SimpAI: Evolutionary Heuristics for the ColorShapeLinks Board Game Competition

4 authors:

Pedro M. A. Fernandes
Universidade Lusófona de Humanidades e Tecnologias
1 PUBLICATION   **2** CITATIONS

SEE PROFILE

Hugo Feliciano
Universidade Lusófona de Humanidades e Tecnologias
1 PUBLICATION   **2** CITATIONS

SEE PROFILE

Pedro M A Inácio
Universidade Lusófona de Humanidades e Tecnologias
1 PUBLICATION   **2** CITATIONS

SEE PROFILE

Nuno Fachada
Universidade Lusófona de Humanidades e Tecnologias
55 PUBLICATIONS   **245** CITATIONS

SEE PROFILE

Some of the authors of this publication are also working on these related projects:

Project   Image Analysis of Lipid Droplets&Allies, Techniques and Finds View project

Project   Lusófona Videogames View project

SimpAI: Evolutionary Heuristics for the *ColorShapeLinks* Board Game Competition*

Pedro M. A. Fernandes[†,1], Pedro M. A. Inácio[†,1], Hugo Feliciano[1], and Nuno Fachada[2,1]

[1] School of Communication, Arts and Information Technologies, Lusófona University, Lisbon, Portugal
`{a21803791, a21802050,a21805809}@alunos.ulht.pt`
[2] HEI-Lab - Digital Human-Environment and Interaction Lab, Lusófona University, Lisbon, Portugal
`nuno.fachada@ulusofona.pt`

[†]These authors contributed equally to this work.

Abstract. We present SimpAI, an AI agent created for the *ColorShape-Links* competition, based on an arbitrarily sized version of the Simplexity board game. The agent uses a highly efficient parallelized Minimax-type search, with an heuristic function composed of several partial heuristics, the balance of which was optimized with an evolutionary algorithm. SimpAI was the runner-up in the competition's most challenging session, which required an AI agent with good adaptation capabilities.

Keywords: Board games · Artificial intelligence · Evolutionary heuristics · Simplexity · ColorShapeLinks

1 Introduction

In this paper, we present SimpAI, an artificial intelligence (AI) agent created for the *ColorShapeLinks* board game competition [3]. This competition is based on an arbitrarily sized version of the Simplexity board game [1]. In this game, the board is a vertically placed grid and pieces fall with gravity. In a similar fashion to Connect-4, the first player to place n pieces of the same type in a sequence, wins. Pieces are defined by color, white or red, and shape, round or square. The first player wins with round or white pieces, while the second player wins with square or red pieces. Shape has priority over color as a winning condition, and while players only play with pieces of their color, they can play with pieces of both shapes, making the game more interesting and complex.

In a standard Simplexity game, the board has six rows and seven columns, and victory can be achieved with a sequence of four pieces of the same shape or color, either vertically, horizontally or diagonally. Fig. 1 shows the possible

---

(a) Color victory for white player.   (b) Shape victory for white player.

(c) Color victory for red player.   (d) Shape victory for red player.
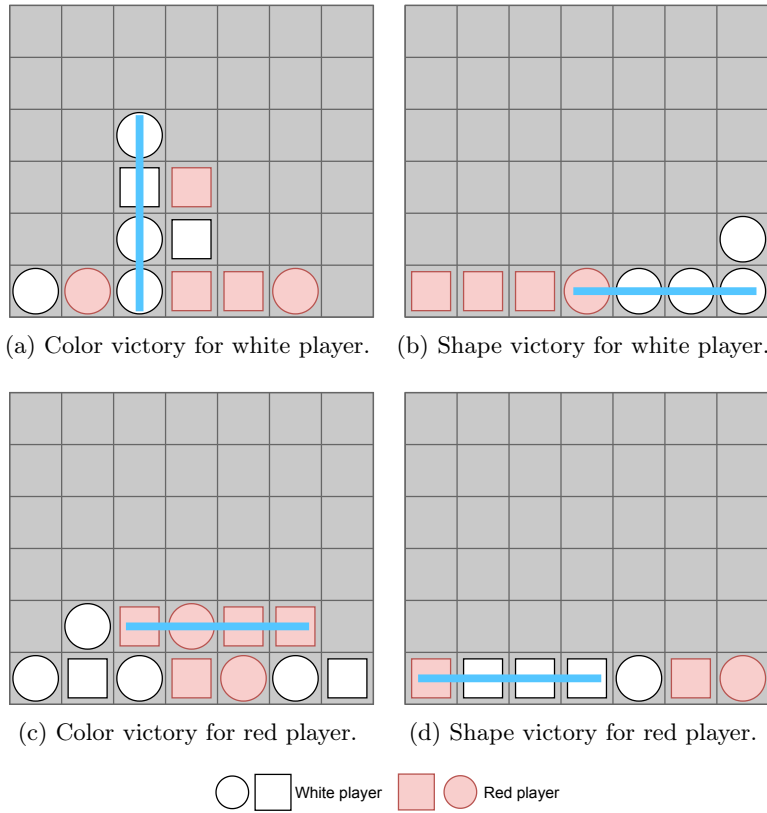
White player   Red player

Fig. 1: Victory conditions for the Simplexity board game.

victory conditions for both players in a standard Simplexity game, i.e., either by color or by shape, with the latter having priority as shown in Fig. 1b and Fig. 1d.

The SimpAI agent was developed as a learning and exploration project in the context of an AI course unit at Lusófona University's Bachelor in Videogames degree [4]. It searches for the best solutions in the game board using a classical Minimax-type search approach, with a number of optimizations to search as deep and wide as possible. The search is guided by a composition of five heuristics, the weights of which were optimized using an evolutionary algorithm. The agent finished the competition in second place in the *Unknown Track*, for which the board configuration and time to play were only announced after the competition deadline.

This paper is organized as follows. In Section 2, we review board game AI techniques, as well as two competing agents in the first edition of the *Color-ShapeLinks* competition. In Section 3, we discuss in detail the proposed agent,

namely the optimized search algorithm, the implemented heuristics and how we performed heuristic weight optimization using evolutionary algorithms. In Section 4 we describe the results obtained using this approach and how the agent fared in the two competition tracks. A discussion of these results takes place in Section 5. Section 6 closes the paper, discussing potential improvements and offering some conclusions.

2   Background

The classical search approach in board game AI is the Minimax algorithm [8,11]. This algorithm performs a depth-first search of the game tree down to a predefined search depth, bubbling up board evaluations obtained at maximum depth with a given heuristic function in a recursive fashion. Minimax evaluates game states at each depth from the AI's perspective, while assuming the adversary will also choose the best move for himself. As such, Minimax *maximizes* board evaluations when the AI is playing, *minimizing* them in the opponent's turn. Minimax has been optimized and improved through the years. Some of these optimizations, discussed for example in reference [8], are summarized in the following paragraphs.

Negamax is one of the most basic improvements to Minimax, evaluating boards from the perspective of who is playing at a given depth. The code ends up being simpler and slightly more efficient than a pure Minimax, since the algorithm only performs maximizations. Conversely, Alpha-beta pruning is a crucial optimization for Minimax (or Negamax), making use of the upper and lower bound evaluations returned by the heuristic function – the search window – in order to determine if a branch from a specific move is worth exploring further, pruning it in case its evaluation crosses these limits.

Aspiration search takes this idea further, attempting to narrow the search window in order to prune more branches, speeding up the algorithm. It accomplishes this by calling the algorithm with a range based on the previous search, the lower bound being the subtraction of a window size to the previous result and the upper bound being the addition of the window size to the previous result.

The Negascout optimization performs a scout test by fully examining the first move of each board position with a wide search window, using the resulting score to narrow the search window for the following moves. If all these following moves fail, the algorithm conducts the search again with a full width search window. From the initial move with a wide search window, the algorithm derives an approximation for the window sizes for successive moves, while also pruning large numbers of branches.

Move ordering works in tandem with Alpha-Beta pruning, ordering the possible moves at each depth by their heuristic score, from highest to lowest, so the algorithm evaluates the most likely best moves first, increasing the chances of pruning potentially worse branches and therefore making the algorithm perform faster and explore further.

Iterative deepening is an optimization which works well with a time limit. It searches with incremental depths, and keeps going while it still has time to keep searching. It essentially performs depth-first search in a breadth-first fashion, allowing the AI to use as much time as it can without going over a preestablished time limit.

Zobrist hashing makes use of an hash table and a collective of various integer numbers for each position of the board, all different, that are later used by the AI to convert a given board state into a rapidly storable and retrievable numerical identifier. These values are stored in a hash table along with the heuristic score for the respective board states, thus avoiding heuristic reevaluation of the latter.

The state of the art in board game AI can be considered a combination of Monte Carlo Tree Search (MCTS) and deep reinforcement learning [8,6,11]. MCTS works by simulating as many random play outs as possible from a given board state while there is available thinking time, selecting a move based on the amount of wins, ties and losses for each node. In turn, deep reinforcement learning combines artificial neural networks with reinforcement learning (e.g., Q-Learning), enabling the AI to learn the best possible moves in any given board state.

Concerning the first edition of the *ColorShapeLinks* competition, the winners of each track used a number of techniques discussed in the previous paragraphs. The ThunderAI agent won the *Base Track* using MCTS with a custom board implementation. In turn, the first place in the *Unknown Track* was claimed by SureAI, a very well tuned Negamax-based agent.

3   Methods

The SimpAI agent was developed in C# (.NET Standard 2.0 [10]), a requirement for the competition as the agents need to run both in the Unity game engine [9] and in the console. The implementation is divided into two different parts, which work together to form SimpAI:

1. The search algorithm, used to search for promising future moves in the time it has available to think, discussed in Subsection 3.1.
2. The heuristic, used to classify future board states according to their strategic value, thus guiding the search algorithm towards finding the best move. The heuristic is presented in Subsection 3.2.

In practice, the heuristic is a combination of several partial heuristics. These are weighted in order to give the final heuristic value for each board state. The optimized weights were obtained using an evolutionary algorithm, as discussed in Subsection 3.3.

3.1   Search

The search is conducted through a Negamax algorithm, with a number of optimizations, namely alpha-beta pruning, move ordering, iterative deepening and zobrist hashing, all of which were discussed in Section 2.

The search was also parallelized, allowing several branches to be evaluated at the same time by distributing the workload by the available CPU cores. In practice, this was achieved with the `Parallel.Invoke()` method in C#, which internally optimizes the number of available worker threads according to the number of hardware threads. Since the Minimax family of search algorithms works recursively, and recursive algorithms are difficult to parallelize effectively [2], each possible move at the first depth level is processed iteratively in the main thread. Then, each corresponding Minimax (Negamax) subtree, from the second depth level and onwards, is recursively processed in one of the available worker threads, as shown in Fig. 2. When all the threads finish their work, results for each subtree are compared back in the main thread, and the branch with the best heuristic score determines the move to perform.
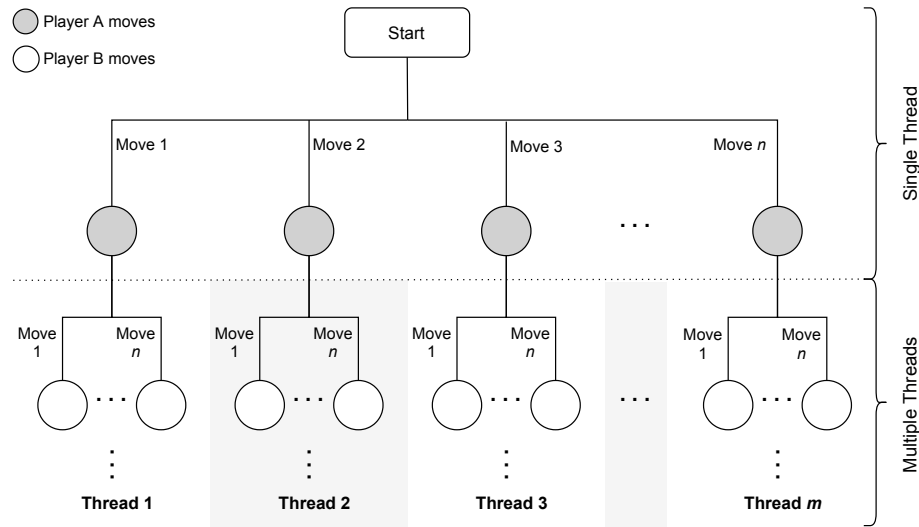

Fig. 2: Parallelization strategy for SimpAI. The first level of depth is iteratively processed in a single thread. From level 2 onwards, each subtree is recursively processed in one of m available threads. For simplification, it is assumed that there are always n moves to be performed at each depth level. Player A represents the player about to play and Player B the player that plays afterwards.

3.2   Heuristics

The heuristic value for each board state is determined by the weighted sum of five partial heuristics, discussed in the following paragraphs.

HorizCenterHeuristic considers the overlapping area between a winning sequence starting from the left side of the board with another from the right side

as a more valuable location for a player's piece. Pieces included in this overlapping area, shown in orange in Fig. 3 for a board with Simplexity's standard dimensions, are able to be a part of a sequence expanding to their right and another expanding to their left. Positions outside this interval are too close to the edge of the board to allow such sequences; therefore, as their distance to the overlap interval increases, the less valuable these positions become according to *HorizCenterHeuristic*.
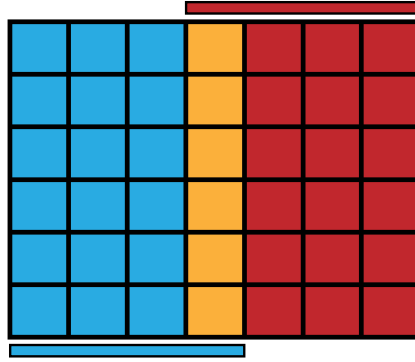

Fig. 3: *HorizCenterHeuristic* on a board with Simplexity's standard dimensions. Blue columns show positions where a horizontal sequence starting from the left side of the board can form. Red columns likewise for sequences starting from the right side of the board. Orange positions indicate where these sequences overlap.

VertCenterHeuristic is the vertical counterpart of *HorizCenterHeuristic*, as shown in Fig. 4.

VertDiscHeuristic considers the area above the number of rows needed to form a vertical winning sequence starting at the bottom of the board as a poor location to place pieces. As shown in Fig. 5, positions above this threshold, displayed in grey, are largely out of reach in an early game, with a winning sequence containing these top spaces requiring a large amount of pieces below them. These are only likely to be part of the winning sequence if the game reaches a very advanced stage where only this area is available.

BuildFromAfarHeuristic considers that placing pieces far away from each other while still at a distance that would allow for a winning sequence between them is a valuable move. In the sequence portrayed in Fig. 6, an agent following this heuristic – first to play – places a piece in the center of the board. Once the adversary places its first piece, the agent answers with a piece in a position at a
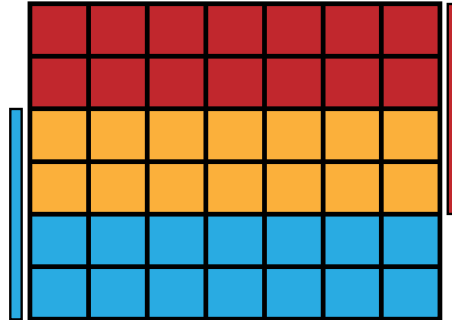
Fig. 4: *VertCenterHeuristic* on a board with Simplexity's standard dimensions. Blue lines show positions where a vertical sequence starting from the bottom of the board can form. Red lines likewise for sequences starting from the top. Orange positions indicate where these sequences overlap.
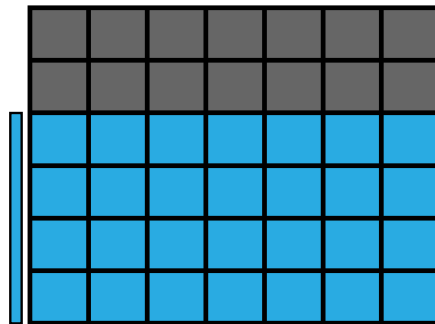

Fig. 5: *VertDiscHeuristic* on a board with Simplexity's standard dimensions. Blue rows can be used to form a vertical winning sequence starting from the bottom of the board. Grey positions are outside this interval.

distance equal to a winning sequence and with an open path to the piece in the middle. If the adversary blocks this potential sequence, the agent places a piece in the opposite position at a winning sequence distance from the center piece. Eventually, the adversary has his pieces nullified (with no way of making an horizontal winning sequence) and the agent has two possible winning sequences.

By placing pieces far from each other while at the same time putting them at a distance that allows for a winning sequence, enables the agent to possibly – according to the board's length – start three different horizontal sequences: i) left to the leftmost piece, ii) between both placed pieces, or, iii) to the right of the rightmost piece. This pressures the adversary to decide which possible sequence to preemptively stop, with this heuristic allowing for the agent to immediately continue another sequence that may, once again, force a similar decision on the adversary, changing its focus from building its sequences to trying to stop the agent's ones.
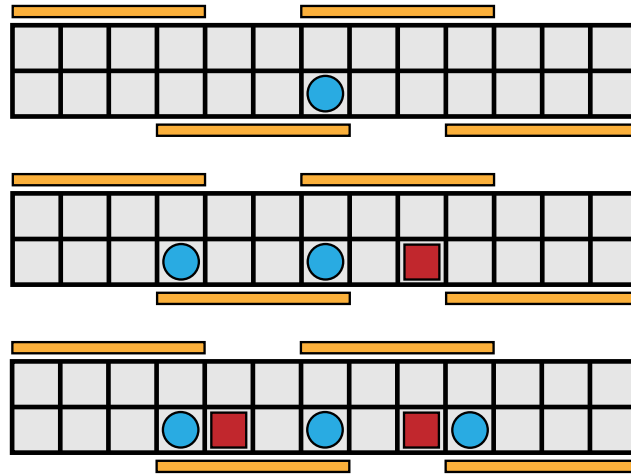

Fig. 6: Three stages (progressing from top to bottom) of an agent using *Build-FromAfarHeuristic* to make decisions on the bottom horizontal slice of a board with 13 columns and a winning sequence of 4 pieces. The orange bars indicate the relevant winning sequences. The agent's pieces appear in blue, while the adversary's appear in red. Both use their specific piece shapes in this example.

DumpFromAfarHeuristic takes into account that placing the adversary's shape pieces in the corner of the map is the safest way of neutralizing them, as these are the easiest positions to isolate, as shown in Fig. 7.

Any winning sequence will require at least one space from the areas calculated by *HorizCenterHeuristic* and/or *VertCenterHeuristic*. Therefore, spaces lose value according to how far they are from these intervals. Taking this into

account, *DumpFromAfarHeuristic* defines these lower value spaces as ideal locations to dispose of adversarial pieces.
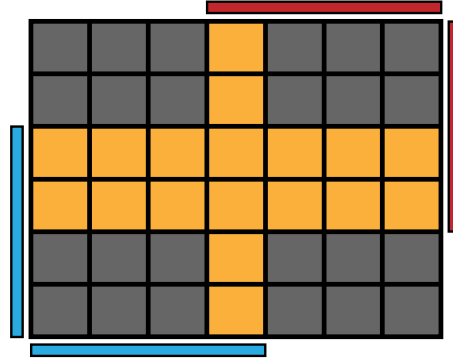

Fig. 7: *DumpFromAfarHeuristic* on a board with Simplexity's standard dimensions. Orange spaces indicate the areas that are part of the *HorizCenterHeuristic* or *VertCenterHeuristic* valuable areas. Grey positions are outside these intervals.

3.3   Heuristic Weight Optimization with Evolutionary Algorithms

The weights of each partial heuristic were obtained with an evolutionary algorithm implemented with the DEAP library [5]. The algorithm population is composed of n individuals representing SimpAI agent instances, with various partial heuristic weights. Each individual is defined by five attributes corresponding to the heuristic weights, as shown in Fig. 8.

Individual				
HorizCenterHeuristic weight	*VertCenterHeuristic* weight	*VertDiscHeuristic* weight	*BuildFromAfarHeuristic* weight	*DumpFromAfarHeuristic* weight

Fig. 8: An individual in the evolutionary algorithm population is defined by five attributes corresponding to the weights of the partial heuristics.

Individuals in the first generation are initialized with random weights. The fitness of individuals in each generation is given by their score in a *ColorShape-Links* competition between them. In this competition, all individuals play against all others, and all have the opportunity to play first against any given opponent.

Individuals are selected for the next generation using tournament selection [7], completely replacing the original population. In tournament selection, two

individuals are randomly drawn from the original population. A copy of the one with best fitness is selected for the next generation, and then both individuals are returned to the original population and can be selected again. This process is repeated n times, yielding the base population for the new generation.

Each pair of individuals in this new base population then undergoes uniform crossover with probability p_c, being replaced in place by their offspring. Uniform crossover means that each attribute between the mating individuals is swapped with probability p_{c_a}.

Mutation is the final step in defining the new generation of individuals. Individuals undergo mutation with probability p_m, and in these, mutation is applied to each attribute with probability p_{m_a}. Since attributes – i.e. the heuristic weights – are numeric, a gaussian mutation operator, in which a value drawn from the normal distribution (mean μ, standard deviation σ) is added to an attribute, is the most natural choice.

At this point a new generation of n individuals is fully formed. Their fitness is then obtained by performing a new *ColorShapeLinks* competition between them, similar to what was done for the first generation. The algorithm stops when it reaches a predefined number of iterations, l, returning the individuals ordered by fitness, from best to worst. Otherwise the process starts over, with a new generation undergoing selection, crossover and mutation, further improving the heuristic weights of the population.

4   Results

4.1   Heuristic Weight Optimization

After testing several parameter combinations for the evolutionary algorithms, we settled for the values displayed in Table 1. These values provided a good balance between optimization quality and the duration of the optimization process – which was performed using standard Simplexity rules.

Table 1: Parameters used for the evolutionary algorithm runs.

Param.	Value	Description
n	50	Population size
l	500	Number of generations
p_c	0.4	Crossover probability for pairs of individuals
p_{c_a}	0.5	Crossover probability for each attribute
p_m	0.2	Mutation probability per individual
p_{m_a}	0.5	Mutation probability for each attribute
μ	0.0	Mean of normal distribution used for mutation
σ	0.25	Standard deviation of normal distribution used for mutation

We performed five runs of the evolutionary algorithm with different initial populations, and the best individual at the end of each run was selected for a final competition. The final standings for this competition, as well as the weights of each of these "best" individuals, are presented in Table 2.

Table 2: Weights of the partial heuristics for the best individuals in five runs of the evolutionary algorithm. Position (Pos.) refers to the position of the individual in a final competition between these individuals.

Heuristic \ Pos.	1st	2nd	3rd	4th	5th
HorizCenter	0.5386	0.8322	1.0383	0.9882	0.7913
VertCenter	4.5551	8.0820	0.1755	6.9733	5.4766
VertDisc	-0.9044	-0.5215	8.4260	0.2584	0.0797
BuildFromAfar	2.9906	2.2055	3.4491	2.3593	5.5618
DumpFromAfar	4.9090	6.7064	3.3461	6.7073	4.1143

Naturally, the individual in the first position was selected as our final agent for the *ColorShapeLinks* competition.

4.2 The *ColorShapeLinks* Competition

The competition was held on two tracks, each having specific parameter values, and naturally, separate final standings. The first session, the *Base Track*, was run on Simplexity's default board configuration, as shown in Table 3. Here, agents were limited to one CPU core.

The second session, the *Unknown Track*, was held in a configuration to be defined by the results of that week's EuroMillions draw, after the final submission deadline. This way, the AI agents were unable to specifically prepare for this configuration. The final parameters for this session are also shown in Table 3. There were no limits in the access to the CPU cores, with the AIs being able to take full advantage of the available computing power, provided multithreading was implemented.

In the *Base Track*, the submitted AI was ranked 6th – the last position, thus a clearly poor result. On the other hand, in the *Unknown Track*, SimpAI, working at its full multithreaded capacity, was the runner-up, proving that it can be efficient and adapt to boards of different dimensions.

5 Discussion

5.1 Optimization of Heuristic Weights

By analyzing the five individuals from Table 2, it is possible to notice some patterns, with only the individual in the 3rd position clearly deviating from

Table 3: Configurations for the two competition tracks.

Parameter	Value	
	Base Track	*Unknown Track*
Rows	6	8
Columns	7	13
Win Sequence	4	4
Round Pieces	10	26
Square Pieces	11	26
Time Limit (ms)	200	325

the rest. All individuals, except the 3rd and 5th, had both *VertCenter* and *DumpFromAfar* heuristics with relatively high values, suggesting the expected synergy between the two. Another similarity between the individuals, except with the 3rd, was the fact they all considered the *HorizCenterHeuristic* less effective than its vertical counterpart.

Considering how the 3rd individual contrasts with the rest of the group, it is important to try and find a difference between the agents classified above and below it, possibly giving further insight as to how the heuristics may have influenced the tournament results. The most significant difference between the two groups has to do with the *VertDiscHeuristic*: the top 2 individuals have a negative value, while the 4th and 5th do not. This means that the 3rd individual specifically defeated agents with positive *VertDiscHeuristic* weight and that, unlike it, considered *VertCenterHeuristic* more valuable than *HorizCenter-Heuristic*. This tells us that matches between them may have extended to the higher spaces of the board and, due to a conflict between *VertDiscHeuristic* and *HorizCenterHeuristic*, the 4th and 5th individuals did not try to get the spaces where the areas of both heuristics overlapped, with the 3rd agent capturing them and gaining control of that area of the board, significantly increasing its chances of victory.

As already stated, all individuals, except the 3rd, valued *VertCenterHeuristic* quite more than *HorizCenterHeuristic*. We argue several possible reasons, which individually or in combination, may have lead to this outcome:

1. Simplexity's standard dimensions – used for the tournaments ran by the evolutionary algorithm – have more columns than rows. This may have affected the *HorizCenterHeuristic* performance, as it would be triggered less often and more games would have been won due to *VertCenterHeuristic*, since there would not be as many played positions in its *HorizCenter* area.

2. *VertCenterHeuristic* is more efficient than the horizontal version. Regardless of the size of their central areas, *VertCenterHeuristic*'s central area is more accessible due to being in a location very likely to be reached in most games: the middle rows of the board. Contrary to this, *HorizCenterHeuristic*'s central area extends to the top of the board, and consequently, an AI prioritizing

this heuristic may be wasting its pieces in positions not as commonly reached in an average game.

3. The *HorizCenterHeuristic*'s weight may have been affected by a conflict with *VertDiscHeuristic*. This is further explored below when analyzing the latter's weight.

In any case, the optimization process clearly led to agents which "understood" the importance of both *CenterHeuristics*, recognizing their overlaying areas as the most valuable board positions.

The most unexpected weight was that of *VertDiscHeuristic*, with it having been attributed a negative value for the top 2 contenders. We highlight two possibilities as to why this may have happened:

1. The heuristic only has real influence in the agent's decision if it is choosing a play in a board where there are available positions in and out of its top area. In other words, there may be several games where *VertDiscHeuristic* is never actually used, for example when the area immediately below it is sufficient to finish games before *VertDiscHeuristic* is actually employable. The top 2 individuals may have won most of their games by mostly following the other heuristics.

2. The fact that *VertDiscHeuristic* weights are negative in the top 2 individuals may be a coincidence related with the initial randomization of weights in the optimization process. However, there may be an actual reason as to why these weights are negative, and we argue it may be related with a conflict between *VertDiscHeuristic* and *HorizCenterHeuristic*. The latter's center area consists of columns that extend to the top of the board, meaning that part of them will come in conflict with the former's "exclusion" zone. It may be that, by affecting the value of the overlapping positions of both heuristics' area of influence, *VertDiscHeuristic* deters the agent from taking top center spaces, leaving them open for the opponent to take, possibly costing games that extended to the final stages. This would mean that *VertDiscHeuristic* ended up doing more harm than good, and by negatively weighting it, the optimization process recognized those top center positions as important for the late-game.

Summing up, it is possible to identify the best combination as having *VertCenterHeuristic* and *DumpFromAfarHeuristic* as the most valuable, working together to get rid of and isolate the opponent's pieces, while at the same time maintaining overall control of the board; and *VertDiscHeuristic* with a negative weight, so that it does not conflict with *HorizCenterHeurtistic*.

5.2 Performance of SimpAI in the *ColorShapeLinks* Competition

In retrospective, the poor result of SimpAI in the *Base Track* can most likely be attributed to the track's technical restrictions, which limited computing power to one CPU processor. SimpAI, designed from the ground up for using multiple

processors, did not scale well under these restrictions. Furthermore, even with the use of Zobrist hashing, having five, completely separate partial heuristics – instead of a faster, single one – probably took its toll when evaluating boards. As such, it is our belief that part of this result stems from the fact that SimpAI was not able to search as deep as its opponents, in spite of the implemented optimizations.

The *Unknown Track* took place on a workstation with 8 cores/16 threads[3]. This was most likely the reason that, SimpAI, working at its full capacity, finished second – thus offsetting the computational weight of using five separate heuristics.

6   Conclusions and Future Work

In this paper we presented SimpAI, an AI agent created for the *ColorShape-Links* board game competition. The agent was implemented with an efficient Minimax-type search, and an heuristic function composed of five different partial heuristics, the weights of which were optimized with an evolutionary algorithm. SimpAI reached the second position in the *Unknown Track*, a promising result given the fact that the track's parameters were unknown beforehand, thus requiring an AI agent with good adaptation capabilities.

There are some elements that are likely to bring significant improvements to the agent's performance. For example, a wider initial set of partial heuristics could help refine what are indeed the most important features for winning *Color-ShapeLinks* matches. From the results, it was clear that some heuristics, such as *VertDiscHeuristic*, did not work out as we initially envisioned – and could probably be excluded altogether, speeding up board evaluations. Additionally, a few partial heuristics use "magic" numbers to provide their evaluation. These could also be considered as parameters to be optimized by the evolutionary algorithm, further refining a winning strategy. Regarding the optimization process, a more comprehensive experimentation with the evolutionary operators and their parameterization could potentially yield better winning combinations. Finally, and while this was not the intended learning goal for the authors of this project, the use of MCTS, deep learning and/or reinforcement learning, could potentially produce stronger and harder to beat agents.

References

1. Brain Bender Games: Simplexity. Discovery Bay Games (2009), `https:// boardgamegeek.com/boardgame/55810/simplexity`
2. Eliahu, D., Spillinger, O., Fox, A., Demmel, J.: FRPA: A Framework for Recursive Parallel Algorithms. Master's thesis, EECS Department, University of California, Berkeley (May 2015), `http://www2.eecs.berkeley.edu/Pubs/TechRpts/2015/ EECS-2015-28.html`

---

[3] Information provided by the organizers at our request.

3. Fachada, N.: ColorShapeLinks: A board game AI competition for educators and students (Dec 2020), `https://arxiv.org/abs/2012.09015`, arXiv:2012.09015 [cs.CY]

4. Fachada, N., Códices, N.: Top-down design of a CS curriculum for a computer games BA. In: Proceedings of the 2020 ACM Conference on Innovation and Technology in Computer Science Education. pp. 300–306. ITiCSE '20, ACM, New York, NY, USA (Jun 2020). https://doi.org/10.1145/3341525.3387378, `https://doi.org/10.1145/3341525.3387378`

5. Fortin, F.A., De Rainville, F.M., Gardner, M.A., Parizeau, M., Gagné, C.: DEAP: Evolutionary algorithms made easy. Journal of Machine Learning Research **13**, 2171–2175 (jul 2012)

6. François-Lavet, V., Henderson, P., Islam, R., Bellemare, M.G., Pineau, J.: An introduction to deep reinforcement learning. Foundations and Trends$^{®}$ in Machine Learning **11**(3-4), 219–354 (2018). https://doi.org/10.1561/2200000071

7. Goldberg, D.E., Deb, K.: A comparative analysis of selection schemes used in genetic algorithms. In: Rawlins, G.J. (ed.) Foundations of Genetic Algorithms, vol. 1, pp. 69–93. Elsevier (1991). https://doi.org/10.1016/B978-0-08-050684-5.50008-2

8. Millington, I.: AI for Games. CRC Press, Boca Raton, FL, USA, third edn. (Mar 2019). https://doi.org/10.1201/9781351053303, `https://doi.org/10.1201/9781351053303`

9. Unity Technologies: Unity®(2020), `https://unity.com/`

10. Wenzel, M., Warren, G., Dykstra, T., Victor, Y., Latham, L., Wagner, B., Dolean, T., O'Connor, R., Jones, M., et al.: .NET Standard. Microsoft Docs (Mar 2020), `https://docs.microsoft.com/dotnet/standard/net-standard`

11. Yannakakis, G.N., Togelius, J.: Artificial Intelligence and Games. Springer (2018), `http://gameaibook.org`