Towards Diverse Non-Player Character behaviour discovery in multi-agent environments

1st Jan Kirk
The Mærsk Mc-Kinney Møller Institute
University of Southern Denmark
Odense, Denmark
jakir20@student.sdu.dk

2nd Marco Scirea
SDU Metaverse Lab
University of Southern Denmark
Odense, Denmark
msc@mmmi.sdu.dk

***Abstract*—This paper introduces a method for developing diverse Non-Player Character (NPC) behaviour through a multi-agent genetic algorithm based on Map-Elites. We examine the outcomes of implementing our system in a test environment, with a particular emphasis on the diversity of the evolved agents in the feature space. This research is motivated by how diverse NPCs are an important factor for improving player experience.**

We show how our multi agent map-elite algorithm is capable of isolating the evolved NPCs in the chosen feature space. Results showed that variation in agent fitness could be predicted with 40% from agent genomes, when agents played 100 games each.

I. Introduction

The application of Artificial Intelligence (AI) in games is a vast field, encompassing numerous aspects within a game such as generation of content for the game world, like maps, objects, and music; these aspects could be unchanging, after initial creation by AI techniques, or also subject to change over the course of the game, by the use of AI. Content like the aforementioned are important aspects of a game; in terms of creating rich and varied player experiences, they could be classified as environmental aspects of a game.

One of the classical applications of AI in games consists in controlling Non-Player Characters (NPCs). This is usually achieved by creating some structure for chaining scripted behaviours (like a Finite State Machine, as commonly used in the games industry), or by applying some sort of machine learning approach.

This paper proposes a method to manufacture NPCs with diverse fixed behaviours, while maintaining some reasonable level of performance in achieving the agent's goals. The performance could be defined as anything which is desired within the game, e.g., win, interact with human players as much as possible, or make varied meaningful conversation.

As a test environment, we developed a simple game environment where performance is defined as collecting gold, which is a win-condition. In the game, 200 agents will concurrently play while being allowed to act both cooperatively and competitively.

Each NPC is based on a predefined Behaviour Tree (BT), which can be tweaked by MAP-Elites to express different behaviours. The NPCs can gather food, weapons, gold, form groups, or fight against each other. Since their behaviour is defined by a genome (a sequence of numbers), a specific NPC could indeed just have its genome shifted to a slightly different one, in case variance is desired.

979-8-3503-5067-8/24/$31.00 ©2024 IEEE

II. Background / Related Work

A. Quality-diversity algorithms

Quality-diversity algorithms represent a specialized subset of evolutionary algorithms that not only aim to find the best solution but also prioritize maintaining diversity within the solutions discovered. By exploring a wide range of potential solutions and identifying a diverse set of high-performing solutions, quality-diversity algorithms can avoid premature convergence on a single optimal solution and enhance the robustness of the solutions obtained [1]. Examples of quality-diversity algorithms include the Multi-dimensional Archive of Phenotypic Elites (MAP-Elites) and Novelty Search with Local Competition [2]. These algorithms have demonstrated effectiveness in balancing exploration and exploitation, leading to the discovery of a diverse set of high-quality solutions across various domains, such as robotics, optimization, and machine learning [3].

Research applying quality-diversity algorithms to games has shown significant promise in enhancing the generation of game content. Quality-diversity algorithms, such as MAP-Elites, have been utilized to evolve diverse and high-quality solutions for various aspects of game design, including procedural content generation and level design [4]. By promoting diversity in the solutions found, these algorithms have been instrumental in creating a wide range of game content that covers different gameplay scenarios and challenges [5]. For instance, quality-diversity algorithms have been applied to evolve diverse repertoires of game elements, such as decks in card games like Hearthstone, providing insights into game dynamics and aiding in rebalancing game elements [5]. These algorithms have also been instrumental in evolving game levels and scenes through interactive and constrained approaches, allowing for mixed-initiative design in creating levels typical of computer role-playing games [6]. By leveraging the principles of quality and diversity in evolutionary computation, researchers have been able to address challenges in game content generation, such as ensuring variety, balance, and player engagement.

B. *The Map-Elites Algorithm*

The central algorithm for this work is *Map-Elites* (ME), developed by Mouret and Clune [7]. ME is a variant of genetic algorithms (GA). Contrary to most search algorithms, ME finds multiple high-performing solutions within a multidimensional feature space which the user gets to define; this space is called the feature space. However, the search is not conducted in the feature space, but in the space which encompasses all possible solutions, as defined by the genome of a solution.

III. Benchmark environment

The benchmark environment was designed to present a range of viable options for the NPCs to develop different behaviours. It was not developed with the intention of being played by humans, so it is more akin to a simulation, rather than a game. However, the long-term hypothesis is that results from this study could be used in an arbitrary game which requires NPCs to create diverse agents.

Figure 1 shows the view of the playing-field; blue dots are NPCs, one or more little spheres above the NPC represents a group of NPCs, yellow dots are gold, red dots are food, and salmon coloured squares are weapons.

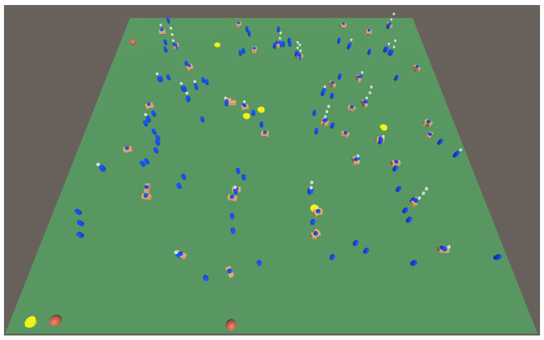

Fig. 1. The benchmark environment while the simulation is in progress.

In the game, the NPCs can observe their surroundings to discover the positions of *objects of interest*:

- **Food** which will replenish their health by 50 HP (they start with 100 HP).
- **Weapons** which will increase their damage output.
- **Gold** pieces, each with value 1. The amount of gold collected is the measure of success in the game.
- **Opponents** (other NPCs) which they can either group with (allowing for cooperation) or attack (competition).

Food, weapons, and gold are picked up when the agent collides with it. However, gold can also be obtained indirectly by killing another NPC, which would result in looting their gold.

The general behaviour of the agents is represented with a behaviour tree (see Fig. 2) that is tweaked through the Map-Elites (see section III-A). In short, the main actions that the agent can execute are:

- **Walking randomly**, when no object of interest has been found.
- **Walk to an object of interest**.
- **Form (or join) a group** with other agents.
- **Fight** with another agent or group of agents. Fights are only resolved through the death of one of the agents/groups, there is no option to flee a fight.

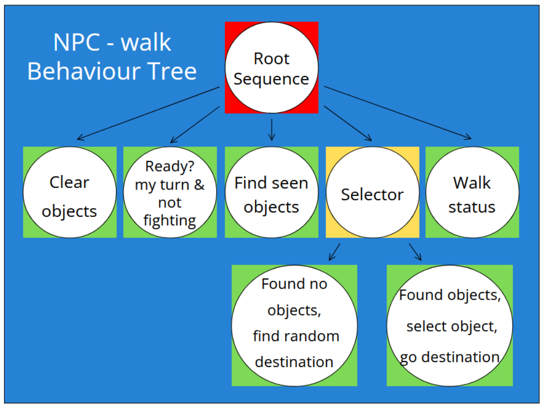
NPC - walk
Behaviour Tree
Root Sequence
Clear objects
Ready? my turn & not fighting
Find seen objects
Selector
Walk status
Found no objects, find random destination
Found objects, select object, go destination

Fig. 2. Base behaviour tree.

When in a group, members will share collected food and gold, but decisions are made by the group leader. Moreover, weapons will only be assigned to the current leader. The members take turns being leader, which decides where to go. When a group participates in a fight, the leader is the only one actively fighting. If the leader dies during a fight, a different agent takes its place and continues the combat. After a successful fight, the leader will change and the group member with the highes HP will take their place. The game starts with 200 NPCs equally spaced, in a grid, and ends when 20% is left; on the hardware used for the study, which took approximately 10 seconds.

A. *Genome*

To allow ME to find different behaviours we have encoded some of the parameters of the BT described in the previous section, which change the priorities of the agent in deciding which actions to take. We designed a genome consisting of 9 genes, each represented as a floating-point number between 0 and 1: **GroupTendency**, **FoodHigh**, **WeaponHigh**, **OppHigh**, **GoldHigh**, **FoodLow**, **WeaponLow**, **OppLow**, and **GoldLow**.

Gene 1 represents how likely the agent is to want to join a group (or have other NPCs join their group). Genes 2 to 5 represent the likelihood of the agent choosing to go towards either food, weapons, an opponent, or gold when the NPC's health is above 50. Genes 6 to 9 represent the same preferences but when the agent's health is below 50.

IV. Methods

The original implementation of ME does not support multi-agent environments, we propose a hybrid implementation of ME (II-B) and more standard genetic algorithms to fit the proposed problem. Our algorithm deviates from standard MEs by having multiple agents simultaneously play the simulation while allowing for interactions between the population's individuals. When a game is completed, the results are used to evaluate the agents, obtaining their features, eventually updating the elite-map, and producing the next generation. The algorithm is similar to standard evolutionary algorithms in the selection strategies for creating new individuals (ranking the current generation and selecting parents through a roulette wheel method), and crossover/mutation operators.

A. *Feature space*

In ME methods, the solutions are placed in a feature-space which is separate from the solution-space in which individuals are purely ranked by performance (as in traditional evolutionary algorithms). We define a three-dimensional feature-space that describes characteristics of different NPC behaviour in our simulation:

- *GroupTendency* was chosen since it was deemed interesting to evolve agents which could both be prone to act alone or work together in a group.
- *SurvivalTime* was chosen since it could be interesting to have agents which would obtain a good score, perhaps by being reckless and shortlived; or building a good score over a longer time by being careful by going for food, weapons, and gold before going for other NPCs. Or by being in a group.
- *TotalDamage* was chosen to have agents which would either obtain a lot of gold through killing, or perhaps gaining a good amount of gold by direct pick-up.

B. *Implementation details*

This section describes the general structure of the evolutionary loop and how our multi-agent ME differs from the vanilla version. At all times 200 agents will be playing. That number was chosen since that was what the available hardware could handle. We decided to add a requirement for agents to be eligible to be placed in the elite-map: individuals must have played some number of games. Vanilla ME does not include such a requirement but, given that our environment is very dynamic, we felt it was important to alleviate the effect of a "lucky run" by averaging the performance (and features) of the agents over multiple games. Individuals that have not yet satisfied the requirement are kept in the population until the criterion is met. Once the agent has played the required number of games, i) the features of the agent are determined, ii) based on the features the appropriate cell in the elites-map is determined, and iii) if the agent has a higher performance than the current solution in the cell, it can replace the previous elite.

The total number of iterations (games) is determined by how many generations the system is run for:

$$iterations = generations * minimum\_number\_of\_games$$

C. *Selection strategy*

When a new population needs to be generated, we have chosen to apply a strategy that includes both individuals in the current population and elites. Since every couple of two

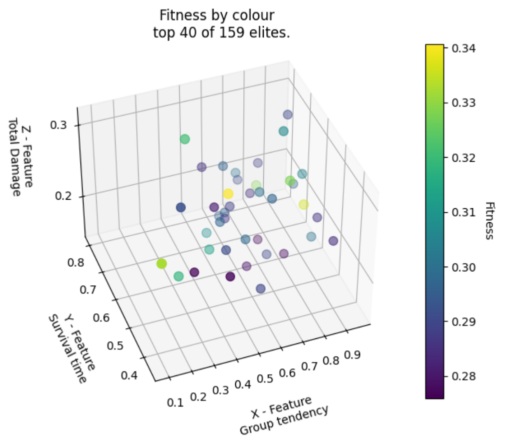

Fig. 3. Fitness from all features. Top 25%. 120 generations, sample-size 20, resolution 10, 2400 iterations.

parents will produce two offspring, 100 map-elites and 100 playing-agents is chosen. The parents for the new generation are chosen as:

- If possible, 50 elites and 50 individuals from the current population will be used.
- If there are not enough elites (such at the start of the evolutionary run) extra playing agents will be chosen from the population as needed.
- Roulette wheel is held in each category, based on rank by fitness. By using rank instead of actual fitness value, it is ensured that individuals with very low or very high fitness do not get an overly high/low chance of being selected.
- Once the pool of parents has been selected, random couples (from both categories) will be selected for crossover.

D. *Crossover and mutation operators*

We have chosen to implement a two-point crossover operator. Each gene in the new offspring will have a chance of mutating. The mutation chance is set to 1% to prevent drastic changes in the population. To encourage that a gene does not mutate too far from the original, Box Muller Transform [8] was used to generate a normalised new gene value around the original value.

V. Results

This section reports on the results of experiments with running the system with various number of generations and number of required games (from now on referred to as *sample-size*). For all the experiments the resolution of the feature-space is set as 10, meaning that the elites can be placed in a 10x10x10 matrix.

The top 25% of obtained elites (with 120 generations and sample-size 20) is visualised in figure 3. Since group-tendency is directly encoded [7] there is a good diversity on the x-feature. Features *survival-time* and *total-damage* are indirectly encoded, and have lower diversity, respectively around 0.4-0.8 and 0.2-0-3.

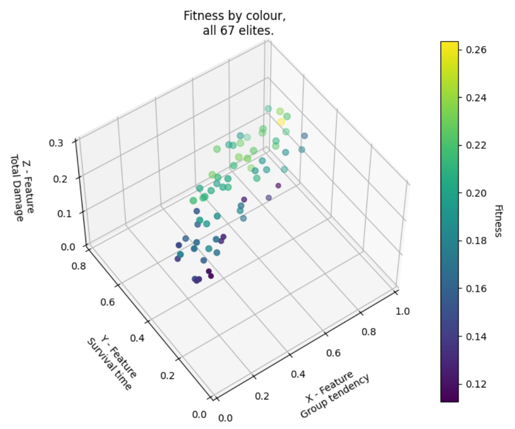

Fig. 4. Fitness from all features, all elites, 100 generations, sample size 100, resolution 10, 10000 iterations.

Since in a game development situation the developer would need to implement NPCs based on their genomes, there isn't much value in the method if indirectly encoded features and performance cannot be predicted to a 'good' degree by the genome. Then, since there is a lot of randomness in the prototype environment, we hypothesize that letting each individual play more iterations will enhance the relationship between genome and indirect features/performance. To test this, another simulation was run with 100 generations, sample size 100, and resolution 10. Figure 4 shows all the developed elites, which amounts to 67. It can be observed that the range of survival-time and total-damage has narrowed further down to approximately 0.5-0.7, and around 0.2.

VI. Discussion

A. Discussion of obtained results

1) 120 generations, sample-size 20: In this simulation 159 elites, and 40 high performing elites were evolved (see section V and figure 3). The diversity in features *group-tendency* and *survival-time* was good; however, since it is hard for the NPCs not to get killed since they cannot flee, we hypothesize that high total-damage is rather based on luck from one game to another.

2) 10000 iterations, sample-size 100, 100 generations: This simulation resulted in an even more narrow *total-damage* diversity, due to the limitations mentioned in section VI-A1. However, prediction of fitness from genome had significantly improved with increased *sample-size*, to 40%. Features *survival-time* and *total-damage* alone were predicted by 70% and 9.4%.

B. Strengths and weaknesses of the test environment

Strong sides of the test environment are that the rules are simple, the NPC decisions are limited to selecting a destination, based on their genome and the objects they currently see. A clear limitation is that the rules are too simple for the environment to resemble a game with human players. As mentioned, the method developed could be used for any game which uses NPCs, although there is a challenge in selecting appropriate features for a particular game.

C. Strengths and weaknesses of multi-agent Map-Elites

The presented method allows to harness Map-Elites's quality of developing multiple well performing diverse solutions to discover multiple agent behaviours simultaneously. Moreover, it allows the agents to interact with each other as part of the evolutionary process.

We believe that being able to customise the feature-space would make the method interesting to game developers since they would be able to adapt it to their needs. With the addition of a user-friendly interface to define genome, features, fitness, and data collection, a developer would be in full control.

VII. Conclusions

We present a multi-agent Map-Elites algorithm to cultivate diverse agents within a feature space. The long-term aim is to establish a framework for producing NPCs for more complex games that include human players. We developed a multi-agent test environment; however, after several sessions and numerous iterations, it was observed that the options available for the NPCs in the game were too limited. The survival aspect of the game was found to be overly challenging and random, hindering the true evolution of diverse behaviour.

Results indicate that around 100 games (sample-size) were needed to represent each agent with reasonable accuracy in the elite-map. It was found that 40% of fitness variation, based on the agent genome, could be predicted. Further development would be focused on enhancing the prediction, in turn improving the usability of the product.

In conclusion, while the prototype game requires adjustments to reduce randomness, the successful implementation of the multi-agent Map-Elites algorithm demonstrates its potential in evolving diverse Non-Player Character behaviours, paving the way for more engaging and dynamic gaming experiences.

References

[1] J. K. Pugh, L. B. Soros, and K. O. Stanley, "Quality diversity: A new frontier for evolutionary computation," *Frontiers in Robotics and AI*, vol. 3, p. 40, 2016.

[2] J. Nordmoen, F. Veenstra, K. O. Ellefsen, and K. Glette, "Quality and diversity in evolutionary modular robotics," in *2020 IEEE Symposium Series on Computational Intelligence (SSCI)*, pp. 2109–2116, IEEE, 2020.

[3] A. Cully and Y. Demiris, "Quality and diversity optimization: A unifying modular framework," *IEEE Transactions on Evolutionary Computation*, vol. 22, no. 2, pp. 245–259, 2017.

[4] D. Gravina, A. Khalifa, A. Liapis, J. Togelius, and G. N. Yannakakis, "Procedural content generation through quality diversity," in *2019 IEEE Conference on Games (CoG)*, pp. 1–8, IEEE, 2019.

[5] M. C. Fontaine, S. Lee, L. B. Soros, F. de Mesentier Silva, J. Togelius, and A. K. Hoover, "Mapping hearthstone deck spaces through map-elites with sliding boundaries," in *Proceedings of The Genetic and Evolutionary Computation Conference*, pp. 161–169, 2019.

[6] A. Alvarez, S. Dahlskog, J. Font, and J. Togelius, "Empowering quality diversity in dungeon design with interactive constrained map-elites," in *2019 IEEE Conference on Games (CoG)*, pp. 1–8, IEEE, 2019.

[7] J.-B. Mouret and J. Clune, "Illuminating search spaces by mapping elites," *arXiv preprint arXiv:1504.04909*, 2015.

[8] G. E. Box and M. E. Muller, "A note on the generation of random normal deviates," *The annals of mathematical statistics*, vol. 29, no. 2, pp. 610–611, 1958.