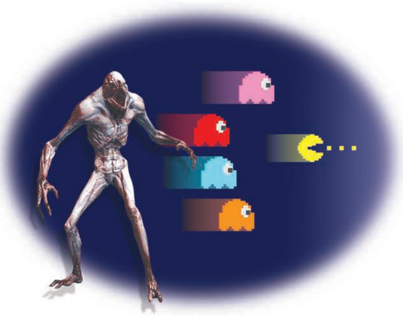

CREATURE—COURTESY OF KONAMI.
PAC-MAN—IMAGE LICENSED BY INGRAM PUBLISHING

Playing with evolution

Raluca D. Gaina

ROBOTS—©SHUTTERSTOCK.COM/BLUE PLANET STUDIO

Evolution: a process we've all been through but have yet to fully understand. You started off as a tiny little creature who couldn't talk, couldn't walk, and didn't even have feet. There were many other possibilities, too, but, somehow, nature decided you were the fittest. You were given the chance to live and do your thing in the world, while millions of others were discarded at the concept stage. Genes from your parents were selected, combined, and modified in an attempt to make you best adapted to your environment. This whole process is fascinating: how does nature decide what features give you the chance to be the best that you can be?

That's not a question to be answered here. Instead, this article explores some of the ways in which this process was coded into programs that can do a variety of cool things in games. These programs fall under the name of *evolutionary algorithms*. It is not only that the games themselves evolve over time due to new computational resources and the diversity of people gaining access to tools to bring their creations into existence and to the wider public—we can also evolve content within the games themselves.

How do evolutionary algorithms work?

While there are many ways in which these algorithms can be implemented, a typical scenario works as follows: we consider populations (or groups) of individuals. An individual

Digital Object Identifier 10.1109/MPOT.2021.3124305
Date of current version: 14 March 2022

(or what would be one person for humans) is going to be one solution considered for whichever problem we are addressing: a player who wins a game, a sequence of actions to execute that lead to winning, a designer who creates interesting levels for games, and so on. Individuals are represented by a genotype and a phenotype. The genotype is the genetic encoding, or the genes, that define the unique properties of the individual: for example, how risky a strategy should be or how far into the future a player should imagine the game playing out. The phenotype, then, is the manifestation of the genotype in the context of the problem: the behavior of the player or levels created by a designer.

To create an evolutionary process, we start with a randomly generated population of individuals: the genes for each of them are selected at random from the pool of all possible genes. We then evaluate each individual's fitness, or how good they are at solving the given problem: how often a player wins at the game or how interesting the levels created are. Each individual gets a score based on this evaluation, which is its fitness score. We can then take the fittest individuals (or those with the highest scores), combine their genes [a process called *crossover* (Fig. 1)], and change some of them (by randomly replacing some with others from the gene pool, known as *mutation*) to create more individuals called *offspring*.

After evaluating the offspring as well, we choose to keep only the best ones in our population; this selection completes the first generation, resulting in a population of individuals that are similar or better than before. Then we repeat this process for several generations. How long? This depends on the application. However, at some point, we stop the process and choose the best individual in the final population to actually solve our problem to the extent of that individual's ability. Generally, these are the algorithms used for optimization problems: they find a solution that best addresses a given problem (Fig. 2).

Evolutionary algorithms playing games

A big question remains: what do evolutionary algorithms actually do in games? They can play games; design levels; optimize games, levels, players, and strategies; and much more.

Rolling horizon evolutionary algorithms (RHEAs) are an example of a class of algorithms that use the ideas of this evolutionary process to play games (Fig. 3). What they evolve is action plans: at each step in the game, they choose a sequence of actions of a particular length (which defines how far into the future the algorithm can see) that brings the player closest to winning the game.

Here, individuals are sequences of actions (in practical terms, a list of numbers, where each number corresponds to an action). In other words, they imagine, at every game step, several possible future scenarios based on what they might try to do (given by the evolutionary algorithm) and pick a plan based on the result from these simulations.

Therefore, to evaluate an individual, a model of the game is used to simulate the effect of the action sequence. Just like human intuition, this is often inaccurate (or plain wrong) in games with elements of chance or randomness as well as those where a player does not know everything about the world (for example, when parts of the map are covered in the fog of war or if an opponent holds a hand of cards that is hidden from the player).

Individuals can only find this out when they actually follow their plan in the real game, although research has found that even inaccurate or smaller, more abstract world models

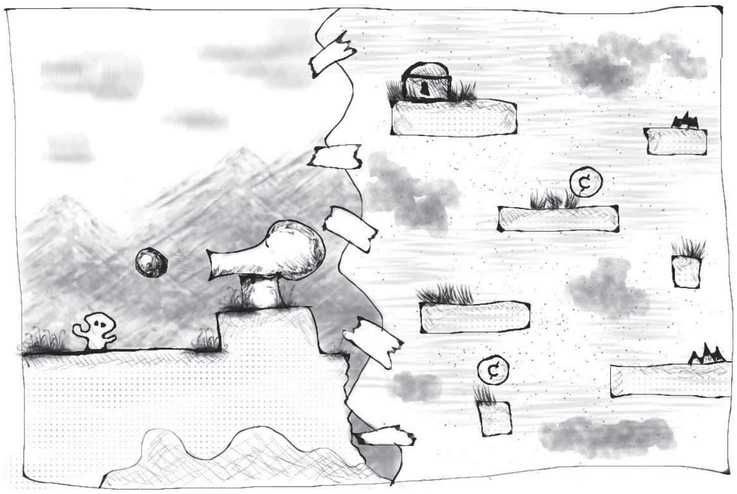

FIG1 Crossover for game levels.

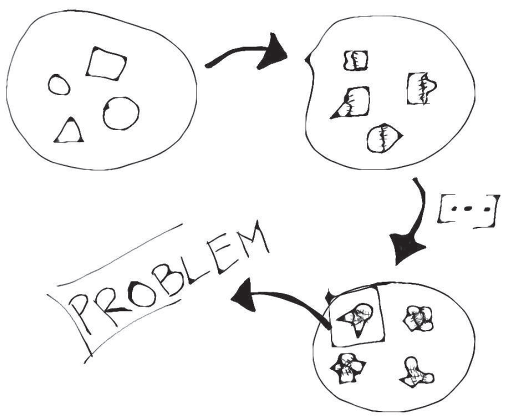

FIG2 An evolutionary process.

As long as you can define gene pools, genotypes, and phenotypes in a way that a computer understands them, you can apply evolutionary algorithms to bring some of the power of evolution shown by nature to your problems.

could be good enough to approximate what may be the best thing to do in different situations. As with any problem, a "good enough" solution is often preferred to the alternative of spending more resources to possibly find something better.

The game state reached after following each action plan is given a value based on the likelihood that the player will win from there (often, the game score is a good indication of this), and this value becomes the fitness of the individual. At the end of the process—for example, when the time budget given has been used up—the algorithm chooses the first action of the best plan found to execute in the game, and then it repeats the process in the next game step.

RHEA has been used for many games, and it is an area increasingly more researched in recent years due to its ability to adapt to many unknown situations through the power of evolution and its surprising strength in situations where the player receives little information about progress in the game, such as puzzle games. Many modifications to the basic method were tried, and many parameters controlling the decision-making process were adjusted for different environments, which include video games, such as *Space Invaders*, or even complex modern strategy board games, such as *Terraforming Mars*.

The algorithm works better when preserving action plans evolved from one game step to the next instead of starting from scratch each time or when it dynamically modifies the length of the action plans evolved depending on the diversity of information in the environment. All modifications produce players able to perform the best in at least one game, but no single configuration of the algorithm is able to solve all given problems. We are still not yet at the stage of a generally applicable adaptive algorithm.

Evolutionary algorithms evolving other algorithms

If we frame the problem differently, we can add another layer on top: RHEA itself has a series of control parameters that indicate how the algorithm behaves, such as the number of individuals in a population, length of an action sequence, or mutation rate. These parameters can be seen as genes themselves—so we can use an evolutionary algorithm to evolve RHEAs, which, in turn, evolve action sequences (Fig. 4).

This has been tried in two ways: optimizing parameters 1) so that RHEA obtains the highest win rate by playing a game several times and trying out different configurations and 2) to maximize the improvement in the quality of individuals in the population from one generation to the next. While these methods do not always succeed in producing better results, they can be used to

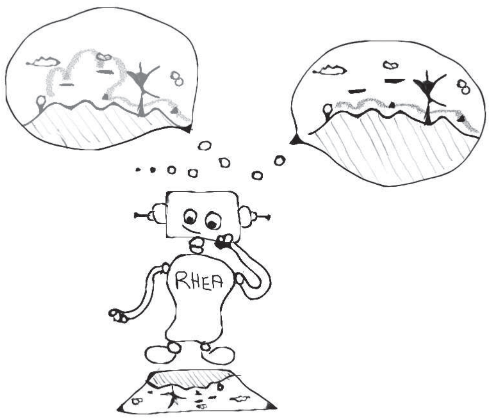

FIG3 RHEA simulates multiple trajectories through a level to make decisions about what to do in each situation.

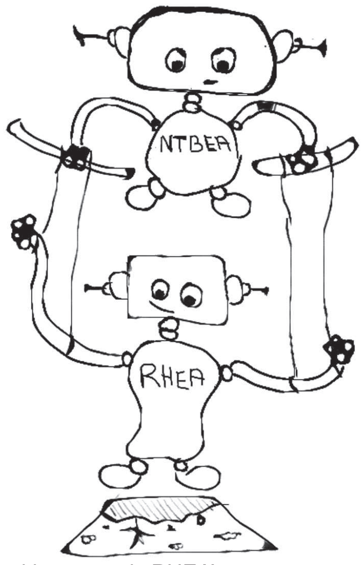

FIG4 An optimization algorithm controls RHEA's parameters while RHEA plays games. NTBEA: N-Tuple Bandit Evolutionary Algorithm.

solve some very difficult problems without any domain information. We could, perhaps, continue adding such layers of algorithms, which may be one path toward fulfilling the great quest for general artificial intelligence (AI).

As with any problem, a "good enough" solution is often preferred to the alternative of spending more resources to possibly find something better.

Evolutionary algorithms evolving games

We can also look at another side of the problem: to play a game, we first need a game. What if AI created the game before playing it? Typical applications of evolutionary algorithms do not go quite that far, although some have tried and even succeeded in creating new board games: *Yavalath* is a classic example here.

However, they are more often used to create pieces of games, such as game levels: we can encode an individual as a list of numbers again, but, this time, each gene represents a tile on the screen. Combining two different levels would then take chunks from each to create a new level, and mutations would lead to changing some tiles to other types. The only thing left is to evaluate the levels and give each a fitness score—this is a larger problem, which can be addressed in several ways:

- An algorithm could extract features from the level (for example, the distribution of ground tiles or number of enemies present) and assign a fitness score depending on the distance to ideal target values.
- Several automatic game players could play the level, and statistics recorded about their game play could be used to determine a level's score (for example, did the more proficient player win more than the silly player?).
- A human could be brought into the mix to assess levels subjectively and use their expertise and wishes for the final design to guide evolution in the desired direction.

The game *Super Mario Bros.* is a great example here, where extensive research has shown it is possible to generate whole new levels with a similar look and feel to existing human-designed levels or even generate these on the fly, just in front of the player, during play itself to maximize a player's enjoyment.

These methods could also be combined. Recent work shows this in practice for evolving structures in *Minecraft*. In this application, you start with a base structure and work together with an evolutionary algorithm to improve it by iteratively choosing one of the options the algorithm gives you and then letting it create more offspring from there.

Generally, as long as you can define gene pools, genotypes, and phenotypes in a way that a computer understands them, you can apply evolutionary algorithms to bring some of the power of evolution shown by nature to your problems and simply let potential solutions evolve until they're good enough. If only this worked for all of life's problems! Maybe soon.

Read more about it

- R. D. Gaina, "Rolling horizon evolutionary algorithms for general video game playing," Ph.D. dissertation, Queen Mary Univ. London, UK, Apr. 2021.
- R. D. Gaina, M. Balla, A. Dockhorn, R. Montoliu, and D. Perez-Liebana, "TAG: A tabletop games framework," GitHub, 2020. [Online]. Available: https://github.com/GAIGResearch/TabletopGames
- A. Khalifa, "Current framework version: 0.8.0," GitHub, 2006. [Online]. Available: https://github.com/amidos2006/Mario-AI-Framework
- T. Shu, J. Liu, and G. N. Yannakakis, "Experience-driven PCG via reinforcement learning: A super Mario bros study," 2021, arXiv: 2106.15877.
- D. Grbic, R. B. Palm, E. Najarro, C. Glanois, and S. Risi, "EvoCraft: A new challenge for open-endedness," in *Proc. EvoApplications*, May 2021, pp. 325–340, doi: 10.1007/978-3-030-72699-7_21.

About the author

Raluca D. Gaina (r.d.gaina@qmul.ac.uk) earned her B.Sc. and M.Sc. degrees in computer games at the University of Essex in 2015 and 2016, respectively. She is a lecturer in game artificial intelligence (AI) at Queen Mary University of London, London, E1 4NS, U.K., where she earned her Ph.D. degree in intelligent games and games intelligence in April 2021 (in the area of rolling horizon evolution in general video game playing). She did a three-month internship at Microsoft Research Cambridge and was involved in the organization of various competitions and committees related to game AI. Her research interests include general video game playing AI, evolutionary algorithms, and tabletop games.