G2: Slather

Project 2

COMS 4444 Programming and Problem Solving

2016 Fall

Bruce Wu

ChengYu Lin

Yogesh Garg

Problem Statement

This project is inspired by slither.io, a strangely addictive multi-player online game.
You are controlling a circular cell living and moving in an environment with other cells. Cells grow over time. Once they double in radius they stop growing, and can reproduce into two smaller daughter cells that can independently move about the environment. Cells produce a pheromone that is secreted onto the ground as a cell passes (a small point-like dose at the center of the cell). Other cells of the same species are able to traverse pheromone points, but other species (i.e., other players' cells) find the pheromone repulsive and will not traverse such points. Pheromones do not last forever though; they eventually wear out.
Your goal is to maximize the population of your cell type after many turns of the game have elapsed. To achieve this goal, you will program the cell to decide its actions on each turn of the simulator. A cell may move, or reproduce if it has reached full size. The cell has access to a limited amount of locally sensed information (see below). It also can keep a small amount (one byte) of state memory from one turn to the next. Your code will be called by the simulator on each turn with this memory item supplied, and your code should return a byte for use in the next iteration. (Obviously, you are not allowed to declare persistent variables or read/write persistent data that would circumvent this requirement.)
On each turn, the simulator will choose a random ordering of all cells on the board, and resolve moves/growth/reproduction one cell at a time according to that order.
The world is 10cm by 10cm, and wraps around like a torus, so that a cell moving off the left edge appears on the right edge, and analogously for the top and bottom edges. Cells start out with a diameter of 1mm, located at random nonoverlapping positions in the world. You will instruct the cell to move by up to 1mm on a turn, in any direction. You will be supplied with the memory state together with a list of pheromone points (from both self and foreign species -- you will know which points are from which species) located within $d$mm of the circumference of your cell, where $d = 1$ is a parameter. This list will include all self-pheromone points covered by the cell. Additionally, any other cells that extend into this $d$mm buffer will be visible: you will be supplied with the center and radius of all such cells. The simulator will reject any moves that would cause the cell to traverse a foreign pheromone or overlap another cell; in such a case the cell will stay put.
Cell diameters grow by up to 1% each turn, but stop at 2mm. Cells will grow by less than 1% if the simulator determines that further growth would cause the cell to traverse a point covered by a foreign pheromone or by another cell. This growth is automatic, and determined by the simulator without explicit instructions from your controller, once the cell has moved. The deposition of pheromones is also automatic.
When a cell reaches a diameter of 2mm, you may instruct it to reproduce into two daughter cells each of diameter 1mm (some cell mass is lost in this process). The simulator will choose a random direction for the split, and the two daughter cells will be within the footprint of the original parent cell. When a cell reproduces, there is no additional cell movement or growth on

that turn. At the end of reproduction you give the simulator two different memory bytes, one for each daughter cell.
An unimpeded cell should be able to reproduce in 70 turns. The simulator will stop the simulation when 1000 consecutive turns have happened without any reproduction.
Several aspects of the simulations will be determined by parameters:

d
The distance that a cell can see.
t
The number of turns before a pheromone wears out.
p
The number of players (species).
n
The number of starting cells for each player.
We will vary these parameters when we run tournaments at the end of the project. Your player will have access to d and t, but will not know p or n since this is nonlocal information.
Some initial things to think about:

- How does t affect strategy?
- How might you achieve some kind of emergent behavior from local actions? What kind of emergent behavior do you want?
- What would you use the memory byte for?

Motivation

The overall goal of this problem is how to achieve goal optimal behavior for a group of players given each one of them only has local information. Each cell/player only has local information: It cannot see beyond its vision, and it cannot keep a memory of more than one byte of information. Although the simulator does provide the current location of the cell, in an ideal refined version of it, each cell will only get the neighbor information relative to its own coordinates, so it has very limited access to the big picture. Many concepts of human players based on global information might be therefore not applicable to the program we are writing.

We abstracted the problem into having two kinds of player, scouts and occupiers. The scouting players will try its best to move as far away from current location as possible, therefore maximizing potential future growth and pheromone traces, whereas the occupier will try its best to defend the current territory itself and its neighboring ally cells occupy.

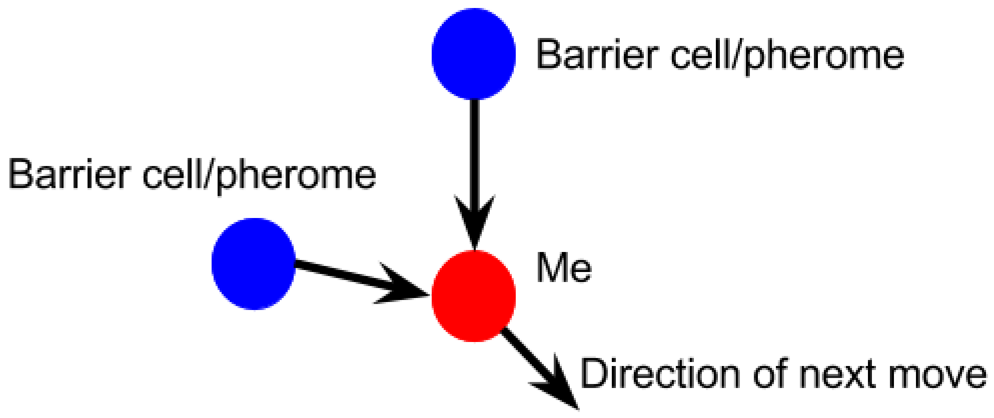
Individual Strategy Evolution

1 Scouts

The role of scouts is to find the free spaces and head towards them. Based on this idea we have several approaches:

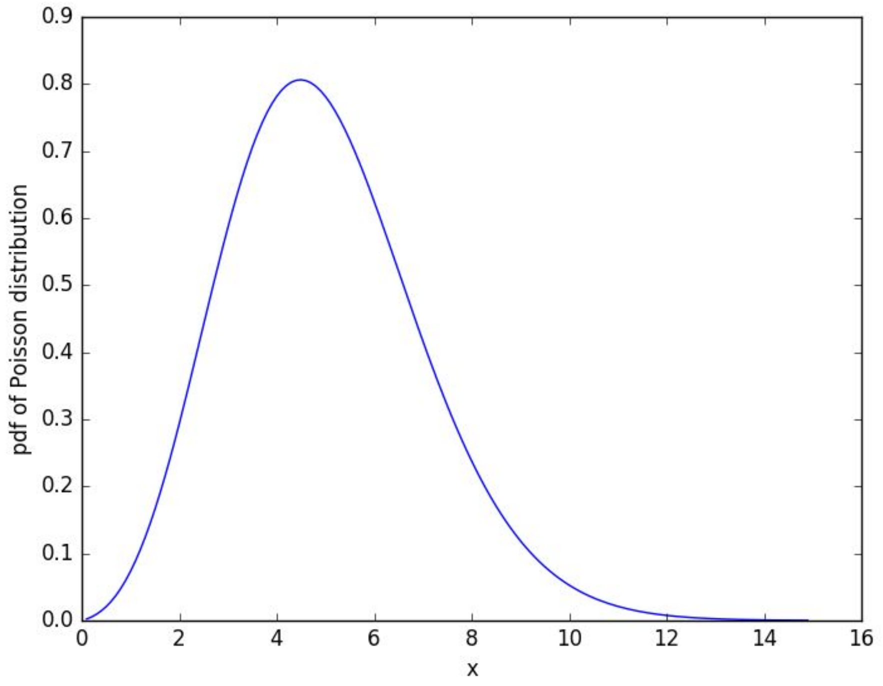
Summing up vectors:

Although the most straightforward idea is to calculate the largest free angles (it will be discussed later). In our first implementation, for simplicity we just sum up all the vectors from other cells/pheromones to my own cell, and let the cell go along this direction. In the first week, this strategy was dominating other groups' first ideas like simple circling, random walks and so on.

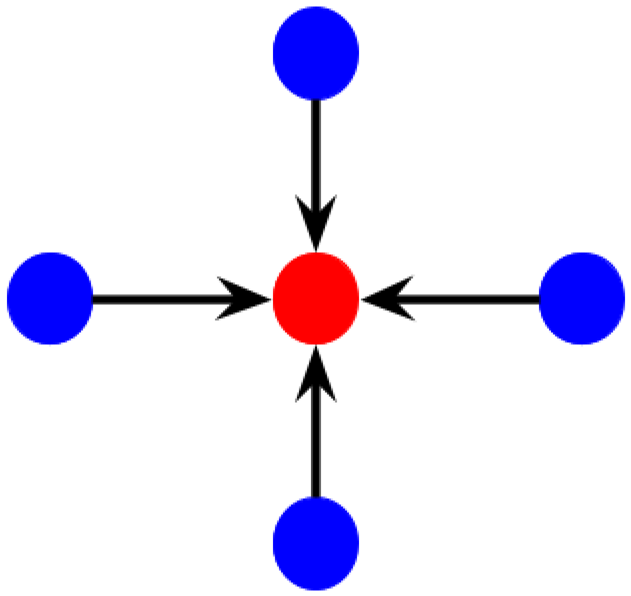


What's good about this idea is:
1) it's easy to implement and also efficient. Although some groups tried to implement the idea of moving towards the largest "free" angle in their first deliverables, they seemed to encounter some bugs which made their players less competitive.
2) We can assign different weights to different vectors. For example, the weight can be inverse proportional to the square of the distance like the gravity in the real world, or it can be a constant for any distance because we only care about it's direction.

It turns out that the choice of weight function is very important when running under different parameters. We tried different weight functions $w(d)$ where d is the distance between our own cell and other cells/pheromones. We found out a very interesting phenomenon: when the vision is small, $w(d) = d$ performs better. But when vision becomes larger, it's better to set $w(d) = 1/d$ or $w(d) = 1/d^2$. Which means we should be looking for a weight function that is growing in the first phase and then quickly diminishing after. Based on this idea finally we choose our weight function to be the probability density function of continuous Poisson distribution: $w(d) = \frac{\lambda^d e^{-\lambda}}{\Gamma(k)}$. By experiments we choose $\lambda = 4$, and now it's robust against any parameters. It turns out that, what matters the most are those cells or pheromones whose distance to my own cell fall into a certain range.

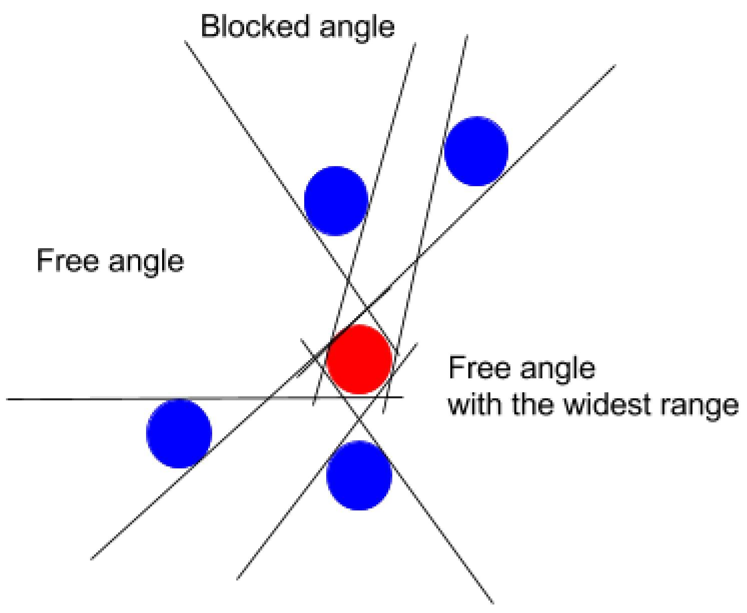


But we cannot ignore the drawback of it. Taking the sum of all the relative vectors, we can only guarantee that other cells and pheromones are less dense in this direction. Although it's a good approximation to the idea of "free" space, it doesn't precisely capture the idea of it. Then second problem of it was, it performs poorly during the late game. That is because, when the map is dense enough, the distribution of all the cells and pheromones is very close to be uniform. In this case, the sum of all relative vectors would have small norm and doesn't give us any useful information. Also it may fail in certain scenarios like the following figure. Later we switch to another approach.



Largest “free” angle:

As in the simulation of second week, we are beaten by group 5 who showed the strength of the largest “free” angle approach. What this strategy does is, it looks around our own cell, figures out the range of angles where our cell cannot be blocked when moving along that direction (illustrated as the figure below), then choose one with the widest range.



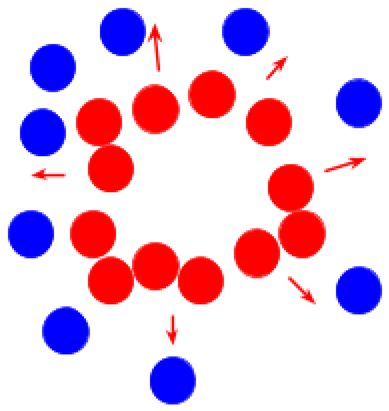
This is a good characterization of the notion “free” space. When you move along the free angle with the widest range, it’s harder for other cells to block your way.

The strategy alone doesn't significantly improve the performance. Here we thank group 5 for providing another important idea: we shouldn't count our own pheromones as barriers. We believe the reason behind this is, our own pheromones block too many choices, and also after we make some failed attempts, we should be able to fall back to other directions instead of being blocked by our own pheromones. With this idea, we came back to tier 1 players.

Also there's an issue with the vision range. When the vision is large and we see too many cells and pheromones, it's harder for us to get a "free" angle even if there's some good move in local area. Also consider the scenario when we see a free space which is far away from us, it doesn't make sense if we couldn't find a way to it. So we limit our vision to 4 (experiment shows this parameter doesn't affect the result too much). In addition, if we cannot find a free angle for a certain vision range, we'll progressively decrease it to find a "local solution".

Aggressive Scout:

The largest free angle idea is too good so that the majority of groups switch to it. As everyone tries to "discover the new world", it's time to bring back one of our old topic: how to "defend our own territories"? Of course we can use strategies like Griders and Circlers. But is it possible that we can let cell itself to decide the defensive move instead of giving them specific orders? Let's first think of what is the most effective way to occupy an area by cells. Of course we should let our cells form a circle that no one else can come inside, a small example:



The best defense is a good offense.

We should let our cell rally towards the perimeters to hold the "enemy's attack" rather than hiding inside. So we design our aggressive scout: if a cell finds out that it's "at war", it only avoids our own cells and moves aggressively towards the enemy cells. Group 9 developed this idea independently and simultaneously, and we ended up dominating other strategies at that week.

There's an issue about when we need to change our strategy from scouting the free spaces to the aggressive one. We decided to use the following criteria: if the total angle blocked by our own cells and pheromones is larger than 180 * 0.7, we attack our enemies aggressively. It's similar to the density estimator which is used by other groups. We expect our cells near the perimeter to move outside to leave a larger space inside. Also for those isolated cells in enemy territories, we hope it can find a way back to one of our own territories to condense our border.

2 Occupiers

Vertical/Horizontal Grider

Our first attempt at occupying is trying to divide the empty board into grids. For each individual cell it would try to go in one of four vertical/horizontal directions, in an attempt to separate the area into ally territory at maximum efficiency. However this turns out to be a bad design since when an occupier is interrupted by enemy cells nearby, it cannot maintain its previous direction, therefore we cannot always assume we can divide the board into rectangular grids. Also the natural restriction of localized information forbids the cells from communicating with each other, so it’s very hard to coordinate between cells to set up even a modest size rectangular grid. We switched to different ideas of occupiers after first week.

Circler

One of our most successful implementation is the Circling occupier, or the Circler. We agreed for any pheromone length t, forming a circle with circumstance t is one of the most efficient way to occupy since no enemy cells can get with in the circle as long as we keep circling. We used the memory byte to store previous angle with the horizontal line, and calculated the required turning degree for the given circle. We also specified that when the original circling angle doesn’t work, we will try any of the four circling directions (current angle plus 0,90,180,270 degrees) so that even if we are blocked on one direction we can continue to circle on other directions.

Distance Keeper/Controlled Spreader

This player appeared from the mid-project realization that we can perhaps benefit from all other players being Scouts. We want a player that starts with its own territory in the beginning, gradually spread to its neighboring empty spaces while maintaining distances between ally cells during the process.

We tried implementing it in the following fashion: given a lower and upper bound of distances between ally cells, for each individual cell it will first calculate the minimum and maximum distances between ally cells. If the minimum value drop below the lower bound or the maximum value rise above upper bound, it will try to go in a randomly sampled direction that increases or decreases the minimum/maximum distance between ally cells. Ideally this would automatically keep the distance between cells in a proper range, therefore achieving the desired result.

However since the cells are using only local information, after a certain point when the inner cells of the cluster reproduces, it will naturally crowd each other out. The outer cells do not know it’s on the perimeter, and therefore cannot expand fast enough to create space for inner cells. In later implementations other groups achieved this effect by having the cells go towards enemy when the density of enemy cells reach a threshold, but it is difficult to achieve such effects intentionally, therefore we did not go forward with this approach.

Improved Grider

An improved/reduced version of Distance Keeper was implemented by us in later stage of the project, where instead of trying to keep a maximum and minimum distance between cells, we only keep the minimum distance between cells. Therefore we can use it to prevent cells from getting too crowded with each other. We rely on this functionality for our final occupier implementation.

GridCircler

Our GridCircler is a combination of Grider and Circler. It will first see if it’s too closed to ally cells, if also it will try to move away from them. If there are not immediate ally cells nearby, it would switch to Circler mode. This way it will gradually expand our territory by forming a cluster of circles, with each cell occupying one circle to its maximum efficiency. It achieved good performance result in our testing for 1v1 matches, but it has some problems when it was put into a 9 group battle royal.

The weakness of any starting occupier strategy is that it’s heavily dependent on spawning conditions. For example if the player spawning next to you has some strange behaviors and keeps attacking you from the beginning, the growth of the cluster will be impeded. The other problem is the initial density. If given a large n, it is very likely our cells will be close to each other at the beginning of the game, and therefore cannot effectively circle or maintain the minimum distance.

Overall Strategy Evolution

1 Scout first, circle later

Intuitions:

One hypothesis we had at the beginning of the project is that we want our cells to be Scouts as much as they can in the earlier stages of the game, since there are many empty spaces on the board and we can grow ourselves at exponential rate unimpeded, and at later stage of the game we want our cells to occupy as much space as possible since there are few empty spaces left to occupy easily.

Therefore we defined our initial strategy to be the following:
The cells are Scouts by default, and they will start circling if and only if there are ally cells nearby. This way our cells can spread out as much as they can in the initial stage of the game, and when the board become sufficiently crowded they will naturally encounter the cells from the same species, and start occupying.

Performance:

This strategy was proven to be enormously successful, with our in class demo using the default parameters having a score 3-4 times better than the second place.

2 Pure Scouts

Intuitions:

After the first implementation, we mostly inherited the “spread out first, occupy later” strategy. Additionally, we found out that if our Scouts work well, in the later stage our cells will automatically form clusters and occupy spaces around them, so it’s not required to use occupiers in a later stage of the game. Also we found that circlers tend to collude with each other often, so we removed them from our strategies temporarily.

Performance:

Since most groups are using similar strategies, our advantage became less obvious. However overall we still remained in the top third part of the performance distribution overall.

3 Occupiers from the start

Intuitions:

During the middle of our 6 week project period, most teams are adopting the strategy of going towards the largest free angle within vision. We hypothesized that this will benefit a strategy that occupies the space from the beginning, since if we can start from a small area and gradually expand our territory in the right way, since everyone is avoiding each other we can gain enough space to achieve a higher score than pure scouting strategies.

Performance:

We never put this strategy into demonstration since its performance never met with our expectation. There are two big issues with this strategy.

One is that since everyone else is growing at exponential rate, the empty space left on the board will be removed in very small amount of time. According to our estimate, around score 256 the entire board will be mostly filled, leaving only small spaces between cells to be occupied. If the occupier cannot establish a size around 400 territory at that time, it cannot beat pure scouts since there will be no more space to occupy.

The other issue is that it is very difficult to coordinate cells by a globally optimal behavior. Since we are in a giant cluster, the outer cells need to realize they are on the perimeter and they need to expand, the inner cells need to realize they are within friendly territory and try to avoid collusion with their own species as much as they can. We implemented a simple version of this, however the cells would end up maintaining the distance up to a certain point, when the cluster gets too big to manage all the inner cells will crowd each other out and stop growing. Later in the project period some group achieved this behavior unintentionally by having the cells all circling towards the same direction, but it would not solve the expanding territory issue for the outer cells.

4 Pure Scout improved/Hybrid Occupier

Intuitions:

Our final submission includes the improved version of pure scout. We basically implemented the same “scout first, occupy later” strategy as first week, but we added a few improvements to the scout (aka finding the largest open angle) player. We also programmed the cells to go towards enemy cells if there are too many ally cells, therefore achieving the same effect as Circler as an occupier but with better result.

We also tried a different occupier from start strategy, where each cell would go in circle from the beginning, but would send out scouts with a certain probability to explore unoccupied spaces by our cells. Although this strategy has better results than pure scout in 1v1 matches, it is less robust against the randomness in spawning number/locations, therefore we did not include it in our final submission.

Performance:

Our player achieved the ideal result in the tournament (see tournament performance analysis section).

Other improvements

In the end we use the aggressive scout as our main strategy. As everyone would use similar strategy, to distinguish us from other players, we need some magical tricks.

1 Useful tricks

First trick is mentioned before, we limit our vision and then decrease our vision if we cannot find any free angle nearby. It makes our performance more consistent in different parameters.

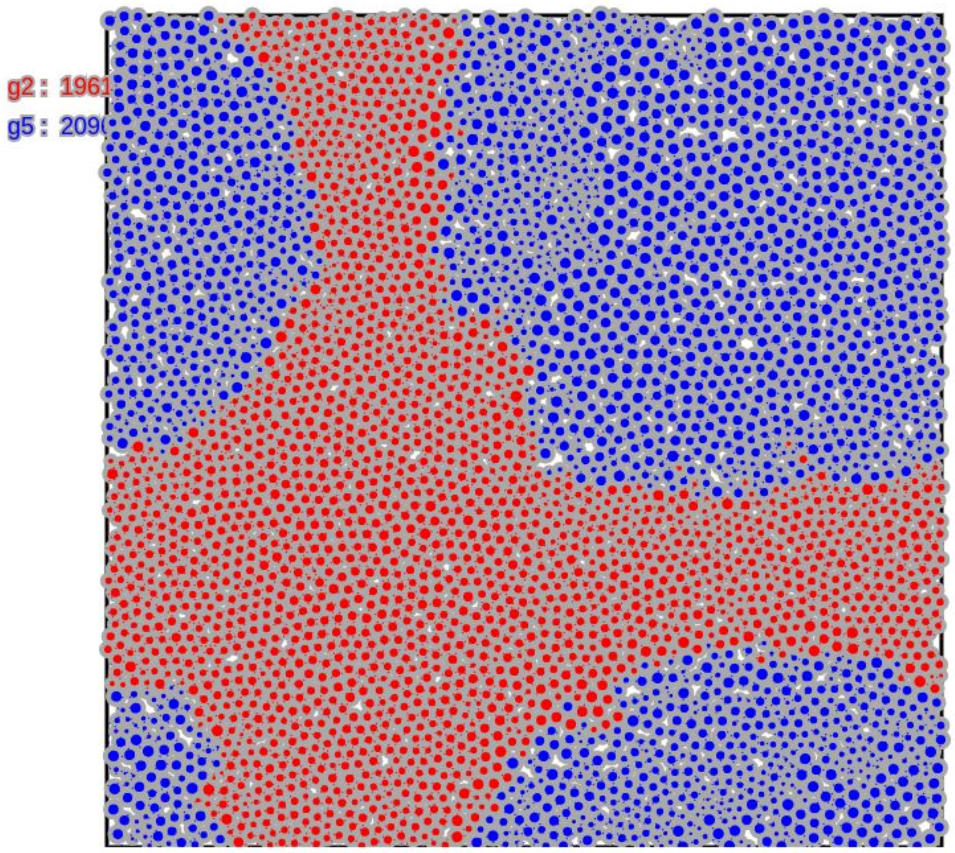
Second trick is, if we cannot move along the direction by 1mm, we try to progressively decrease our step size. We believe that, not adding this trick is the main reason why we lose to group 9 when we showed up using the similar strategy at that week. It’s very important for the aggressive scout during the “aggressive phase”, as a slightly increase on the radius leads to a large area expansion.

We believe that the third trick is what help us win the final tournament. This trick is, after a cell moves, we want a small gap between it and our own cells so that both of them would have space to grow. To implement this, we just regard our own cells as ones with slightly larger diameter and reject all moves that will cause a collision between our own cells. The consequence of this trick is amazing. First we could maintain a larger gap between our own cells so that they can occupy a larger area. Second, we can utilize the space more effectively than others. As our experiments shows that, compare to the same strategy without this trick, most of our own cells could have one more round of reproduction which gives a huge leads in the final score.

2 Clustering

As any practical strategy results into the scenario that every player forms some clusters on the map. We should know what kind of clusters is better.

If we believe that the ideal solution is to condense our cell near the perimeter. It’s better that we should have only one giant circular cluster because it has the best perimeter/area ratio. You could use the least number cells to occupy a large area. This is what group 5 did in their final submission:



They use “move when necessary” strategy to achieve this solution. The performance is pretty good in 1-vs-1 matches. But we can’t say this is a good strategy in 9 groups competition for the following reason:

- It highly depends on the initiation. Although it’s not a problem in 1-vs-1 matches. But when there are 9 players on the board, it requires that your first cells should be born on some “less crowded” spaces.
- It’s not robust against different kinds of strategies. When there are more players, the possibility of “irrational players” should be considered. It’s possible that some players would stay at some place near you to block you from growing, or even constantly attack you. Then other players can outperform you.
- It’s more important to figure out how to let your cells “effectively occupy” the space.

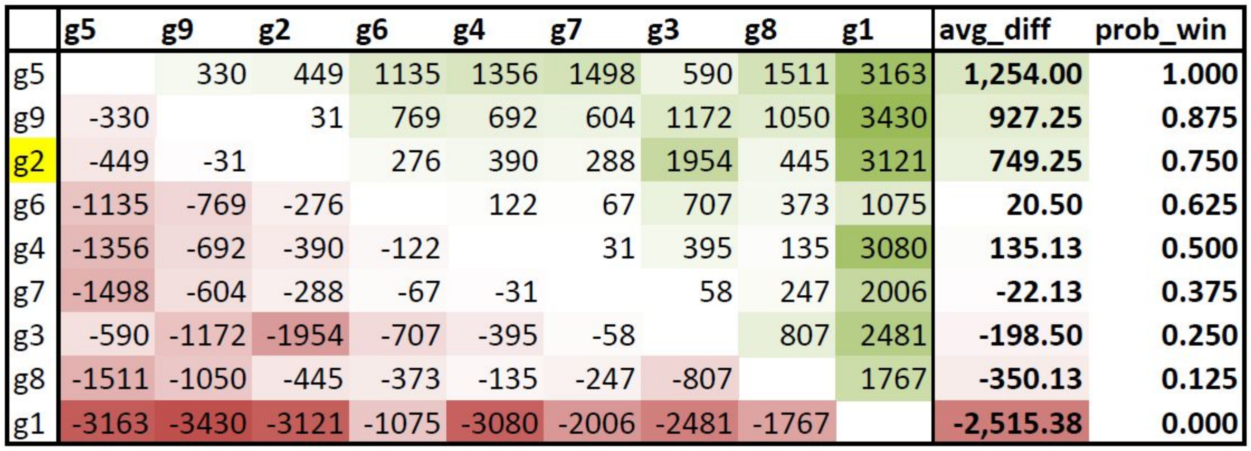
Based on these reasons, we choose not to switch to this strategy with a single cluster. Especially first two reasons encourage you to “find a nicer place” during the first phase.

Tournament Performance Analysis

Because we chose a strategy focused more on all group tournaments, our performance was not as good in the pairwise tournament. However judging by number of games won, we can still be within the top 2 players among 9 groups, only losing to g5, with g9 having a similar performance as us.

However, we can see our strategy is robust against all parameters. Judging by the average rank of the game, we are first in overall category, and first in 4 out of 6 games, second in the remaining two. Because our pure Scout strategy does not depend on any specific d or t, we were available to achieve steady performance in all scenarios.

Pairwise tournament, n = 1, d = 5, t = 20



	g5	g9	g2	g6	g4	g7	g3	g8	g1	avg_diff	prob_win
g5		330	449	1135	1356	1498	590	1511	3163	**1,254.00**	**1.000**
g9	-330		31	769	692	604	1172	1050	3430	**927.25**	**0.875**
g2	-449	-31		276	390	288	1954	445	3121	**749.25**	**0.750**
g6	-1135	-769	-276		122	67	707	373	1075	**20.50**	**0.625**
g4	-1356	-692	-390	-122		31	395	135	3080	**135.13**	**0.500**
g7	-1498	-604	-288	-67	-31		58	247	2006	**-22.13**	**0.375**
g3	-590	-1172	-1954	-707	-395	-58		807	2481	**-198.50**	**0.250**
g8	-1511	-1050	-445	-373	-135	-247	-807		1767	**-350.13**	**0.125**
g1	-3163	-3430	-3121	-1075	-3080	-2006	-2481	-1767		**-2,515.38**	**0.000**

Pairwise tournament, n = 2, d = 2, t = 10

	g5	g2	g9	g4	g6	g7	g3	g1	g8	avg_diff	prob_win
g5		90	87	4066	399	572	1272	1172	1522	**1,147.50**	**1.000**
g2	-90		39	94	227	214	1297	1055	1155	**498.88**	**0.875**
g9	-87	-39		111	422	147	1548	1460	1516	**634.75**	**0.750**
g4	-4066	-94	-111		135	-4093	1295	919	748	**-658.38**	**0.500**
g6	-399	-227	-422	-135		147	290	749	795	**99.75**	**0.500**
g7	-572	-214	-147	4093	-147		918	602	856	**673.63**	**0.500**
g3	-1272	-1297	-1548	-1295	-290	-918		1131	490	**-624.88**	**0.250**
g1	-1172	-1055	-1460	-919	-749	-602	-1131		448	**-830.00**	**0.125**
g8	-1522	-1155	-1516	-748	-795	-856	-490	-448		**-941.25**	**0.000**

All groups tournament

config	g1	g2	g3	g4	g5	g6	g7	g8	g9	avg_config
"n = 1, d = 1, t = 2"	305	499	326	505	434	380	333	274	490	**394**
"n = 1, d = 10, t = 1"	32	532	262	473	383	449	581	259	454	**381**
"n = 1, d = 4, t = 4"	172	607	149	518	477	448	602	304	507	**420**
"n = 20, d = 2, t = 15"	200	679	148	541	447	455	377	222	548	**402**
"n = 63, d = 2, t = 10"	164	606	89	542	309	384	295	120	483	**332**
Average For Player	**181**	**587**	**192**	**517**	**411**	**422**	**432**	**235**	**498**	**386**

config	group\rank	1	2	3	4	5	6	7	8	9	w_rank
all	g2	25	15	7	1						1.67
all	g4	6	17	13	5	5	1			1	2.90
all	g9	5	8	17	10	5	3				3.23
all	g7	9	4	3	3	6	17	4	1	1	4.48
all	g6			3	24	15	4	2			4.54
all	g5	4	3	6	5	12	16	1		1	4.58
all	g8					2	1	22	17	6	7.50
all	g1					1	2	13	20	12	7.83
all	g3					1	4	7	9	27	8.19
"n = 1, d = 1, t = 2"	g4	3	4	1	2						2.20
"n = 1, d = 1, t = 2"	g2	2	4	3	1						2.30
"n = 1, d = 1, t = 2"	g9	2	1	4	3						2.80
"n = 1, d = 1, t = 2"	g5	3	1	2	2		1			1	3.40
"n = 1, d = 1, t = 2"	g6				2	5	1	2			5.30
"n = 1, d = 1, t = 2"	g3					1	4	3	1	1	6.70
"n = 1, d = 1, t = 2"	g7					1	3	4	1	1	6.80
"n = 1, d = 1, t = 2"	g1					1	1	2	5	1	7.40
"n = 1, d = 1, t = 2"	g8					2			2	6	8.00
"n = 1, d = 10, t = 1"	g7	5	2	1							1.50
"n = 1, d = 10, t = 1"	g2	2	6								1.75
"n = 1, d = 10, t = 1"	g4	1		3	2	2					3.50
"n = 1, d = 10, t = 1"	g6			1	4	3					4.25
"n = 1, d = 10, t = 1"	g9			3	2	1	2				4.25
"n = 1, d = 10, t = 1"	g5					2	6				5.75
"n = 1, d = 10, t = 1"	g3							4	4		7.50
"n = 1, d = 10, t = 1"	g8							4	4		7.50
"n = 1, d = 10, t = 1"	g1									8	9.00

config	group\rank	1	2	3	4	5	6	7	8	9	w_rank
"n = 1, d = 4, t = 4"	g2	5	2	3							1.80
"n = 1, d = 4, t = 4"	g7	4	2	2	2						2.20
"n = 1, d = 4, t = 4"	g4		1	4	1	3	1				3.90
"n = 1, d = 4, t = 4"	g9	1	2		2	4	1				3.90
"n = 1, d = 4, t = 4"	g5	1	2	1	1		4	1			4.30
"n = 1, d = 4, t = 4"	g6			1	3	3	3				4.80
"n = 1, d = 4, t = 4"	g8						1	9			6.90
"n = 1, d = 4, t = 4"	g1								7	3	8.30
"n = 1, d = 4, t = 4"	g3								3	7	8.70
"n = 20, d = 2, t = 15"	g2	8	1	1							1.30
"n = 20, d = 2, t = 15"	g9	1	4	3	2						2.60
"n = 20, d = 2, t = 15"	g4	1	5	3						1	2.90
"n = 20, d = 2, t = 15"	g5			3	2	4	1				4.30
"n = 20, d = 2, t = 15"	g6				7	3					4.30
"n = 20, d = 2, t = 15"	g7					2	8				5.80
"n = 20, d = 2, t = 15"	g8							9	1		7.10
"n = 20, d = 2, t = 15"	g1						1	1	8		7.70
"n = 20, d = 2, t = 15"	g3								1	9	8.90
"n = 63, d = 2, t = 10"	g2	8	2								1.20
"n = 63, d = 2, t = 10"	g4	1	7	2							2.10
"n = 63, d = 2, t = 10"	g9	1	1	7	1						2.80
"n = 63, d = 2, t = 10"	g6			1	8	1					4.00
"n = 63, d = 2, t = 10"	g5					6	4				5.40
"n = 63, d = 2, t = 10"	g7				1	3	6				5.50
"n = 63, d = 2, t = 10"	g1							10			7.00
"n = 63, d = 2, t = 10"	g8								10		8.00
"n = 63, d = 2, t = 10"	g3									10	9.00