of them
- (*) difficulty levels
- (*) the formula to calculate the difficulty level of new mobs
- (*) whether the death rate count is active. If not, $F(c)$ will only use the diversity value
- the values of: maximum generation number (E), the minimum ($SMIN$) and maximum ($SMAX$) number of mobs for each difficulty level (if no levels have been defined, these thresholds are used for the total number of mobs in the whole system), the mutation probability (T), the diversity threshold (S), the probability for dependant traits to appear (M)
- the values for α and β in the fitness function $F(c)$.

Once the designer has set all the above parameters, she can move on to the next phase and start to interact with the GA at the core of DRAGON. For example, through an appropriate interface, she can set the number of cycles to perform.

5.2 Monsters genealogical tree interface

The second relevant interface we have developed and tested is that for visualizing the monsters' generation history (Fig. 3). The design of this interface has been particularly delicate, since its clarity and efficacy are of overwhelming importance to give an effective support to the game designer. The interface should have been able to keep track of the monsters' archetype in each generation, of their relationships, of their characteristics (described by their chromosomes), and of whether they are still alive (i.e., present in the game) or they have become extinct. We have refined it by interacting with a sample of testers (see Section 6.2).

The final version of the interface allows to track the complete evolution of a monster population and to interact with it, and also to start a cycle of the GA. As shown in Fig. 3, mobs' archetypes are represented by randomly coloured cells arranged in rows (semi-transparent cells stand for extinct monsters). Each row represents the population in a generation. The current generation number, the current population numerosity and its diversity value are displayed at the bottom left area of the screen. A new generation can be started by pressing the button "cycle".

By selecting one of the cells, it becomes highlighted and all the information about the related chromosome are shown (id, name, parents, difficulty level, biome, age, whether it has reproduced, fitness value, death rate and value for each gene). The complete genealogic tree of the selected monster is also highlighted. To note that, to simplify backtracking and reminding the genealogy of each monster, the name of newly generated monsters is a combination of the first and the second halves of the names of its parents (thus the coupling of a syren and a goblin will produce a sylin and a gobren).

6 DRAGON testing

Once the prototypal plugin has been developed, we needed to validate it at least from two points of view: firstly, we wanted to verify whether DRAGON was able to produce generations of mobs able to satisfy our goal in terms of diversity, coherence and difficulty. In other words, we wanted to understand if the archetypes produced by DRAGON were potentially able to generate a stream of populations of mobs enough varied but, at the same time, believable and able to challenge players with different skill levels. Secondly, we wanted to make sure that our tool was simple and comfortable enough to be understood and used by a video game designer (that may not have a strictly technical background). In the same vein, we wanted to collect some opinions about the perceived usefulness of DRAGON as a tool to support the game designers' work.

Especially in the first case, the validation has been a quite complex. As a matter of fact, DRAGON is not a game, but just an experimental tool based on the EDPCG paradigm aimed at supporting the game designer work. It produces archetypes of mobs in the form e.g. of binary strings, that are not very "effective" in terms of entertainment. To have a real mob for a game we would need at least a tool able to automatically generate 3D models, animations, and visual/audio effects for each mob produced by DRAGON. Such a tool does not yet exist, to the best of our knowledge, and its design could be the subject of a future ad hoc research. Moreover, to appropriately validate our algorithm, we would need a "realistic" testing, that implies the development of a MMORPG that embeds DRAGON, and to involve thousands of players for a time span long enough to go through different mobs' generations. This would

need a development process that requires a huge effort in terms of both time and money (e.g., the development cost of World of Warcraft has been estimated to be around 200 million dollars), this would go well beyond the scope and the limited resources of a research project. Finally, even in the hypothesis that the embedding of DRAGON into an actual MMORPG would be a viable solution, a game affects players through many different channels (visual, audio, storytelling, etc.) that intermingle to create the "player experience", hence it would be quite hard to isolate and accurately measure the contribution of DRAGON to the overall experience. Nonetheless we needed at least a preliminary validation, hence, the only feasible solution we have devised has been to develop an appropriate testing environment to simulate what would have happened in a real multiplayer game. We have designed and developed an agent-based simulation that should approximate accurately enough a certain set of aspects of a MMORPG. These aspects are those relevant to verify whether DRAGON would perform effectively from the perspective of offering a flexible and varied population of mobs and to which extent this population adapts to the behaviour of players.

The second aspect to test should have been easier to test, since we only needed to hire some game designers and ask them to test the plugin for its usability and effectiveness, but the validation has been to some extent reduced due to the current unfavourable sanitary situation. The main goal of this testing phase was, as we already stated, double sided: on the one hand we wanted to verify whether the interfaces to interact with the algorithm were simple enough to understand, and, on the other hand, we needed to have at least a preliminary feedback on the meaningfulness of the tool as a support to the work of game designers, both in terms of the algorithm structure and the outcomes it produces.

6.1 Simulating a game supported by DRAGON

As pointed out before, to obtain a preliminary validation of DRAGON, we needed to simulate the subset of aspects of a MMORPG that focus on the interaction with the monsters. In particular, we concentrated on the variety of monsters we can obtain across a certain number of generations. It is important to remind that the "real" experience that a player will have from fighting the 3D animated model of a mob in an actual game is the product of many aspects, that include visual effects, audio, setting, context of the fight, etc. All these aspects, beside the fact that cannot be easily simulated – especially for a MMORPG –, should not be considered for our purposes. Actually, what we need to test is whether or not DRAGON produces streams of monsters potentially able to sustain the player engagement in the game. The real effect that the mobs would have on players in an actual game would suffer the bias introduced by "aesthetic" and story-based components that may heavily impact the player emotionally. That is to say that a potentially adequate archetype of mob might become more or less intriguing depending also on aspects that cannot – and must not – be controlled by DRAGON. Consequently, we set up an agent-based simulation that helped us to measure the *diversity* and the *coherence* of the produced population, while the distribution of the mobs across the difficulty levels was intrinsically controlled by the algorithm. Both aspects, as we stated, contribute to the believability of the game world and to the player engagement with the game. The simulation, to maintain coherence also on the technical side, has been developed in Unity 3D.

The aspects that we considered relevant to include in the simulation are those more strictly related to the interaction among the *mobs* and the *players* in a massive online game. Hence, our first need has been to define a system to describe monsters and their difficulty level. Since we

wanted to avoid introducing distortions in the simulation derived from poorly designed mobs (we are not professional game designers), we have decided to adopt the monsters system of Dungeons & Dragons (D&D) by Gary Gygax and Dave Arneson [19]. D&D is the founder of the pen and paper RPGs genre. It has been published in 1974 and it is probably the most renown and the best-selling RPG of all the times. Moreover, countless video games have based – and still base – their setting and rule system on D&D (some among the most famous are: Baldur's Gate by BioWare, 1998; Planescape: Torment by Black Isle Studios, 1999 and Neverwinter Nights by BioWare, 2002). Hence D&D it is a natural choice to test an EDPCG aimed at producing monsters for video games based on the RPG paradigm. Last but not least, the huge diffusion and the enormous player-base of D&D is a warranty that the formal system of the game is balanced and effective. In particular, to design the simulation we have used as testbed, we have adopted the set of rules contained in the Player's and Master's Manuals of the 3.5 D&D version [12].

To represent all the characteristics of D&D monsters, that in the original game are described by words and in terms of "dice rolls", we needed a 200 bits chromosome, containing 89 genes grouped in 3 classes (see Table 6). In the abilities class, 29 genes out of 33 may have one or more dependencies. This means that a monster that inherits a certain physical trait can develop any related ability (e.g., inheriting claws offers the possibility to be able to dig and/or to use them in combat). As possible spawning points for new mobs, we have included 9 among all the biomes present in D&D. The starting population of the simulation was composed by 25 monsters, selected among those described in the D&D manuals. To choose them we have followed some commonsense criteria of "good game design": maximize the diversity among monsters and the representativeness in terms of types of mobs and difficulty for the player. Moreover, we have discarded monsters that do not use sexual reproduction methods (e.g., undead), because having them to reproduce would have violated the consistency of the metaphor, thus undermining the believability of the game environment (no one could easily imagine a zombie giving birth). For the sake of simplicity, we have mapped the 20 difficulty levels of D&D monsters (i.e. how difficult is for the player to kill the monster and survive) on 8 levels, as showed by Tab.7.

To avoid unbalancing the simulation, we have adopted the key lines presented in D&D manuals to define the formula to automatically calculate the difficulty level of a mob, starting from its chromosome. According to the D&D rules, around 20 different characteristics related to each monster contribute to define its difficulty level. These characteristics include, for example: size, any magical ability, immunity to specific types of attacks, life points, etc. (see [12] for the complete list). To validate it, we have then verified how much accurately it could calculate the difficulty level of monsters present in the manual (for which the difficulty level has been already set by the game authors). To note that in the pen & paper D&D game, it is up to the player that acts as "Dungeon master" to manually calculate the difficulty level of new monsters she decides to introduce in the game, in our tool this operation has been automatized.

Table 6 Chromosome structure for D&D monsters

Gene class	Genes number	Bits
Basic	13	78
Physical aspect	43	61
Abilities	33	61
Tot.	89	200

Table 7 Starting population with difficulty levels

Monster	Diff. level D&D	Diff. level DRAGON	Monster	Diff. level D&D	Diff. Lev. DRAGON	Monster	Diff. level D&D	Diff. level DRAGON
Goblin	1	1	*Arpy*	4	3	*Frost worm*	12	5
Orc	1	1	*Griffin*	4	3	*Kraken*	12	5
Kobold	1	1	*Pixie*	4	3	*Beholder*	13	5
Gnoll	1	1	*Djinn*	5	3	*Planetar*	16	6
Triton	2	2	*Manticore*	5	3	*Red dragon*	20	8
Badger (cruel)	2	2	*Troll*	5	3	*Pit devil*	20	8
Werewolf	3	2	*Hydra (7 heads)*	6	3	*Balor (demon)*	20	8
Ogre	3	2	*Stone giant*	8	4			
Scorpion	3	2	*Gorgon*	8	4			

This implied that the game designer must enter the formula to calculate the difficulty level she has defined in the tool before starting to generate monsters, and then DRAGON will automatically calculate the difficulty of new mobs. Entering the formula is a quite easy operation, since practically any game designer that works on video games has enough familiarity with very diffused game engines like Unity 3D to be able to enter a short formula in C# when she is prompted to do it.

Once the starting population of monsters and the difficulty level equation have been in place, we needed to decide how to simulate the interaction with players, in order to collect data to feed the fitness function. We created a set of agents that simulate human players. Again, our guiding principle has been to include in the simulation only those aspects that are relevant to test the effectiveness of DRAGON in generating a stream of monsters. To be coherent with D&D, each agent plays as a character, whose experience (acquired mainly by killing monsters, as described in D&D manuals) starts at level 1 and can grow till level 20. We left aside all those aspects related to the cosmetic personalization of the character (such as race, class, sex, appearance, clothing, etc.), since they are functional mainly to the self-identification of the player with the character, hence they are not relevant for our purposes. We also needed to mimic the player population in a MMORPG [5]. Typically, the player base is composed by players that spend a different amount of time playing: it ranges from the so-called *casual* players (busy people that play "when they have the time") to the *hardcore* gamers, that spend a few dozen of hours weekly "in game". In MMORPGs players tend to form strong bonds and communities (a strong motivator to keep them playing a certain game), but this aspect is not relevant for our purpose. Hence, we needed to simulate a huge player base, that grows in time (a successful MMORPG attracts more and more players – e.g. WoW reached more than 14 million players in few years) and that is composed by players spending different amounts of their free time playing. This last point is particularly relevant, since it will affect the velocity of the character progression, thus its ability to compare with mobs increasingly difficult to overcome. Last but not least, different players may prefer to fight with different monsters, due to a wide variety of factors (e.g., personal taste, equipment or characteristics of their character, story-related mission to accomplish, etc.). To mimic the aspects of a real game, the simulation starts with 1000 agents (those who start to play as soon as the "new game" hits the market), and the players population grows at the rate of +20 agents at the beginning of each new week. Players are always evenly distributed among hardcores and casuals. For each day of the week, each agent has a certain probability to play: 45% for casuals, 90% for hardcores

Table 8 Alpha and S values for the different tests

	T1		*T2*		*T3*	
	T1.1	*T1.2*	*T2.1*	*T2.2*	*T3.1*	*T3.2*
α	1	0	0.5		0	1
S	0.2		0.25	0.2	0.25	

(thus simulating the different amount of time spent playing by the two categories). Thus, a hardcore player's character gains 1 level per week, while a casual's 1 each 2 weeks. Each agent that plays randomly selects a type of monster among those that have the same level of its character and kills 10 instances of it (thus mimicking the preference for a certain type of monster). The procedural generation of new monsters takes place once per week, at the beginning of the week.

While keeping the simulation structure fixed, DRAGON parameters can be set at different values, depending on which peculiar aspect is under examination or the requirements of a specific game. In particular, we have set: $E = 10$, $SMIN = 3$, $SMAX = 10$, $T = 0.01$, $M = 0.05$, and $\beta = 0.05$, while α and S, vary according to the type of test, as summarized in Table 8.

As a first thing, we wanted to verify *to what extent the values produced by diversity function D(c)* (1) *do affect the diversity of the population by influencing the fitness function F(c)* (3). This is relevant because the fitness function has the task to take into consideration the behaviour of the players. We run two slightly different set of tests by varying α values (see T1 in Table 8): T1.1 maximizes the impact of the *diversity* value in the fitness function and T1.2, on the contrary, minimizes it. For each test, we run 50 separate simulations, each of which performs 800 cycles of procedural generation. For each simulation, we calculated the diversity value after each cycle, and for each test we chose the simulation (out of 50) that had the minimum diversity value. Figure 4 and Fig. 5 summarize, respectively, the results of test T1.1 and T1.2. It is easily observed that in the first test the diversity value is quite high (it varies between 25,55% and 32,2%), while in the second case it rapidly collapses and then oscillates between 22,79% and 29,63%.

Fig. 4 Test T1.1 result (diversity on vertical axis, # of cycles on the horizontal axis)

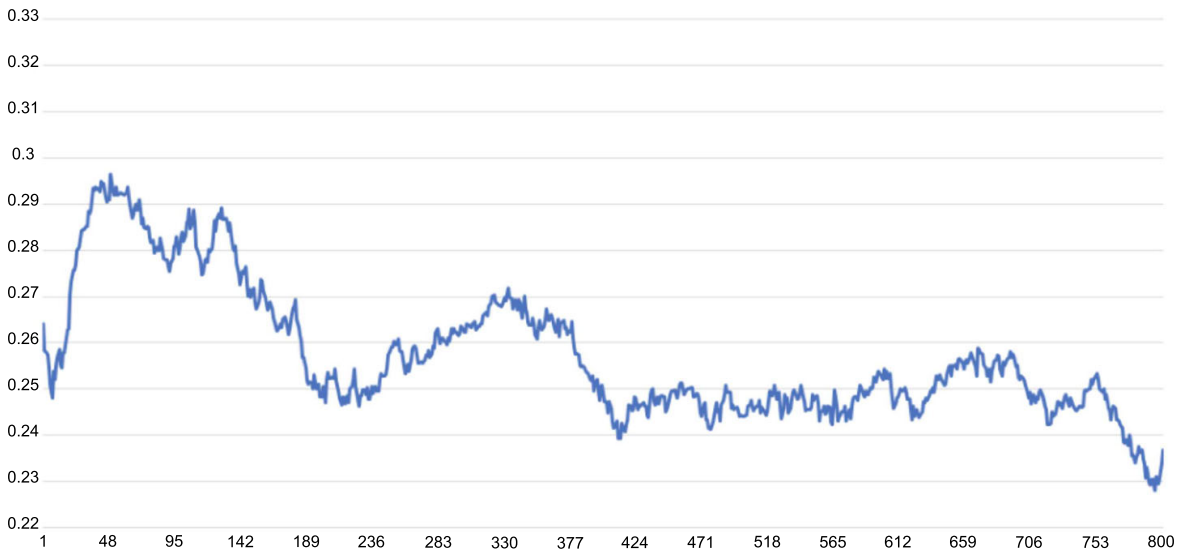

Fig. 5 Test T1.2 result (diversity on vertical axis, # of cycles on the horizontal axis)

Our next goal has been *to verify the efficacy of the adaptive mutation, by controlling the trend of the diversity function* during a simulation. The aim of the adaptive mutation, actually, is to increase diversity in the population when chromosomes start to become too similar. This time too, we run two different sets of test, with different values for S (see T2 in Table 8) to verify how well the adaptive mutation could manage to keep the diversity over the predefined threshold. Again, we run the simulation 50 times, each of which included 800 generation cycles. For each run of the simulation, we calculated the diversity level after each generation cycle, and we kept the lowest values of diversity. The result can be seen in Figs. 6 and 7: in T2.1 the diversity practically never drops under 25%, and only once reaches the minimum in 24.52%. As we foresaw, the adaptive mutation manages to increase the diversity every time it gets too low. In the same vein, in T2.2, the diversity never gets under the threshold and reaches the 23,25% as it minimum value, that is well over $S = 0.2$.

Anyway $F(c)$ is influenced also by the *death rate* function $M(c)$. In particular, we wanted to estimate *how often the player will be in the desirable situation of meeting a monster that is a*

Fig. 6 Test T2.1 result (diversity on vertical axis, # of cycles on the horizontal axis)

Fig. 7 Test T2.2 result (diversity on vertical axis, # of cycles on the horizontal axis)

descendant of a monster she already encountered. To reach this goal, we keep track of how many times – after each generation cycle – each agent encounters a new monster and how often this monster is a descendant of another one the agent encountered some cycles before. The average of this frequency, calculated for all the agents, is the *re-encounter* value (*FRI*). Again, we run two sets of test with different values for α (see T3 in Table 8), to have the two extreme values for the fitness function (i.e., when it depends only on the death rate value and when it depends only on the diversity value, while the death rate has no effect at all). Each test is composed by a set of 20 parallel simulation, each of which performs 20 generation cycles (20 cycles is the expected time for a hardcore player to reach level 20). For each instance of the simulation, *FRI* is calculated after each cycle. For both tests, we calculated the average *FRI* across its simulation instances. As shown by Fig. 8, *FRI* rises rapidly in T3.1 and reaches its maximum value (42.49%) after 20 cycles. As for T3.2 (see Fig. 9), FRI grows slower, and after 20 cycles reaches the maximum value of 38.6%, hence demonstrating that the introduction of the death rate in *F(c)* helps in generating monsters that are relatives of those preferred by the players. To further validate this statement, we also run a test (with the same α than T3.1) on 100 cycles: in this case *FRI* = 71.19% at the end of all the instances of the simulation (see Fig. 10).

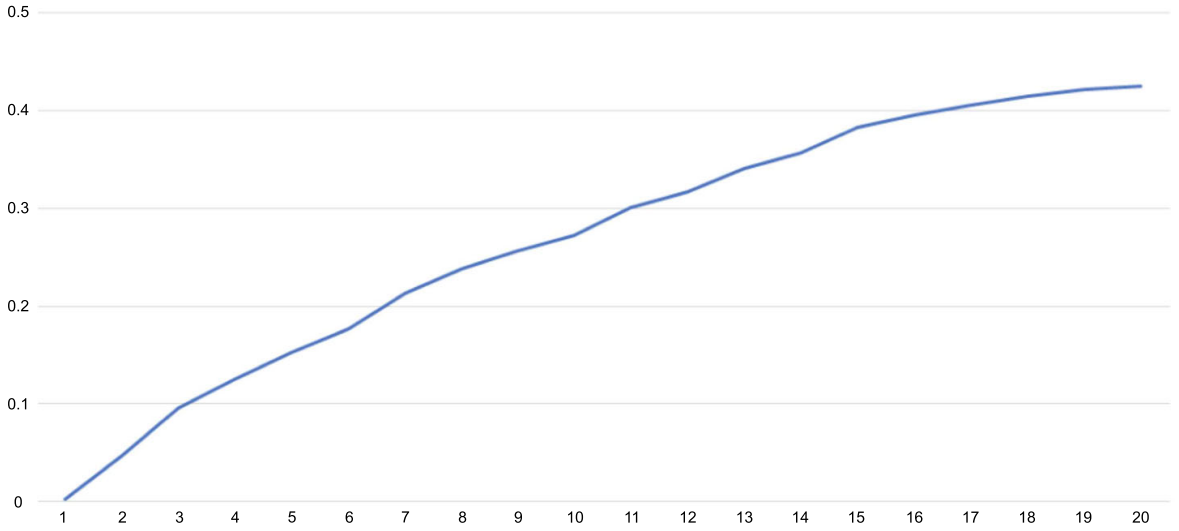

Fig. 8 Test T3.1 result (FRI on vertical axis, # of cycles on the horizontal axis)

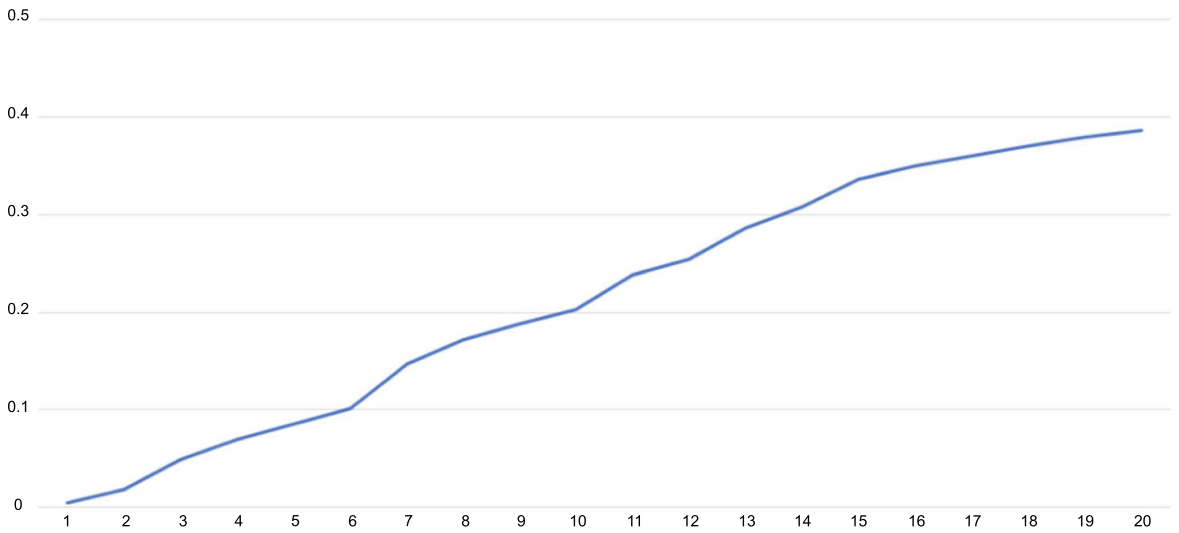

Fig. 9 Test T3.2 result (FRI on vertical axis, # of cycles on the horizontal axis)

6.1.1 Some considerations about the test outcomes

From the tests we have performed it seems that the introduction of a fitness function based on the diversity and death rate values and of an adaptive mutation works well in order to reach our research goals. In particular, $\alpha = 0.5$ seems to be a good compromise, that sustains population diversity even when the threshold S has been violated, but that – at the same time – promotes the re-encounter (*FRI*) by leveraging the death rate value. In the same vein, $S = 0.25$ appears to be a good value, since the diversity value never drops under 23–25%.

Anyway, it is important to remember that the "optimal" value of the different parameters may vary considerably depending on the specific type of game and on how the chromosome has been defined (e.g., with a reduced starting population or a shorter chromosome the diversity may drop quickly). Actually, one of our main goal was to produce an EDPCG tool flexible enough to be easily adapted to a great variety of different games. Hence it is up to the game designer to fine tune the parameters choice basing on the game she is working at.

We have also run a preliminary "technical" test focused at evaluating the performance of different types of hardware when generating monsters through DRAGON. As a matter of fact, to produce an effective support to the game designer work, the software application must not

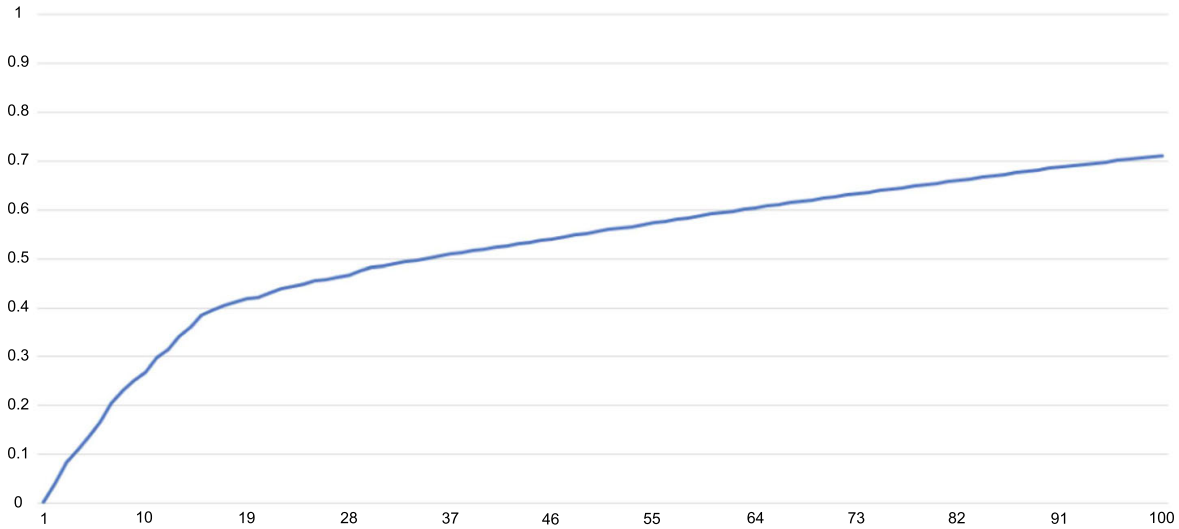

Fig. 10 Test 3.3 result (FRI on vertical axis, # of cycles on the horizontal axis)

be a cumbersome thing: it should produce results quickly, without requiring a dedicated hardware and sparing enough time to give to the designer the possibility to run different simulations – with different chromosomes and/or different values for the parameters – in order to fine tune the types of monsters produced. We have run 100 cycles of the (complete) algorithm, including the simulation with agents after each cycle, on two different hardware configurations. In one case we used a PC with and Intel(R) Core (TM) i7-5500U CPU at 2.40 GHz with 4 GB of RAM, that used Windows 8.1 as operating system. With this configuration, and after repeating the above described test for 10 times, on average the time necessary to complete the cycles has been 22″ (this means on average one cycle each 220 ms). We run the second test on a PC with an Intel(R) Core (TM) i7-4720HQ CPU at 2.60GHz and 16 GB of RAM, the operating system was Windows 10 Home. In this second case, the average time on 10 repetition of the test was around 20″ to complete the 100 cycles (on average one cycle each 200 ms). If we consider that DRAGON is an online PCG algorithm (the generation takes place during the game), and hence that its cycles are called sporadically, we can reasonably conclude that its use will not affect negatively the gaming experience of the players (i.e. it does not introduce lag), and that the game designer can afford to test a certain number of different configurations for her monsters without losing all the time it would have been needed with more traditional approaches, such as physical prototyping or by hand-design each archetype. Last but not least, the features of the monster could even be modified while the game is already played by the players. Probably the real problem would be to find the way to automatically generate – quickly enough – convincing (3D) models and animations to embody the generated monsters. But this is a complex problem that involves different areas of the computer science; hence they are out of the scope of the present work.

6.2 Testing DRAGON with game designers/developers

As we said, the goal of the second testing phase was double-folded: we wanted both to verify the usability of our tool and whether game designers/developers thought that DRAGON could become an effective support to their work.

To accomplish this part of the testing phase we involved a sample of game designers/developers and students in game design/development that habitually use Unity 3D. That prerequisite avoided possible distortions in the collected data deriving from the difficulty to understand the functioning of an unfamiliar game engine. In total we managed to involve 25 testers, 24 of which was under 34 years of age (the remaining 1 was in the age range 45–54. Male and female were represented in the same proportions (24 and 1 respectively), thus reflecting quite accurately the age and gender distribution in the game industry (particularly when considering small indie studios, that are precisely those that would benefit more from tools like DRAGON).

The test was organized in different phases. First of all, testers were explained the purpose of DRAGON and they received a detailed user's manual. The manual described the overall functioning of the algorithm and of the plugin and how to configure each single parameter in order to obtain the desired population of mobs. It also contained several examples, to make it easier for them to understand the effect of each parameter on the generation process. It also instructed the tester about how to define the difficulty level function by modifying the code of a sample test function provided by us. The whole group of testers easily understood the instruction of the manual and judged its content complete and easy to follow. Their answers have been collected in an unstructured way.

During the following phase, the testers were asked to use the plugin. They had to initialize DRAGON by assigning values to the different parameters and then start the generation of mobs and produce several generations. They have been let free to experiment to their liking and as long as they wanted, to let them explore all the features of DRAGON according to their interests and curiosity.

In the third phase, the testers were asked to fill anonymously in a questionnaire. It contained three sections. The first one was devoted to collect some demographical data. The second section was devoted to gain understanding about the playing habits of the testers (preferred game genres, hour spent weekly playing, gaming platforms used, etc.). As far as the preferred genres were concerned, we noticed that 19 testers out of 25 were lovers of the RPG or MMORPG genres (see Fig. 11– more than one answer was possible), thus giving a reasonable guarantee that they were able to give sensible and meaningful feedbacks. Moreover, since RPG/MMORPG games contains many activities common to action and adventure games, they high percentage of preferences for those two genres further reassured us about the relevance and significance of the feedbacks we collected.

Other information we gathered focused on the hours spent playing games each week: 8 testers played 21 or more hours, 5 between 16 and 20 h, 5 from 11 up to 15 h, 2 between 6 and 10 h and the last 5 played 5 h or less. Overall, 18 testers play more than 10 h a week. We could hypnotize than a quite good amount of their time is spent on RPG, MMORPG, action, and adventure games. This implied that our testers were a quite qualified group to judge DRAGON.

We also investigated which were the platforms they use to play. Multiple answers were possible. As showed in Fig. 12, nearly all the testers play on their PC, that is the elective platform for RPG and MMORPG games.

The next and last section of the questionnaire was devoted at collecting feedbacks about DRAGON. Using appropriate sets of questions (both open and closed), we investigated whether understanding it functioning and using it was easy, if it could effectively support a video game designer work for multiplayer RPGs, in which ways the plugin could have been improved and a general opinion on the user experience. On purpose, we have included few quite "general" closed answer questions. The reason of this choice is that DRAGON is a support tool that currently is not integrated in any game, hence it would have been difficult for them to foresee in a very detailed way the impact of generated mobs in an actual game. This mirrors the difficulties we had to face when testing the plugin (Section 6.1). Hence, we have

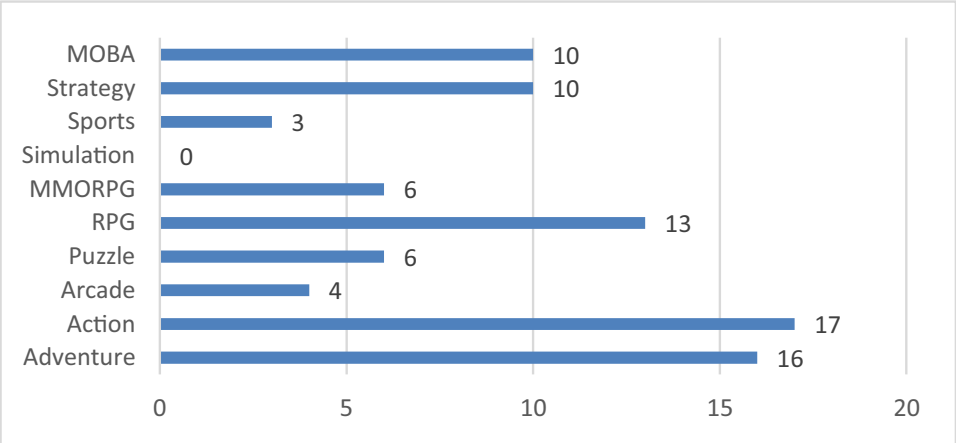

Fig. 11 Testers' favourite game genres (more than one answer was possible)

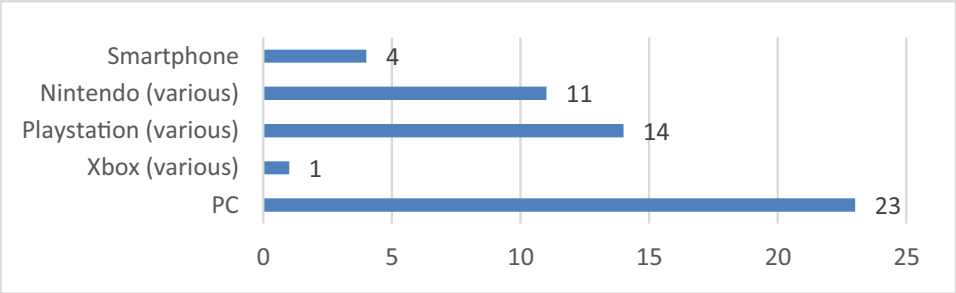

Fig. 12 Testers' favourite gaming platforms (more than one answer was possible)

opted for prompting them to spend as much time as they wanted experimenting with the plugin and then to give us feedbacks in a less formal way (i.e., by filling open answers and giving suggestions). In particular, to stimulate the generation of articulate and useful feedbacks, we have included several open answer questions on specific topics: "What you did not understand in the plugin?", "Which elements of the plugin are not so easy to use?", "Which improvements you'd like to see?", etc.

Almost the whole sample easily understood the purpose of DRAGON and thought it could become a useful tool for a designer. In particular, to the question "Have you understood how the plugin works?" 15 testers answered "definitely yes" and 10 "yes". In the same vein, 19 testers in the sample answered "definitely yes" to the question "Do you think that DRAGON is easy to use?", while 6 of the remaining testers answered "yes" and 1 of them answered "not so much". Particularly relevant for us has been the answer to the question "Do you think that DRAGON could be a useful tool for a designer of RPGs or MMORPGs?", to which 16 of the testers answered "definitely yes" and 9 "yes".

Anyway, many users reported some difficulties in understanding how to use the configuration interface of DRAGON, and 20 of them gave us useful feedbacks and suggestions about how to improve its usability. Basing on what they reported, we kept fix the overall structure of DRAGON (that has been appreciated and considered potentially useful), but we deeply revised its interface. In particular, we have rectified several problems on both the "input" interface (the one used to input personalized parameters) and the "output" interface (the one that presents the monsters' population and genealogy).

In the first case we have:

- added the possibility to insert the chromosomes of the starting population also as integers, and not only as binaries. This required some touch-ups to DRAGON code too
- added a feature that automatically deactivates all the connected fields in the input interface when one of them is deactivated. For example, on the deactivation of the dependencies (the game designer decides to not have any dependencies among different genes), also the field "dependent traits mutation rate" is automatically deactivated (see Fig.13).
- changed several field names to make their purpose more easily understandable
- enhanced tooltip texts (see Fig.14).

While in the second case we have made the following changes:

- in the left bottom corner of the interface we have added the current generation number, the population numerosity and the current diversity degree of the population (see Fig.15)

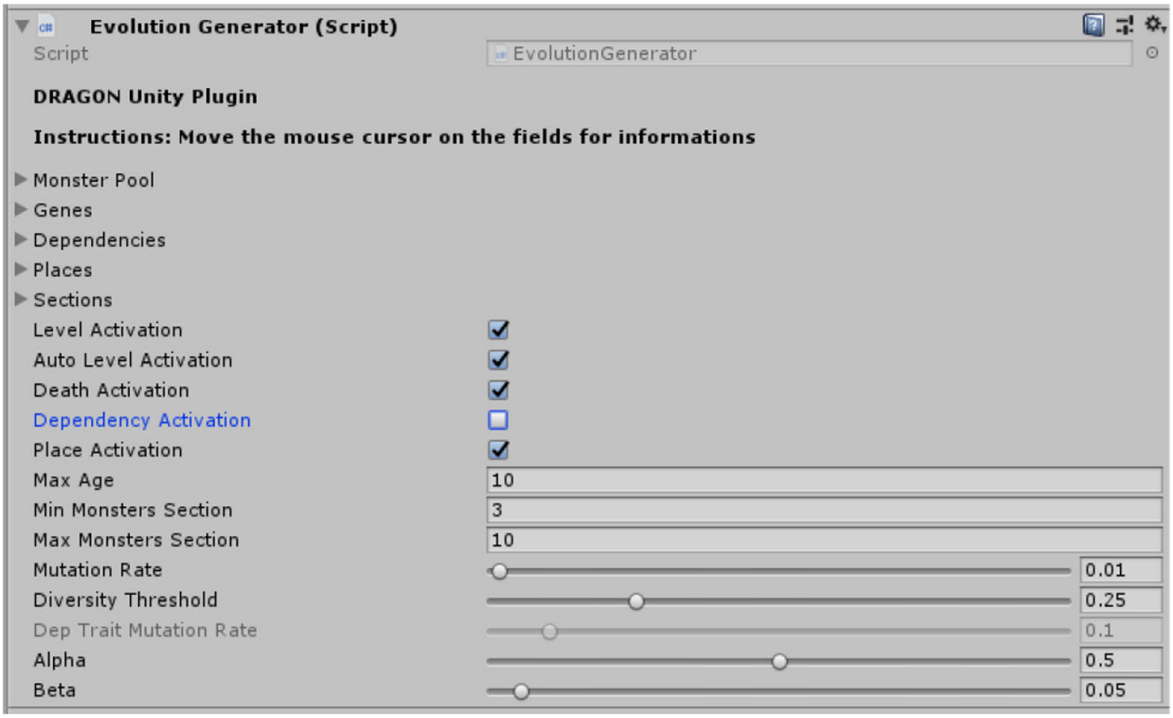

Fig. 13 Automatic deactivation of connected fields

– cells representing extinct archetypes become semi-transparent, thus it is easier to understand both which types of mob are currently present in the population and how the population has evolved in time (this would also help to keep track of the players' preferences for specific treads of mobs)
– by selecting the cell representing a mob, the related information is displayed: chromosome (structure and value of each gene) and genealogical tree (see Fig.15. Please, note that genes' names are in Italian since the game designer that was testing the tool entered them

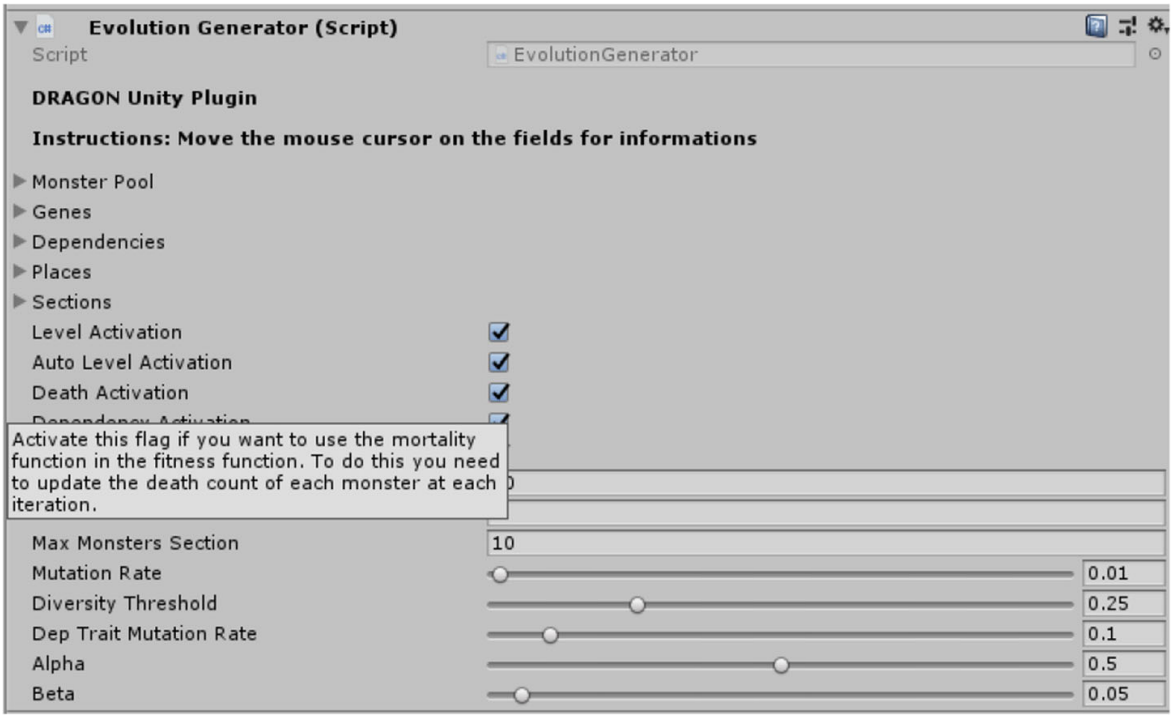

Fig. 14 Example of tooltip box

Fig. 15 Zoom-in on mobs' information

as such). In the first version of the interface this information was showed only when the mouse was over the cell, thus making the consultation rather cumbersome

– we have added the button "Cycle" (see Fig.3) to start a new cycle of generation. Before this function was performed by clicking in any point of the screen and that was not very intuitive
– we have changed the font used to display mobs' information to make them more readable.

According to testers, to which we showed the "new" interfaces during an unstructured meeting, the modifies we made solved all the problems they identified in the first phase of the test.

7 Conclusion and hints for future work

In quite recent time, the work of game designers, level designers and game developers has become more and more complex and demanding. Artificial Intelligence (AI) techniques are

increasingly adopted to support both the game itself and the work of developers. Among the wide zoo of AI approaches and solutions, one that raised the interest of both researchers and practitioners is the procedural generation of contents. Anyway, the automatic generation of contents for an interactive media aimed at making people have fun is not at all an easy matter, since it should "do the math" with the preferences and the behaviour of players. This spiky issue is further complicated in the case of massive online games. Several scholars have tackled this topic, and one of the most promising approaches is that of the EDPCG. The EDPCG paradigm (see Section 2.1) states that – ideally – a game that has some (or all) content that is procedurally generated should be able to foresee the player's preferences. This result could be obtained by modelling the players experience basing on their interaction pattern with some accurately selected gameplay elements. The knowledge harvested by monitoring these elements should be used to feed an appropriate fitness function, whose value changes to dynamically adapt the contents to the players' actions. In our opinion EDPCG alone has some limits, and it could give interesting results only when coupled to some principles of good game design. Actually, dynamically creating and adapting poorly designed contents can hardly improve the overall quality of a game.

In this work, we have tackled the issue of designing and testing DRAGON, an EDPCG tool able to sustain the work of game and level designers facing the problem of suppling – on an ongoing basis – huge quantities of content in order to keep engaged the players of online multiplayer massive video games. In particular, we have focused an under-addressed aspect: that of procedurally generating mobs by keeping into account the preferences of the player and the coherence with the game world. Hence, DRAGON is an online tool able to procedurally generate monsters. It has been implemented as a plugin for Unity 3D, a very diffused video game engine, and it offers to the designers the possibility to personalize the procedural generation in order to make it fit the specific requirements of different game types, while, at the same time, adapting the generation to the players behaviour.

We have based our procedural generation on an ad hoc implementation of GAs. According to the EDPCG paradigm, the value assumed by the fitness function that regulates the generation of new mobs' archetype depends on some actions performed by the players. In particular, we keep track of how many instances of each mob are "killed" by players and we weight it with the diversity of the population, in order to avoid the prevalence of a specific monster type in the mating pool of the GA. Coherently with our assumption that no EDPCG could function appropriately if not coupled to some good game design principles, we have rooted the design of our solution into three key principles of game design (Section 3.1, Section 3.2, Section 3.3): keeping players in the flow, fostering the relation between play and learning and keeping coherence among the dramatic elements of the game. Table 5 in Section 4.10 summarizes how these principles have been embedded into the algorithm of DRAGON. The use of good game design principles during the design phase of DRAGON allowed us also to overcome the intrinsic limits of GOLEM [18]. GOLEM is a stand-alone application for the generation of mobs that provided us a starting point for our work. Table 9 compares GOLEM and DRAGON, highlighting the limits of GOLEM (Section 2.2) that have been overcome by DRAGON. As it can be observed, practically every major problem of GOLEM has been addressed and solved.

Once the plugin has been developed, we have tested it from different perspectives. We wanted to verify whether it was able to create generations of mobs diverse, coherent, and difficult enough to be effective to entertain players. Moreover, we wanted to make sure that DRAGON is a tool simple enough to understand and use for a game designer and whether

Table 9 Comparison between GOLEM and DRAGON

GOLEM	DRAGON
Mobs created without considering players	The outcomes of the players' actions are the basis for the generation of mobs
Generation parameters hardcoded	Game designers can set the generation parameters
Linked to D&D, scarcely flexible	Completely general purpose: chromosomes can have any structure the game designer desires
Offline generator: mobs do not "evolve" with the game, no "natural selection" from the interactions with the players	Online generator: Mobs population evolve with the game, according to the interaction with the players and their preferences
No difficulty levels personalization	Game designers can set how many different difficulty levels they need and how many mobs must be present (at minimum) in each level
No adaptative evolution	Mutation rate can be set by the game designer and can automatically change to maintain diversity
No dependent traits allowed	Game designers can impose that some genes are connected and must express together
Mobs can disappear because randomly selected	Mobs archetypes can disappear only when they are too old
Convergence of the population (no fitness function)	The diversity of the population can be managed by tuning the fitness function parameters
No easy way to keep track of the different generations	Intuitive interface to track mobs' ancestors
Requires knowledge of C++	No coding skills required
Stand-alone application	Pugin for Unity 3D, with easy-to-use interactive user interface

indie game designers considered it a potentially useful tool for supporting their everyday work. To address the first topic we have set up an agent-based simulation (Section 6.1) that we deemed a good enough proxy to obtain a preliminary test about how much the algorithm was able to meet the requirement we had set. Our main concern has been the evaluation of the fitness function, to make sure that – even in extreme situations – it would guarantee populations that are enough diverse, coherent and evenly distributed across the different difficulty levels. We payed particular attention to avoid including in the simulation "game design errors" that would have distorted the results. The encouraging results obtained from this testing phase should be considered preliminary in the sense that a further validation with real player in a prototypal game should be performed to fine tune the algorithm. Unfortunately, due to the development time and cost required to create an actual MMORPG/MMO it is not possible to perform such a test in this stage of the research. Anyway, we verified that the values assumed by the diversity function $D(c)$ correctly affect the fitness function $F(c)$ and that the adaptive mutation regulates the value of the diversity as we expected. Last but not least, basing on the assumption that a player that prefers a certain type of mob would be happy to combat monsters that share some common trait with it, we have defined the re-encounter rate *FRI* (Section 6.1). Using *FRI*, we have verified that the parameter $M(c)$ in $F(c)$ correctly influences the generation by encouraging the reproduction and evolution of mobs more often killed by players. All the result obtained so far confirmed our hypothesis.

A subsequent phase of testing focused on the effectiveness of the interfaces of DRAGON (Section 6.2). We have recruited a sample of 25 testers that mirrors quite accurately the characteristics of game designers working in small indie studios, that is precisely the type of users that could benefit of tools like DRAGON (big studios generally have their own proprietary solutions). The sample is quite limited, but – despite our desire – due to the current

sanitary situation, it has not been possible to enlarge it, nor to run any other testing session. The outcomes of the testing provided us with some very useful feedbacks, that allowed us to refine both the input and the consultation interfaces, by fixing some usability problems. We also implemented a suggestion aimed at increasing the flexibility of the tool (i.e., the possibility to also use integers in the chromosome). Also, the whole sample of testers declared that DRAGON would be a useful tool to support their work.

The last test we performed has been more on the technical side. We have run some simulations to make sure that the performances of DRAGON on consumer-level hardware were good enough to be easily exploited by designers of indie companies. In this case too, the outcome met our expectations.

From the point of view of perspective future work, it is obvious that the preliminary promising results that we have obtained so far would benefit from a further validation in a real MMORPG. Unfortunately, the development and maintenance of this game type requires financial, technical, and human resources that goes well beyond any research project. Moreover, such a testing would require the development of a procedural generator – integrated with DRAGON – aimed at automatically producing the 2D/3D models, animation, visual effect and audio effects for each new mob. Such a tool would require also a careful refinement of the physical traits that monster should or should not have (e.g., in order to facilitate the union of different meshes).

Anyway, from our point of view the most intriguing possible development of DRAGON is the inclusion – as parameters for the generation – of even more aspects. In particular, we intend to revise and further develop the plugin in order to take into account also the Bartle's player types [5]. The Bartle's theory is quite universally applied in video game design. It claims that players (of any genre of game – both video and tabletop) can be clustered according to some well-defined common behavioural traits. By keeping track of players' actions that are typical of each Bartle's type would further increase the possibility to personalize the content produced by DRAGON.

References

1. Alhejali AM, Lucas SM (2010) Evolving diverse Ms. Pac-man playing agents using genetic programming. In: 2010 UK workshop on computational intelligence (UKCI), Colchester, pp 1–6. https://doi.org/10.1109/UKCI.2010.5625586
2. Andrade G, Ramalho G, Santana H, Corruble V (2005) Challenge-sensitive action selection: an application to game balancing. IAT '05: Proc. of the IEEE/WIC/ACM international conference on intelligent agent technology, Washington, DC
3. Andrade G, Ramalho G, Santana H, Corruble V (2005) Automatic computer game balancing: a reinforcement learning approach. In Proceedings of the fourth international joint conference on autonomous agents and multiagent systems (AAMAS '05). ACM, New York, 1111-1112. DOI=http://dx.doi.org/https://doi.org/10.1145/1082473.1082648
4. Barros GAB, Carvalho LFBS, Silva VRM, Lopes RVV (2011) An application of genetic algorithm to the game of checkers. In: 2011 Brazilian symposium on games and digital entertainment, Salvador, pp 63–69. https://doi.org/10.1109/SBGAMES.2011.14
5. Bartle RA (2003) Designing virtual worlds. New Riders Publishing, Indianapolis ISBN: 0-13-101816-7
6. Benbassat A, Sipper M (2011) Evolving board-game players with genetic programming. In: GECCO '11: proceedings of the 13th annual conference companion on genetic and evolutionary computation, pp 739–742. https://doi.org/10.1145/2001858.2002080
7. Chen G, Esch G, Wonka P, Müller P, Zhang E (2008) Interactive procedural street modeling. ACM Trans Graph 27(3) Article 103 (2008), 10 pages:1–10. https://doi.org/10.1145/1360612.1360702

8. Claxton G (1999) Hare brain, tortoise mind: how intelligence increases when you think less. The Ecco Press EAN: 9780060955410
9. Compton K, Mateas M (2006) Procedural level design for platform games. In: AIIDE'06: proceedings of the second AAAI conference on artificial intelligence and interactive digital entertainment, pp 109–111
10. Csikszentmihalyi M (1990) Flow: the psychology of optimal experience. Harper & Row, New York
11. Dormans J (2011) Level design as model transformation: a strategy for automated content generation. In: Proceedings of the 2nd International Workshop on Procedural Content Generation in Games (PCGames '11). ACM, New York, Article 2 , 8 pages. https://doi.org/10.1145/2000919.2000921
12. Dungeons and Dragons (2003) Monster Manual: core rulebook III v 3.5. Wizard of the Coast
13. Ebert DS, Kenton Musgrave F, Peachey D, Perlin K, Worley S (2002) Texturing and modeling: a procedural approach (3rd ed.). Morgan Kaufmann publishers Inc., San Francisco
14. Esparcia-Alcázar AI, Moravec J (2012) Fitness approximation for bot evolution in genetic programming. Soft Comput:1–9
15. Frade M, de Vega FF, Cotta C (2010) Evolution of artificial terrains for video games based on accessibility. Applications of Evolutionary Computation. Springer, Berlin, pp 90–99
16. Frade M, de Vega FF, Cotta C (2012) Automatic evolution of programs for procedural generation of terrains for video games. Soft Comput 16(11):1893–1914
17. Fullerton T (2014) Game design workshop: a Playcentric approach to creating innovative games, 3rd edn. CRC Press, Taylor & Francis
18. Guarneri A, Maggiorini D, Ripamonti LA, Trubian M (2013) GOLEM: Generator Of Life Embedded into MMOs. Proc ECAL 2013: Twelfth Eur Conf Artificial Life:585–592. https://doi.org/10.7551/978-0-262-31709-2-ch084
19. Gygax G, Arneson D (1974) Dungeons & Dragons, vol 19. TSR Tactical Studies Rules, Geneva
20. Halim Z, Raif Baig A (2011) Evolutionary algorithms towards generating entertaining games. Next Generation Data Technologies for Collective Computational Intelligence. Springer, Berlin, pp 383–413
21. Hastings E, Stanley K (2009) Evolving content in the galactic arms race video game. In: CIG2009–2009 IEEE symposium on computational intelligence and games, pp 241–248. https://doi.org/10.1109/CIG.2009.5286468
22. Hastings EJ, Guha RK, Stanley KO (2009a) Demonstrating automatic content generation in the galactic arms race video game. In: Proceedings of the artificial intelligence and interactive digital entertainment conference demonstration program (AIIDE'09). AAAI, Menlo Park (2 pages) https://eplex.cs.ucf.edu/papers/hastings_aiide09.pdf
23. Hastings EJ, Guha RK, Stanley KO (2009b) Automatic content generation in the galactic arms race video game. IEEE Trans Comput Intell AI Games 1(4):245–263. https://doi.org/10.1109/TCIAIG.2009.2038365
24. Hastings EJ, Guha RK, Stanley KO (2009c) Evolving content in the galactic arms race video game. In: 2009 IEEE symposium on computational intelligence and games. Milano, pp 241–248. https://doi.org/10.1109/CIG.2009.5286468
25. Holland JH (1975) Adaptation in natural and artificial systems. The University of Michigan
26. Hom V, Marks J (2007) Automatic design of balanced board games. In: AIIDE'07: proceedings of the third AAAI conference on artificial intelligence and interactive digital entertainment, pp 25–30
27. Hong Y, Liu Z (2010) A first study on genetic algorithms based-evolvable motivation model for virtual agents. In: 2010 international conference on multimedia technology, Ningbo, pp 1–4. https://doi.org/10.1109/ICMULT.2010.5630635
28. Huizinga J (1949) Homo Ludens: a study of the play-element in culture. Volume 15 of Beacon paperbacks Reprint Publisher Temple Smith, 1970
29. Hunicke R, LeBlanc M, Zubek R (2004) MDA: a formal approach to game design and game research. In: Proceedings of the 2004 AAAI workshop on challenges in game artificial intelligence, San Jose
30. Inführ J, Raidl GR (2012) Automatic generation of 2-antwars players with genetic programming. Computer Aided Systems Theory–EUROCAST 2011. Springer Berlin 248–255
31. Isaac AR (1992) Mental practice: does it work in the field? Sport Psychologist 6(2):192–198
32. Johnson S (2005) Mind wide open: your brain and the neuroscience of everyday life. Scribner
33. Klein GA (1999) Source of power: how people make decisions. MIT press
34. Koster R (2013) A theory of fun for game design. O'Reilly Media, Inc., Newton
35. Lee S, Jung K (2006) Dynamic game level design using Gaussian mixture model. In: Yang Q, Webb G (eds) Proceedings of the 9th Pacific rim international conference on artificial intelligence (PRICAI'06). Springer-Verlag, Berlin, pp 955–959
36. Maggiorini D, Mannalà M, Ornaghi M, Ripamonti LA (2015) FUN PLEdGE: a FUNny Platformers LEvels GEnerator. In Proceedings of the 11th Biannual Conference on Italian SIGCHI Chapter (CHItaly 2015). ACM, New York, 138–145. DOI: https://doi.org/10.1145/2808435.2808451

37. Mark B, Berechet T, Mahlmann T, Togelius J (2015) Procedural generation of 3D caves for games on the GPU. In: Proceedings of the 10th international conference on the foundations of digital games (FDG 2015), Pacific Grove California
38. Mazza C, Ripamonti LA, Maggiorini D, Gadia D (2017) Fun Pledge 2.0: a funnny platformers levels generator (rhythm based). In: CHItaly '17: proceedings of the 12th biannual conference on Italian SIGCHI chapter, article no.: 22, pp 1–9. https://doi.org/10.1145/3125571.3125592
39. Miller GA (1956) The magical number seven, plus or minus two: some limits on our capacity for processing information. Psychol Rev 63(2):81–97. https://doi.org/10.1037/h0043158
40. Missura O, Gartner T (2009) Player modeling for intelligent difficulty adjustment. In: DS '09: proceedings of the 12th international conference on discovery science. Springer-Verlag, Berlin, pp 197–211
41. Mitchell M (1998) An introduction to genetic algorithms. MIT press
42. Mora AM et al (2010a) Evolving bot AI in unreal. In: EvoApplicatons'10: proceedings of the 2010 international conference on applications of evolutionary computation - volume part I, pp 171–180. https://doi.org/10.1007/978-3-642-12239-2_18
43. Mora AM, Moreno MA, Merelo JJ, Castillo PA, Arenas MG, Laredo JLJ (2010b) Evolving the cooperative behaviour in unreal™ bots. In: Proceedings of the 2010 IEEE conference on computational intelligence and games, Dublin, pp 241–248. https://doi.org/10.1109/ITW.2010.5593347
44. Mora AM, Fernández-Ares A, Merelo JJ, García-Sánchez P, Fernandes CM (2012) Effect of Noisy fitness in real-time strategy games player behaviour optimisation using evolutionary algorithms. J Comput Sci Technol 27(5):1007–1023
45. Mourato F, dos Santos MP, Birra F (2011) Automatic level generation for platform videogames using genetic algorithms. In: ACE '11: proceedings of the 8th international conference on advances in computer entertainment technology, article no.: 8, pp 1–8. https://doi.org/10.1145/2071423.2071433
46. Müller P, Wonka P, Haegler S, Ulmer A, Van Gool L (2006) Procedural modeling of buildings. ACM Trans Graph 25(3):614–623. https://doi.org/10.1145/1141911.1141931
47. Norton D, Ripamonti LA, Ornaghi M, Gadia D, Maggiorini D (2017) Monsters of Darwin: a strategic game based on artificial intelligence and genetic algorithms. In: De Marsico M, Ripamonti LA, Gadia D, Maggiorini D, Mariani I (eds) Games-human interaction. In proc. of GHItaly17 – games-human computer interaction workshop. Held in conjunction with ACM CH. CEUR-WS, Italy. http://ceur-ws.org/Vol-1956/GHItaly17_paper_05.pdf
48. Onieva E, Pelta DA, Godoy J, Milanés V, Pérez J (2012) An evolutionary tuned driving system for virtual car racing games: the AUTOPIA driver. Int J Intell Syst 27(3):217–241
49. Parish YIH, Müller P (2001) Procedural modeling of cities. In: Proceedings of the 28th annual conference on computer graphics and interactive techniques (SIGGRAPH '01). ACM, New York, pp 301–308. https://doi.org/10.1145/383259.383292
50. Prusinkiewicz P, Lindenmayer A (2004) The algorithmic beauty of plants. Springer-Verlag, Berlin
51. Ripamonti LA, Mannalà M, Gadia D, Maggiorini D (2017) Procedural content generation for platformers: designing and testing FUN PLEdGE. Multimedia Tools Appl. 76, 4 (February 2017), 5001–5050. DOI: https://doi.org/10.1007/s11042-016-3636-3
52. Ripamonti LA, Mannalà M, Gadia D, Maggiorini D (2017) Procedural content generation for platformers: designing and testing FUN PLEdGE. Multimedia Tools Appl. 76, 4 (February 2017), 5001–5050. DOI: https://doi.org/10.1007/s11042-016-3636-3
53. Schell J (2014) The art of game design: a book of lenses, 2nd edn CRC Press
54. Schwarz M, Müller P (2015) Advanced procedural modeling of architecture. ACM Trans Graph. 34, 4, Article 107, 12 pages. DOI: https://doi.org/10.1145/2766956, 1, 12
55. Silva de Carvalho LFB, Silva Neto HC, Lopes RVV, Paraguaçu F (2010) An application of genetic algorithm based on abstract data type for the problem of generation of scenarios for electronic games. In: 2010 IEEE international conference on intelligent computing and intelligent systems, Xiamen, pp 526–530. https://doi.org/10.1109/ICICISYS.2010.5658282
56. Smith G, Treanor M, Whitehead J, Mateas M (2009) Rhythm-based level generation for 2D platformers. In: FDG '09: proceedings of the 4th international conference on foundations of digital games, pp 175–182. https://doi.org/10.1145/1536513.1536548
57. Smith G, Gan E, Othenin-Girard A, Whitehead J (2011) PCG-based game design: enabling new play experiences through procedural content generation. In: Proceedings of the 2nd international workshop on procedural content generation in games (PCGames '11). ACM, New York, article 7, 4 pages. https://doi.org/10.1145/2000919.2000926
58. Smith G, Whitehead J, Mateas M, Treanor M, March J, Cha M (2011a) Launchpad: a rhythm-based level Generator for 2-D Platformers. IEEE Trans Comput Intell AI Games 3(1):1–16. https://doi.org/10.1109/TCIAIG.2010.2095855

59. Smith G, Whitehead J, Mateas M (2011b) Tanagra: reactive planning and constraint solving for mixed-initiative level design. IEEE Trans Comput Intell AI Games 3(3):201–215. https://doi.org/10.1109/TCIAIG.2011.2159716
60. Smith G, Othenin-Girard A, Whitehead J, Wardrip-Fruin N (2012) PCG-based game design: creating endless web. In: FDG '12: proceedings of the international conference on the foundations of digital games, pp 188–195. https://doi.org/10.1145/2282338.2282375
61. Sorenson N, Pasquier P (2010) Towards a generic framework for automated video game level creation. In: EvoApplicatons'10: proceedings of the 2010 international conference on applications of evolutionary computation - volume part I, pp 131–140. https://doi.org/10.1007/978-3-642-12239-2_14
62. Spronck P, Ponsen M, Sprinkhuizen-Kuyper I, Postma E (2006) Adaptive game AI with dynamic scripting. Mach Learn. 63, 3 (June 2006), 217–248. DOI=https://doi.org/10.1007/s10994-006-6205-6
63. Stanley KO, Miikkulainen R (2002) Evolving neural networks through augmenting topologies. Evolutionary Comput 10(2):99–127 MIT press
64. Togelius J, Schmidhuber J (2008) An experiment in automatic game design. In: 2008 IEEE symposium on computational intelligence and games, Perth, pp 111–118. https://doi.org/10.1109/CIG.2008.5035629
65. Togelius J, De Nardi R, Lucas SM (2007) Towards automatic personalised content creation for racing games. In: 2007 IEEE symposium on computational intelligence and games, Honolulu, pp 252–259. https://doi.org/10.1109/CIG.2007.368106
66. Togelius J, Kastbjerg E, Schedl D, Yannakakis GN (2011) What is procedural content generation?: Mario on the borderline. In: Proceedings of the 2nd international workshop on procedural content generation in games (PCGames '11). ACM, New York, article 3, 6 pages. https://doi.org/10.1145/2000919.2000922
67. Toy M et al. (1980) Rogue (PC Game)
68. Wong S, Fang S (2012) A study on genetic algorithm and neural network for mini-games. J Inf Sci Eng 28(1):145–159
69. Yannakakis GN (2011) Togelius J (2011) Experience-driven procedural content generation. IEEE Trans Affect Comput 2(3):147–161
70. Yannakakis GN, Hallam J (2009) Real-time game adaptation for optimizing player satisfaction. IEEE Trans Computation Intell AI in Games 1(2):121–133