Serpentine

Artificial Intelligence Association

Technical report

AI Snakes competition

Bram Grooten
Imre Schilstra
Wolf van der Hert
Dik van Genuchten

May 2020

Abstract

Four members of team Serpentine, a student team of the Eindhoven University of Technology which competes in AI programming contests, participated in the AI Snakes competition. This tournament is part of the IEEE Conference on Games 2020. For the contest teams create a bot that can play 1v1 multiplayer snake. The main strategies of our bot *AdderBoaCobra* are: (1) alpha-beta for optimal pathfinding and (2) circling around the apple to secure a win. Out of the seven competing teams our bot achieved the second highest winning ratio.

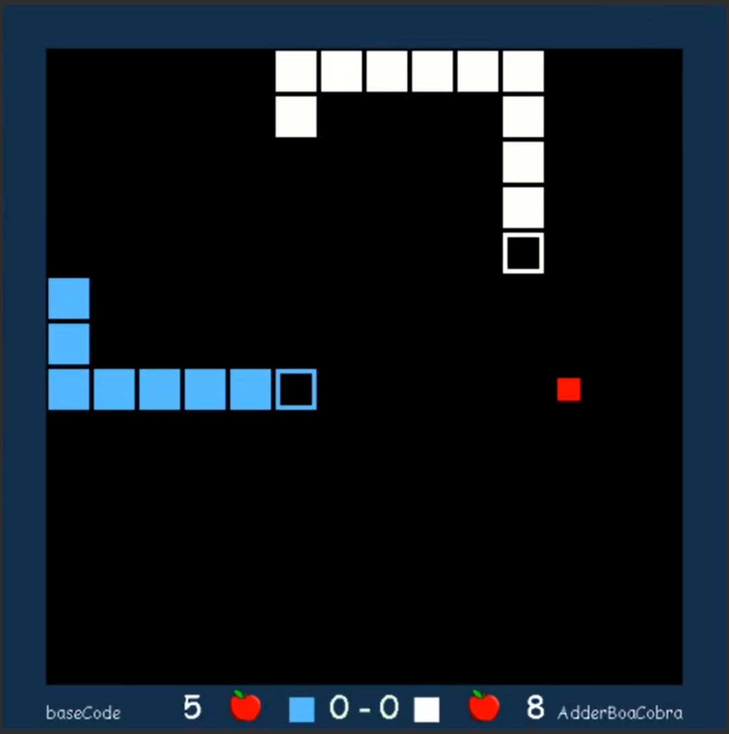

Figure 1: The AI Snakes game.

Contents

1 Introduction

In March and April of 2020 four students of the Eindhoven University of Technology competed as team Serpentine in the AI Snakes programming contest [1], which is part of the IEEE Conference on Games [2]. In this competition teams had to design and implement a bot that can play 1v1 multi-player snake. After working on multiple approaches we submitted our *AdderBoaCobra* bot on the 3rd of May, 2020. Out of the seven competing teams we had the second highest winning ratio. Our code can be found on the GitHub page of team Serpentine [3], where we also added a playable version so people can try to defeat *AdderBoaCobra* themselves.

The game is very similar to regular Snake, except that now there are two competing snakes. The game is played on a 2D board of 14x14 tiles. Both snakes start out with length of 3 tiles and grow by one if they eat the apple. Each turn the bots have 1 second to determine the next direction, after which they move simultaneously. Whenever a snake collides with a wall, the other snake or itself, it loses the game. The only exception is that when snakes collide head-to-head, the longest snake wins. If there hasn't been any collision after 3 minutes the longest snake wins the game as well.

This report is structured as follows. First we will talk about our main strategies in section 2, after which we go deeper into the corresponding implementations in section 3. In section 4 we present the results of the competition. We discuss possible improvements in section 5 and section 6 concludes the report.

2 Strategy

The name of our bot, *AdderBoaCobra*, is a reference to the ABC of *alpha-beta circle* since that is the main tactic of our snake. In section 3 we will go deeper into the implementation of the alpha-beta algorithm and how we circle exactly, but first we will provide the general idea of our strategy in this section.

Our snake has three main strategies: it is either circling, going for the apple, or maximizing reach. These three states will be explained below. Other strategies that are capsuled in these states are: enclosing the other snake and making sure that our snake is not enclosed. We did not put much emphasis on these last two settings, but our implementation of alpha-beta is able to correctly deal with these situations most of the time.

2.1 Circling

As luck is playing a major role in the game, it is best for us to stick to a 'winning state' as soon as possible. If our snake has more apples than its opponent then that lead should be secured until the end of the game. Putting it differently: our snake should make sure its score stays higher than the score of its opponent. To achieve this we will try to keep the other snake from increasing its score by circling around the apple once our snake is long enough.

Sometimes it is not necessary or possible to make a complete circle around the apple, but we can still eliminate the opponent's chances of winning. The details will be explained in subsection 3.2. All of these movements will still be referred to as 'circling'. An example of the result of this strategy can be seen in Figure 2.

2.2 Going for the apple

If we are not able to perform 'circling' at the moment, we want to eat more apples so that we get to such a circling situation as quickly as possible. We want to go towards the apple in the fastest way

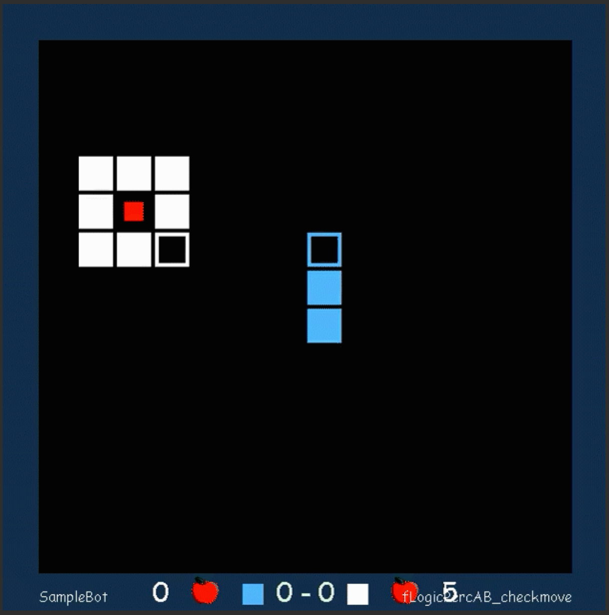

Figure 2: The circling tactic of our snake. For an animation, open this PDF in Adobe Acrobat Reader or Internet Explorer.

of course, but we also need to make sure that we get there safely. This means that we have to check that we don't get enclosed by the other snake or even ourselves.

2.3 Maximizing reach

When we see that the other snake's head is closer to the apple than our own head, we decide to go for as much space as possible, because we assume that our opponent will get to the apple first. This means that we want our snake to take in a position such that when the other snake eats the apple, we have as many empty tiles of the board as possible that are closer to us than to our opponent. In this way we try to increase our chances of being quicker to the next apple, which spawns uniformly at random on one of the empty tiles.

At first, our strategy was to go to the center of the board whenever our snake notices that the opponent is closer to the apple. The idea was the same: to be quicker towards the next apple. However, as you can see in Figure 3, this is not optimal. When our snake (shown in white) is closer to the other snake there are more empty tiles within our reach.

2.4 Overall logic

What strategy should our snake use at which moment? That depends on the current situation. Our snake should perform 'circling' if its score is higher than the opponent, its body (length) is sufficient to leave no gaps in the circle and its distance to the apple is shorter than the opponent's. Our snake should go eat the apple when it's not able to perform 'circling' yet and it's closer to the apple than the opponent. If our snake cannot perform 'circling' and the opponent is closer to the apple, then it goes for as much space as possible.

During these three possible states we might stumble upon an opportunity to enclose the opponent. In that case our snake will do so. We are not actively trying to enclose our opponent at all times. Also, when we recognize that a certain move is dangerous (meaning there is a high probability that we will be enclosed) then our bot chooses a different move.

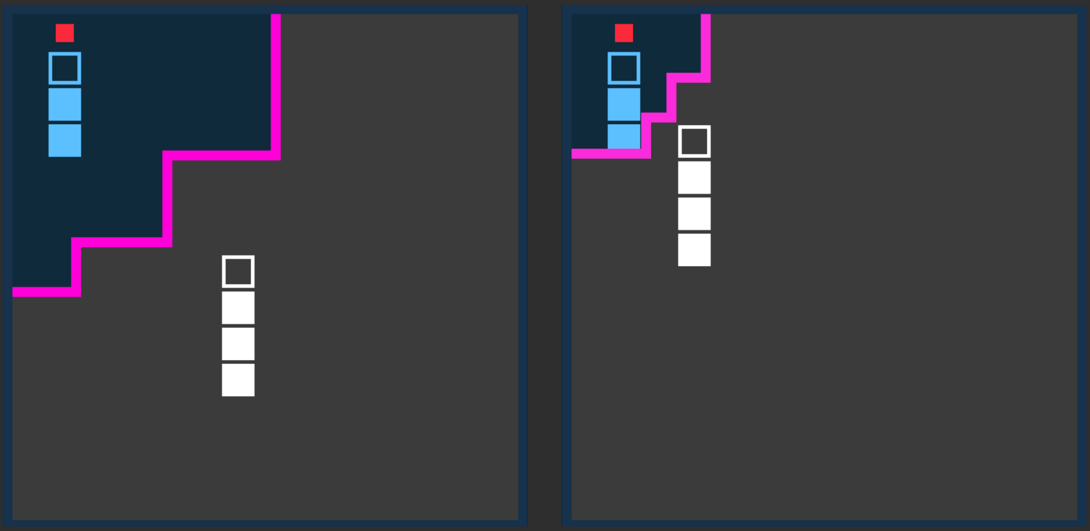

Figure 3: Left: Stay in center. Right: Maximize reach. The grey coloured space covers the 'tiles closer to our snake', with our snake being the white snake.

2.5 Draws

Another important aspect to cover is the occurrence of a draw. A draw might occur when: both snakes die in the same turn, the time runs out, or there is a head-to-head collision. When this happens, the snake with the highest score wins. Only if both snakes have the same score, a draw truly occurs.

The question we asked ourselves: How should we handle draws? First of all, our snake accepts all collisions if it has a higher score, as this results in winning instead of a draw. Furthermore, since the tournament winner is selected by its winning ratio where draws don't count[1], draws do not effect our competition score. For example, if we draw every match and win just one, we have a winning ratio of 100%. Therefore, assuming our snake wins often enough, we also accept collisions that lead to a draw.

A situation that occurred often is: two snakes colliding head-to-head on the location of the apple. Both snakes want to eat the apple, and if both accept a draw, it will happen. If we would avoid draws, then this would be a disadvantage for us since the other snake will get to eat the apple each time this situation occurs.

3 Implementation

In this section we will explain the algorithms we implemented based on our strategy. For implementing most tactics, except for circling the apple, we used the alpha-beta algorithm. The reason for this is that alpha-beta determines which move to do based on the highest score it will get in that state. This also allows us to prioritize a certain strategy above another.

3.1 Alpha-beta

Alpha-beta [4] is an advanced implementation of the minimax algorithm [5], which is used in two-player games like the board games Chess and Go. Alpha-beta results in the same outcome as minimax, but with a faster running time. In order to explain how alpha-beta works we will first explain how minimax works and how we implemented it.

[1] At least that's how we interpreted the rules. See section 4 for more information.

Table 1: The scores used to determine the value of a state, as seen from our snake. The score is calculated by multiplying the weight with the given number.

variable name	weight	possible numbers	short description
weAteApple	800	1 or 0	1 if we ate the apple
oppAteApple	-500	1 or 0	1 if the opponent ate the apple
circlePart1	1000	1 or 0	1 if we covered all circle points
circlePart2	80000	1 or 0	1 if circlePart1 is 1 and we follow our tail, and thus have successfully made the circle
weCantMove	-20000	1 or 0	1 if we have no valid moves left
oppCantMove	10000	1 or 0	1 if the opponent has no valid moves left
goodHeadColl	5000	1 or 0	1 if a head-to-head collision occurred when we were the same length or longer thus resulting in a win or a draw
badHeadColl	-20000	1 or 0	1 if a head collision occurred where we would lose the game
distanceToApple	-5	$[0, 2 \cdot \text{mazeSize}]$	the Manhattan distance between our head and the apple. Not taking into account if a route is actually possible (which required too much running time)
percentageCloser	50	$[0, 1]$	the percentage of empty tiles that are closer to our head, also not taking into account if the route is possible

3.1.1 Minimax

In short, minimax is a depth-first search algorithm where each node is equal to a possible state after $depth$ number of moves. The algorithm gives a value to each state at the maximum depth level, which we call the *score*. For an explanation on how we determined the score, see subsubsection 3.1.3.

The score is calculated from the perspective of our snake, meaning that a high score implies that we have an advantage in this particular state and a low score implies the opponent has an advantage. At each depth level, the lowest or highest score of the multiple child nodes is returned based on which player makes the move at that depth level. If it is our turn, then the algorithm takes the maximum score of all child nodes, meaning we choose the best possible move. If it is the opponent's turn, the minimum score of all is taken, meaning the opponent chooses the worst possible move for us.

3.1.2 Alpha-beta pruning

Alpha-beta pruning uses the fact that minimax is depth-first search as an advantage in order to skip certain states from being examined. It stops evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move. Therefore, alpha-beta evaluates less states which makes it faster than minimax.

3.1.3 State evaluation

The evaluation of a state is given by the sum of the weighted values as shown in Table 1. The evaluation of a state is calculated at the max depth level or leaf of the minimax graph. As a result of this, calculating the state score inefficiently results in reaching a lower depth level. Reaching a lower depth level means that the algorithm looks less steps into the future to determine if a move will be useful. For this reason we needed to make the function that evaluates a state run as efficiently as possible.

3.2 Circling: How and When

To our calculations, 41 scenarios[2] can occur in which circling would result in winning the match. Each scenario has a set of conditions. First of all, our snake should be closer to the current apple than the opponent. Secondly, it needs to have a higher score than the opponent. Thirdly, it needs to have an even number of body parts, to ensure a closed circle.

Then, the position of the apple is checked. If the apple is on a side or in a corner, less tiles have to be covered to enclose the apple when circling, providing chances to circle the apple in earlier stages of the game. The apple does not always have to be completely enclosed. If there is a route towards the apple for the opponent, but no way out, a win is also secured. Therefore, as the snake starts with 3 body parts, the first number of body parts that would suffice is 4. Our snake could then circle next to the apple if it were in a corner.

The problem is however that at this point, the opponent can still gain one point before dying. Should it then meet our snake's head, a draw occurs. The chances are small, but we decided that circling should always ensure winning. Therefore, the first number of body parts that suffices is 6, as it is then possible to have at least 2 points more than the opponent. The scenario's are divided in 'fully enclosed' and those that leave a route towards the apple but no way out, for which our snake needs a minimum of 1 and 2 points in advance respectively. All the conditions for the scenarios are summarized in Table 2.

Table 2: Conditions for circling

min. body parts	apple position	min. points in advance	nr. of scenarios
6	corner or on a side $\leq$2 off-corner	2	8
8	anywhere in the middle	1	1
8	corner	1	4
8	on a side, $>$2 off-corner	2	8
10	corner or on a side 1 off-corner	1	8
12	corner or on a side $\leq$2 off-corner	1	8
14	on a side, $\geq$2 off-corner	1	4
total			41

To make our snake circle the apple, each scenario has its unique set of coordinates relative to the position of the apple that our snake needs to cover. Once our snake has covered all of these 'circle positions' and is able to move directly to the previous position of its tail, it sets a Boolean `circle_complete` to true. This causes our snake to only return the direction towards its tail, which is how we ensure the stability of our circle.

The circle positions can be switched to a backwards order, such that the number of steps that are needed to complete the circle can be decreased. This happens if our snake comes from a direction in which following the set of coordinates in regular order is not immediately possible.

3.2.1 Going towards the circle

Our circling algorithm uses an implementation of breadth-first search (BFS) to go from one circle position to the next. Before we can circle, our snake uses alpha-beta for its optimal pathfinding. Therefore, there has to be a moment where we switch from alpha-beta to circling with BFS.

At first we switched to BFS immediately when our snake notices it can go circle. However, if we are far away from the apple we will first need to get there. And since moving across the board is most safely

[2]Many of them are similar symmetrically. See [6] for the full list.

implemented in alpha-beta, we want to use that as long as possible.

Thus our rule changed to: use alpha-beta until you are on a circle position, then switch to BFS. See lines 5-10 of Algorithm 1. When we saw that circles were sometimes not made very quickly in this manner, we added the restriction that alpha-beta would only be used if our snake is not close enough to the apple. We defined "close enough" to be a Manhattan distance of at most 4.

If our opponent managed to get its snake's head next to the apple while we were still trying to form our circle, then we decide to switch back from BFS to alpha-beta. Finishing that circle will probably not work anymore, so we choose to use the safest part of our implementation.

Algorithm 1 Piece to determine whether we use alpha-beta or BFS to go towards a circle.

```
 1: $useAlphaBeta$ = false
 2: if $distance(head, apple) > 4$ and not $circle\_complete$ then
 3:     $useAlphaBeta$ = true
 4: end if
 5: for each $circle\ position$ do
 6:     if $snake.contains(circle\ position)$ then
 7:         $useAlphaBeta$ = false
 8:         break
 9:     end if
10: end for
11: if $distance(head\_opponent, apple) < 2$ and not $circle\_complete$ and $closed\_circle\_type$ then
12:     $useAlphaBeta$ = true
13: end if
```

4 Results

In this section we will present the results of the international AI Snakes competition that was held on May 9th, 2020. The full tournament can be viewed on YouTube [7]. Our snake is called *B_j_grooten*, since the organization named the bots after the email address of the person who submitted it. All the bots and their respective total results can be seen in Table 3.

Table 3: The total results of the competition.

name	wins	losses	draws	wins+losses	winning ratio
Aoki_eita1130	24	3	3	27	0,89
B_j_grooten	16	6	8	22	0,73
D_kabirov	17	13	0	30	0,57
Bertram_timo	13	12	5	25	0,52
V_vasilev	12	13	5	25	0,48
V_smirnov	6	22	2	28	0,21
A_zhuchkov	5	21	4	26	0,19

Based on the winning ratio, which we computed as follows:

$$\frac{\#\text{ wins}}{\#\text{ wins} + \#\text{ losses}}$$

we ended up in second place. When looking at the number of wins only, we finished third, just one win behind *D_kabirov*. During the competition we assumed that the defining metric was the winning ratio (where draws don't count). It was not completely clear which metric the organization used for the final standings, but at least there was a clear winner.

The competition was played out as a round-robin tournament, with each bot playing five games against every other bot. The results of each match-up are shown in Table 4.

Table 4: All the matchups of the competition. Each entry shows the number of wins for the snake in the row. The remaining games ended in a draw.

	Aoki_eita1130	B_j_grooten	D_kabirov	Betram_timo	V_vasilev	V_smirnov	A_zhuchkov
Aoki_eita1130	-	1	5	3	5	5	5
B_j_grooten	1	-	4	0	3	3	5
D_kabirov	0	1	-	1	5	5	5
Bertram_timo	2	2	4	-	0	3	2
V_vasilev	0	1	0	3	-	4	4
V_smirnov	0	1	0	2	4	-	3
A_zhuchkov	0	0	0	3	0	2	-

5 Discussion

After the competition we were happy with our second place, but also a bit disappointed of not becoming the champion. We noticed during the live stream of the competition that we were the only bot that had a circling strategy, so we thought our chances were pretty good. In this section we will discuss some of the improvements we made and describe bugs that were still in our code. We will finish with a couple of suggestions for the competition.

5.1 Bot improvements

There was a bug in our code which made it impossible for our snake to see a collision correctly if both snakes grabbed the apple simultaneously. In our alpha-beta implementation we did not take into account that both snakes could grab the apple in the same turn. Our snake was made to think that it would grab the apple first in such cases. Due to this fact the snake would think that the move would result in a draw or win, where it in fact resulted in a loss or draw respectively. Unfortunately, we only discovered this during the competition. We think it would have been an easy bug-fix if we would have spotted it earlier.

One of the tactics used by the competition was protecting the apple by "scaring" a shorter snake away, as a head collision would result in a loss for the shorter snake. The longer snake defended its territory, instead of our apple-circling strategy. When this happens there was a chance that our snake would end up in a loop where all the states would be revisited constantly. A possible fix to this would be to remember the last few states we were in and detect if a move would result in the same state. As soon as this is be detected we would make a different choice which could break open the game again, and thus result in a possibility to win.

In the end, it seemed the number of wins was taken into account instead of the winning ratio. In that case, accepting draws was a critical mistake. The winning bot 'Aoki_eita1130' did not seem to accept any draws which showed more promising results.

5.2 Competition improvements

We found out that our tests were not fair for the white snake, see Figure 4b, as it started with less empty tiles closer to it. This caused the first apple have a higher chance to spawn closer to the blue snake resulting in an unfair advantage. The logic with which the test were run placed the top snake from the list always on the blue spot, and the lower always on the white spot, thus giving the lower bots an advantage. In the competition this was fixed by having a symmetric starting position, see Figure 4a.

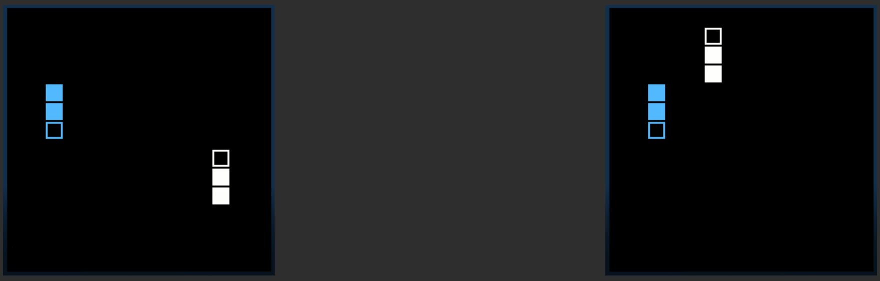
(a) A symmetric fair starting position (b) An asymmetric unfair starting position

Figure 4: Starting positions

We think that having more games in the competition would give less opportunity for luck to determine the outcome. It would also be less enjoyable to watch though, as it could take many days to complete a tournament in which each bot plays 100 matches against every other bot.

If there is another edition of the AI Snakes competition in the future, we would advise to make some adjustments to the game to keep it interesting. Otherwise the same bots could easily be submitted. Some ideas are for example:

- having more than one apple on board at once (which would make our circling strategy worthless),
- having different kinds of fruits (that are worth different amounts of points),
- being able to go through walls and appear on the other side of the board.

For more inspiration, just google the words "play snake" and have a look at the different options of the game [8].

6 Conclusion

We are proud of the bot that we have produced in the previous couple of months. Although our snake still has some bugs, overall the strategy and implementation are satisfactory. Alpha-beta turned out to be a powerful method for us to look multiple turns into the future and select the optimal move. The circling strategy worked quite well in securing many of our wins during the competition. We really hoped to become the champion of AI Snakes, but we'll have to be content with the second place.

As the student team Serpentine, which is named after a snake, we are glad that we finally made a bot that can play the game Snake fairly well. We really enjoyed working on this competition. Hopefully there is another edition next year, so other members of our association can adjust and improve our *AdderBoaCobra*.

References

[1] AI Snakes competition. https://agrishchenko.wixsite.com/snakesai. Accessed on: 9 May 2020.

[2] IEEE Conference on Games. http://ieee-cog.org/2020/competitions_conference. Accessed on: 24 May 2020.

[3] GitHub repository AI Snakes - Serpentine. https://github.com/TeamSerpentine/cog-aisnakes-2020.

[4] Donald Knuth and Ronald Moore. *An analysis of alpha-beta pruning*. Artificial Intelligence. Vol. 6(4), 293-326. (1975). https://pdfs.semanticscholar.org/dce2/6118156e5bc287bca2465a62e75af39c7e85.pdf

[5] John von Neumann. *On the theory of board games.* Math. Ann. 100, 295-320 (1928). https://doi.org/10.1007/BF01448847

[6] Wolf van der Hert. *The Theory of Snakey Circles.* https://www.dropbox.com/s/tk47v5dd8y1vq3k/The%20Theory%20of%20Snakey%20Circles.pdf?dl=0. Accessed on: 24 May 2020.

[7] AI Snakes Competition 2020 - YouTube. https://youtu.be/eqxpSO2_HRI?t=194. Accessed on: 9 May 2020.

[8] Play Snake - Google. https://www.google.com/search?q=play+snake. Accessed on: 29 May 2020.