Evolving a Dota 2 bot: Illuminating search in CGP and NEAT

Paul Templier*‡, Lucas Hervier*‡, Dennis Wilson†
*Institut Supérieur de l'Aéronautique et de l'Espace (ISAE-SUPAERO), Université de Toulouse, 31055 Toulouse, FRANCE
Emails: {paul.templier, lucas.hervier}@student.isae-supaero.fr
†Institut Supérieur de l'Aéronautique et de l'Espace (ISAE-SUPAERO), Université de Toulouse, 31055 Toulouse, FRANCE
Email: dennis.wilson@isae-supaero.fr
‡*These authors contributed equally to this work*

***Abstract*—In this work we present an evolution-based approach applied to Dota 2 in the Project Breezy challenge [4]. The goal of this project is to train an agent to play a 1v1 Midlane match against the game's bots of varrying difficulties, with both sides playing Shadow Fiend.**
The approach we implemented relies on the MAP-elites algorithm assisted with a neural-based simulator of the game to increase behavior diversity and reduce computation load, using CGP agents or NEAT networks as individuals.

I. An evolutionnary approach

Finding the best policy to win a Dota2 1v1 against a bot can be assimilated to a Reinforcement Learning (RL) problem. To adress this issue we chose to turn to evolutionary algorithms which have shown promising results on the Atari benchmark (1).

Game	Human	Double	DQN	Prioritized	A3C:FF	HyperNEAT	CGP
Asteroids	13157	1193.2	2035.4	1654	4474.5	1694	9412
Defender		27510	33996	21093.5	56533	14620	993010
Gravitar	2672	200.5	297	218	303.5	370	2350
JamesBond	406.7	573	835.5	3511.5	541	5660	6130
Kangaroo	3035	11204	10334	10241	94	800	1400
Krull	2395	6796.1	8051.6	7406.5	5560	12601.4	9086.8
Ms. Pacman	15693	1241.3	2250.6	1824.6	653.7	3408	2568
Private Eye	69571	-575.5	292.6	179	206.9	10747.4	12702.2
Skiing		-11490.4	-11928	-10852.8	-10911.1	-7983.6	-9011
Solaris		810	1768.4	2238.2	1956	160	8324
YarsRevenge		6270.6	25976.5	5965.1	7157.5	24096.4	28838.2

Fig. 1. Game Scores Comparisons Atari Games[a]

[a]extract from Evolving simple programs for playing Atari games

In this project we chose to train both CGP Individuals [1] and NEAT Individuals [5], as defined in the respective papers, in a MAP-Elites algorithm. They were implemented in Julia using the **Cambrian.jl** framework, **CartesianGeneticProgramming.jl** for the CGP agent and a custom NEAT agent developed at **NEAT.jl** for this challenge.

As we believed the best way to find interesting individuals, either with NEAT or with CGP, was to explore the behavorial space defined in Section III, we based the exploration on the **Illuminating search spaces by mapping elites** [2] paper. However, a limiting factor in training evolutionary algorithms for Dota appeared to be evaluation time.

II. Dealing with evaluation cost

We quickly realized that one of the main difficulty of this challenge was to deal with the evaluation cost of the fitness. Indeed, in order to evaluate an agent we need him to play a full DOTA2 game which can take up to several minutes in real time, even with a tenfold speeding. Since Evolutionnary Algorithms (EAs) require a large number of function evaluations, we had to find a way to cope with this issue.

A. Early Stopping

A simple but efficient way of reducing the evaluation time was to use an Early Stopping mechanism. We decided to stop the evaluation of an agent if he did not kill a creep after 5min of DOTA time. This functionality reduced the time loss due to static individual, and the evaluation length of "coward individuals" avoiding fights.

B. Neural-base Simulator

Since an evaluation is time consuming, we developped a **Dota Simulator** to predict game state evolutions from previous state and chosen action, using a neural network based on the UNet [3] architecture which we trained on random play. It was used in the offsprings generation phase:

- offsprings are generated for 10 times the size of population
- 100 game steps are simulated for each offspring, and the actions stored
- the actions taken by each individual are used to select the *population size* most unique offsprings based on behavior
- those individuals are added to the new population and evaluated in the real game

By doing so we are seeing much more mutation and/or crossover. Moreover, taking the individuals the more distant from each other increases our chances to cover the behavior space.

III. Problem Modeling

A. Fitness Function

In order to win a 1v1, a player needs to kill the opponent twice or to take his first mid tower. Consequently, we took

into account the number of times we killed the opponent champion and the opponent tower health in the individual evaluation. Moreover, to succeed in one of those two tasks one would also need good laning skills. Therefore, we also took into account the amount of gold collected (*net worth*), the number of last hits, and the number of denies. We also punished being killed and behaviors causing an early stopping. The final fitness function is hence as follows:

$$\begin{aligned} fitness = netWorth + 100 * lastHits + 100 * denies \\ + 2000 * ratioTowerHealth + 1000 * nbKill \\ - 250 * nbDeath - 500 * earlyStop \end{aligned}$$

B. Behavior Space

As mentionned in Section I our goal was to observe individuals with highly diverse behaviors to increase our chances of finding interesting ones. Therefore, we needed a characterization of the behavior space to differenciate playstyles. We described the behavior in a discretized 2-dimensional space featuring the *total damage made to the opponent champion* and the *percentage of the opponent tower health taken* as axes, since the objective is either to kill the opponent champion or to take his tower.

C. Procedure

The approach we present relies on 3 main phases: initialization, step, and run. The implementation of these algorithms is available in the **Project Breezy - SUPAERO** repository. The first phase aims at creating the map and a first population, as seen in **Algorithm 1**.

Algorithm 1: Initialization

Input: $IndType$ the type of individual, cfg file containing evolution parameters
begin
  # we create a population of CGP or Neat Individual
  $population = InitPopulation(IndType, cfg)$
  # we create an empty Map of Elites
  $mapelites = MapElites(feature\_dim, grid\_mesh)$
  # we evaluate all individual
  $evaluate!(population)$
  # we add all the new individual to the map
  $add\_to\_map!(population, mapelites)$
  return $population, mapelites$

The fitness of an Individual is part of its features and is initialized to $-\infty$ before evaluation. The add_to_map function adds an individual to a position determined by the damage he made to the tower and the opponent champion only if no other individual is already register at this position, or if the individual we want to add has a better fitness

Algorithm 2 is the procedure followed to create one generation from another. The $simulate!$ and $select\_diverse!$ functions used the **Dota Simulator** and allowed us select a more diverse population of size $population\_size$ from the $huge\_new\_pop$ list.

Algorithm 2: Step Function

Input: mutation function, crossover function, mapelites
begin
  # we generate a huge amount of offsprings
  $huge\_new\_pop = []$
  for i *in* $1 : 10 * pop\_size$ **do**
    # $select\_random$ return a random individual
    # from the map
    $p1 = select\_random(mapelites)$
    $p2 = select\_random(mapelites)$
    $child = deepcopy(p1)$
    if *mutation happen* **then**
      $mutation(child)$
    if *crossover happen* **then**
      $crossover(child, p2)$
    $push!(huge\_new\_pop, child)$
  $simulate!(huge\_new\_pop)$
  $new\_pop = select\_diverse(huge\_new\_pop)$
  $evaluate!(new\_pop)$
  $add\_to\_map!(new\_pop, mapelites)$

Finally, **Algorithm 3** is a wrapper of the 2 previous algorithms:

Algorithm 3: Run Function

Input: cfg config file, mutation function, crossover function
begin
  $population, mapelites = Initialization(IndType, cfg)$
  for i *in* $1 : cfg["n\_gen"]$ **do**
    $Step(mutation, crossover, mapelites)$

A run of this algorithm does not return a single best individual but a mapping of the behavior space filled with elites in each discretized area, hence yielding multiple potentially interesting agents with diverse playstyles.

REFERENCES

[1] DG Wilson et al. Evolving simple programs for playing atari games. 2018.
[2] Jean-Baptiste Mouret and Jeff Clune. Illuminating search spaces by mapping elitess. 2015.
[3] Olaf Ronneberger, Philipp Fischer, and Thomas Brox. U-net: Convolutional networks for biomedical image segmentation, 2015.
[4] Robert J. Smith and Malcolm I. Heywood. Evolving dota 2 shadow fiend bots using genetic programming with external memory. In *Proceedings of the Genetic and Evolutionary Computation Conference*, GECCO '19, page 179–187, New York, NY, USA, 2019. Association for Computing Machinery.
[5] Kenneth O. Stanley and Risto Miikkulainen. Evolving neural networks through augmenting topologies. volume 10, pages 99–127, 2002.