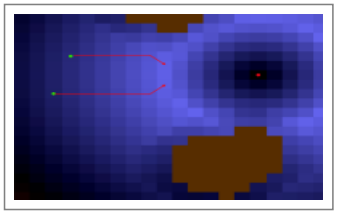
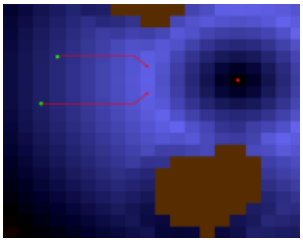
TUTORIAL



Using Potential Fields in a Real-time Strategy Game Scenario (Tutorial)

Johan Hagelbäck on January 31, 2009

Editor's note: This in-depth article was submitted by *Johan Hagelbäck*, lecturer and Ph.D. researcher in adaptive game AI. If you'd like to join Johan and myself (Alex J. Champandard) for a **live online masterclass** (public) on Wednesday, February 4th at 21:00 CET Europe, 15:00 East Coast, then go to live.aigamedev.com for more details. You'll learn about the key concepts in this tutorial, but also learn how potential fields fit in with high-level AI strategy, dynamic navigation in crowds, and optimizing traditional pathfinding. (These sessions are normally for members only, so don't miss it!)



Bots for RTS games may be very challenging to implement. The bot controls an often very large number of units that will have to navigate in a large dynamic game world, while at the same time avoiding each other, searching for enemies, defending own bases, and coordinating attacks to hunt the enemy down. RTS games operate in real-time which can make planning and navigation difficult to handle.

This is a tutorial about an unconventional planning and navigation method that uses multi-agent potential fields. It is based on the work we have presented in [1, 2, 3]. (See the bottom of this page for the white papers references, and links to the corresponding PDF files for download.)

What is a Potential Field?

PF's has some similarities with influence maps. Influence maps are often used to decide whether an area in the game world is controlled by own or enemy units, or if it is an area currently not under control by any forces (no man's land). The idea is to put a positive numerical value on the tiles currently under control by own units, and a negative numerical value on the tiles controlled by the enemy. By letting the numerical values gradually fade to zero we can create a map displaying the influence own and enemy units has in an area. Figure 1 shows an example of an influence map with one own and one enemy unit.

Content Index

0	2	6	2	0	0	0	0	0
2	6	8	6	2	0	0	0	0
6	8	10	8	6	2	0	0	0
2	6	8	6	2	0	-2	0	0
0	2	6	2	0	-2	-6	-2	0
0	0	2	0	-2	-6	-8	-6	-2
0	0	0	-2	-6	-8	-10	-8	-6
0	0	0	0	-2	-6	-8	-6	-2

Figure 1: Influence map showing the areas occupied by own units (red) and enemy units (blue). White areas are no man's land.

Potential fields work in a similar manner. The idea is to put a charge at an interesting position in the game world and let the charge generate a field that gradually fades to zero. The charge can be attracting (positive) or repelling (negative). Note that in some literature about potential fields negative charges are attractive and vice versa. In this tutorial positive charges are always attractive. Figure 2a-c show a simple game world with some impassable terrain (brown), an enemy unit (green) and a goal destination for the unit (E). An attractive charge is placed in the destination position (figure 2a), the charge generates a field that spreads in the game world (figure 2b) and fades to zero (figure 2c).

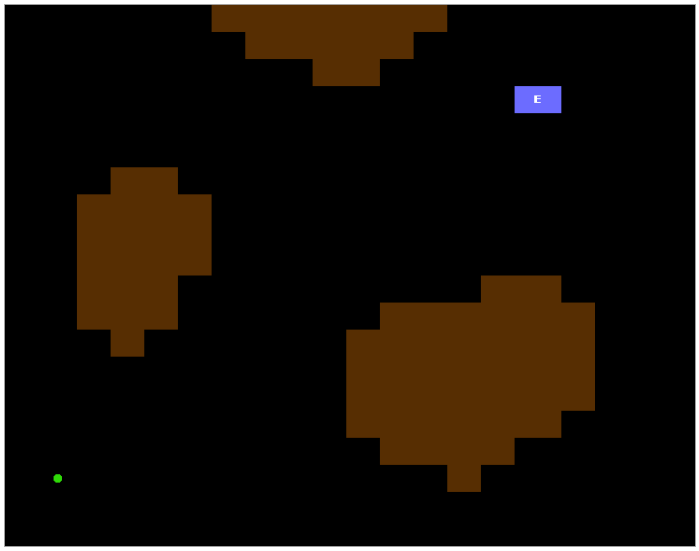


Figure 2a: An attractive charge (E) is placed in the destination for an own unit (green).

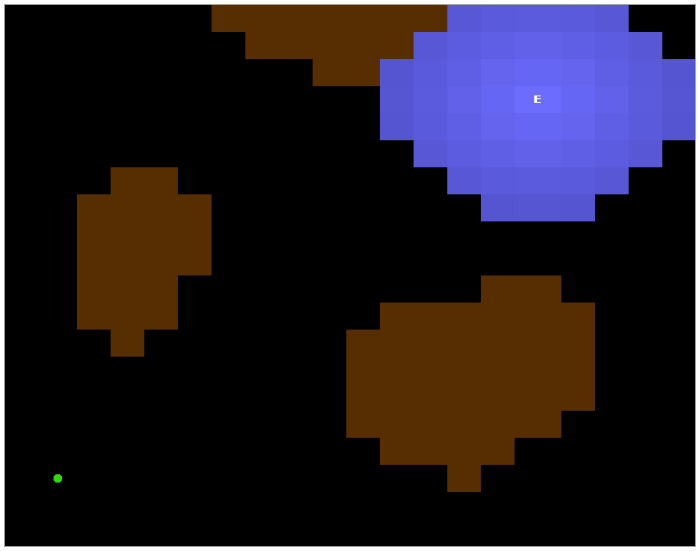


Figure 2b: The charge generates a field that spreads in the game world. Light blue areas are more attractive than darker blue areas.

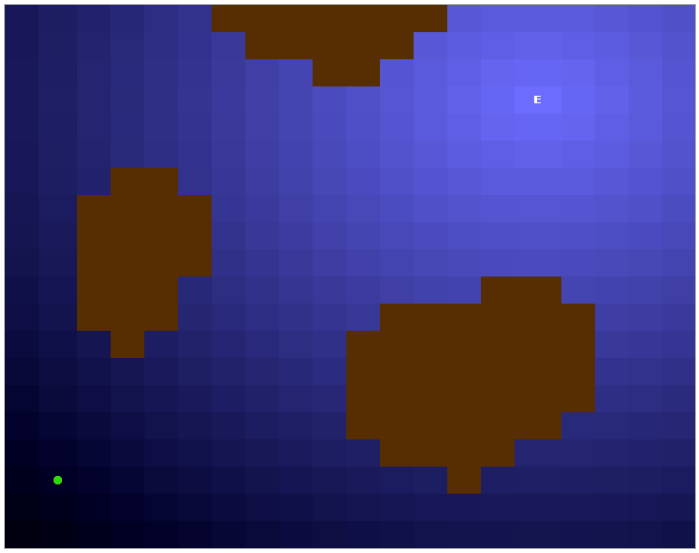


Figure 2c: The field have spread throughout the game world and gradually fades to zero.

Navigation using Potential Fields

In Figure 2c an attractive field is spread around the destination point E. The idea is to let the green unit move to positions with a more attractive value and in the end find a way to its destination. To make this work we also need to deal with obstacles in the game world, in this case the mountains (brown areas). If we let the mountains generate small repelling fields, and we sum the fields with the attractive field from Figure 2c we get a total field which can be used for navigation (Figure 3). Since units always move to the adjacent position with the highest attractive value it will avoid mountains if other paths are available.

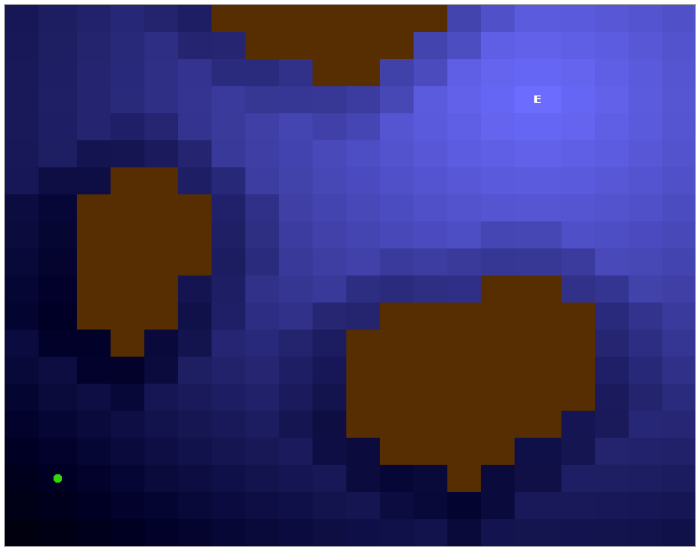


Figure 3: Repelling fields generated by mountains has been added.

The same idea is used for other obstacles. In Figure 4 two own units (white) are added. They generate small repelling fields which again are summed to the total field. Our green unit will then move around other units to avoid collisions. Now we have a final path for our unit (figure 5).

The unit can navigate from its current position to a destination and avoid obstacles without using any form of pathfinding algorithm.

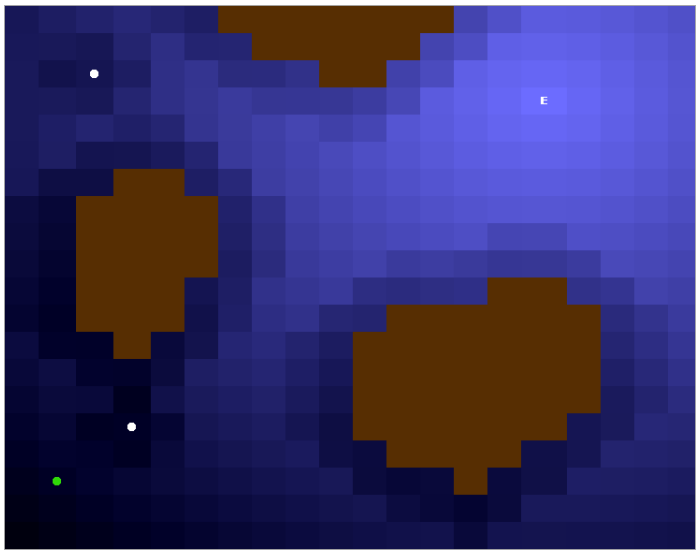


Figure 4: Two own units added. They generate repelling fields for obstacle avoidance.

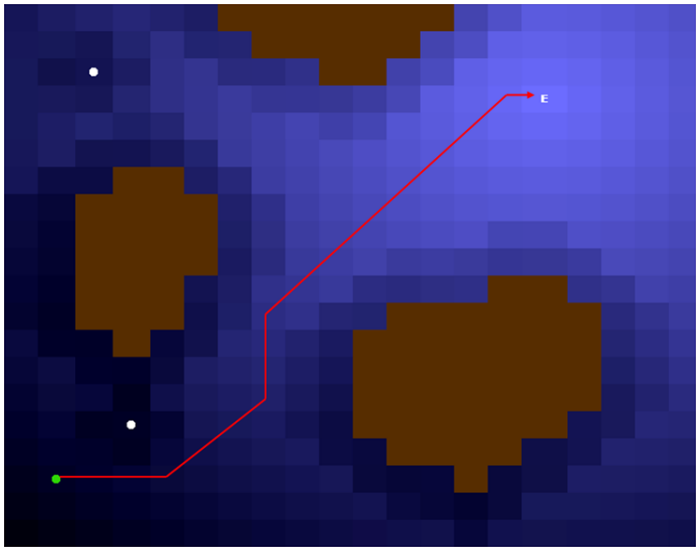


Figure 5: The final path for our unit (green).

Benefits of Using Potential Fields

One of the most important advantages of using potential fields is its ability to handle dynamic game worlds. Since units (agents) only have a one step ahead approach
instead of generating a long path from A to B there is no risk that a path becomes obsolete due to changes in the game worlds. Pathfinding in dynamic worlds can often
be complex to implement and require a lot of computational resources, and a lot of research has been done in this area. When using a potential field based solution the problem is solved automatically. Of course one need to take care when implementing the PF system to make updates of the potential fields as efficient as possible. (More about that later).

Another major advantage is the ability to create quite complex behavior by simply playing around with the shape of the generated fields. Instead of a linear field fading to 0, we suggest the use of non-linear fields. If we for example have ranged units like tanks in our army, we don't want the tanks to move too close to an enemy unit but instead surround it at an appropriate distance (for example maximum shooting range of its cannons). This behavior can be created by putting a 0 charge at the enemy unit position, generate an increasing field with the most attractive value at maximum shooting distance, and then let the field fade to 0. Figure 6 shows an example of two tanks (green) moving in to attack an enemy unit (red) generating a non-linear potential field. Equation 1 shows an example equation of how this type of field can be generated.

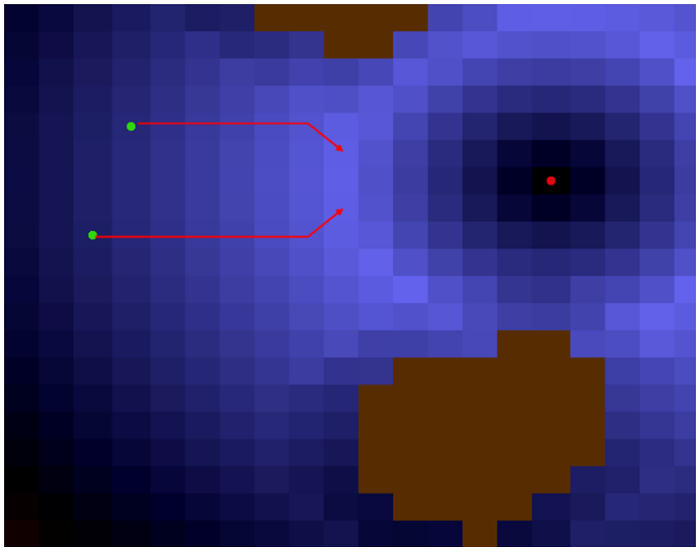


Figure 6: Example of a non-linear field generated by an enemy unit (red). Own units (green) surround the enemy at an appropriate distance.

$$p_{oppU}(d) = w_1 \cdot \begin{cases} 240/d(D-2), & \text{if } d \in [0, D-2[\\ 240, & \text{if } d \in [D-2, D] \\ 240 - 0.24(d-D) & \text{if } d \in]D, R] \end{cases}$$

Equation 1: Example of a non-linear field used to create a surrounding behavior around enemy units. The weight w1 is used to alter the relative strength of the field at runtime. D is the maximum shooting distance, and R is the maximum detection range (the distance from which an agent controlling an own unit detect an enemy unit at).

Another behavior that is simple to implement is attack-and-retreat. A unit moves to attack an enemy unit from maximum shooting range (Figure 7a). After the attack our unit have to reload its weapon and retreats from the enemy until it is ready to fire again (Figure 7b). In the retreat phase all enemy units generate strong repelling fields with the radius of maximum shooting range of the unit. This is an example of how specific potential fields can be active or inactive depending on the internal state of an agent controlling a unit. An inactive field is simply ignored when summing the total potential field.

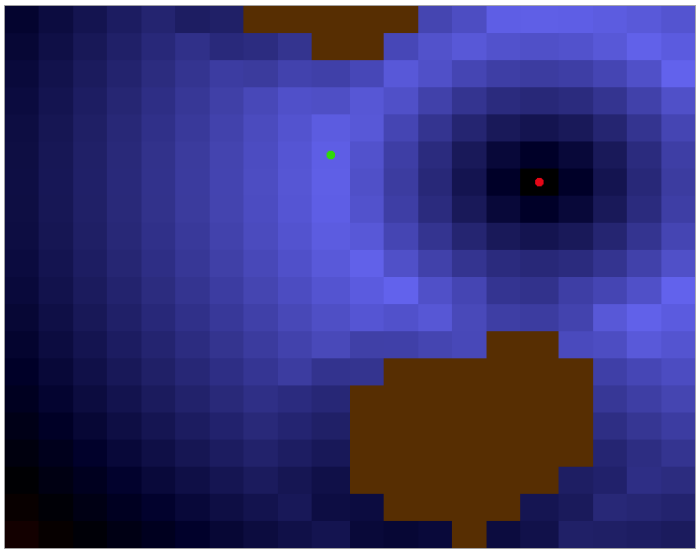


Figure 7a: A unit (green) moves in to attack an enemy unit (red) from maximum shooting range.

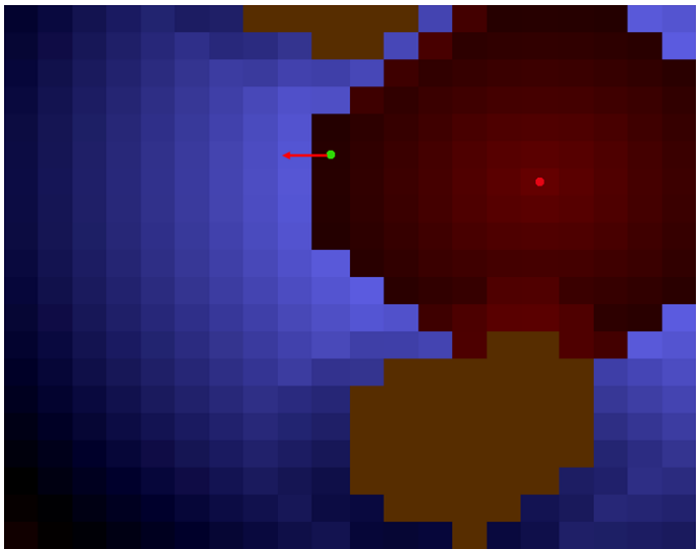


Figure 7b: After an attack the unit (green) have to reload its weapon and retreats from the enemy (red). In this phase enemy units generate a strong repelling field that overrides a part of the attractive field.

Figure 8 shows a screenshot from our potential field based bot for the ORTS engine. The left side of the figure shows a 2D view of the current game state. It shows our tanks (green circles) move in to attack enemy bases (red squares) and units (red circles). The brown/black areas are impassable terrain (mountains). The right side of the figure shows a potential field representation of the game state. As in other figures in this tutorial light blue areas are the most attractive. The white/gray lines are attacks from own tanks. The PF view clearly shows how own units surround the enemy at maximum shooting range while at the same time avoiding collisions with each other and terrain. The behavior of surrounding the enemy worked very well and was probably one of the key things behind our success in 2008 years' ORTS tournament.

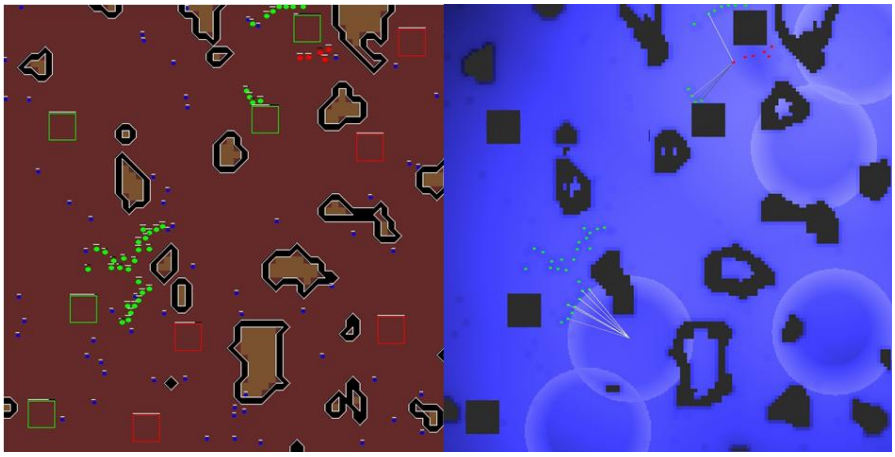


Figure 8: Screenshot from our PF based ORTS bot. Left side is a 2D view of the current game state, right side shows the potential field representation of the game state. The blue circles are neutral units called sheep. They are moving randomly around the map to make pathfinding more complex. (Click to view full size.)

One thing we discovered while developing the ORTS bot was the totally different behavior when summing the potentials enemy units generate in each point compared to just using the highest potential an enemy unit generates in a point. In Figure 9a the potentials generated by the three enemy units are summed and added to the total potential field. By doing this the highest total potential is in the center of the enemy unit cluster (shown with a red ring) and our units were very keen to attack the enemy at its strongest points. The solution was to not sum the potentials, but instead take the highest potential generated by an enemy unit in a point (Figure 9b). In the latter case the highest total potential is a front at an appropriate distance from the enemy (shown with red lines).

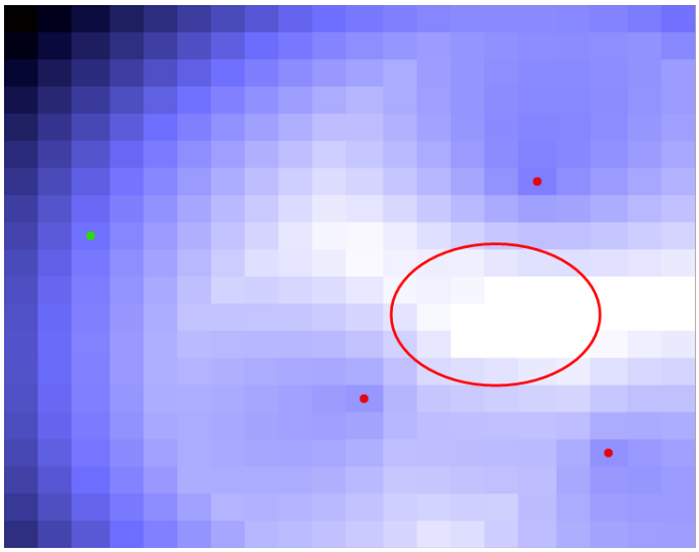


Figure 9a: The potentials generated by enemy units are summed. The most attractive region is in the center of the enemy unit cluster.

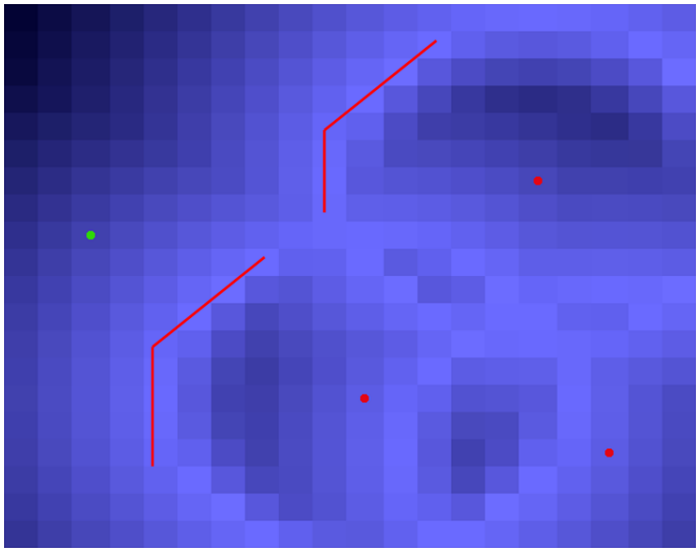


Figure 9b: Only the highest potential generated by an enemy unit is added to the total field. The most attractive region is a front surrounding the enemy.

Notes about the Implementation

With careful design and implementation of the PF system the computational resources needed does not have to exceed a traditional A* based solution. Our

ORTS bot used least CPU resources in a comparison with 2 other pathfinding based bots from 2007 years' tournament. We must however stress that it is hard to exactly measure the CPU usage due to the fact that the winning bot use more resources at the end of a game since it have more units left to control. Threading also made it complex to measure CPU resources used. The comparison were conducted by comparing the total CPU resources used be each client process in a Linux environment over 100 games. We can at least draw the conclusion that the bot was well within the margin of the frametime of 0.125 sec used in ORTS.

To improve performance we divided the potential fields into three categories:

- **Static field**. Fields that are static throughout a game, in our case the terrain. This field is genererated at startup.
- **Semi-static field**. Fields that doesn't require frequent updates. In our case own and enemy bases. This field is generated at startup and only updated if a base is destroyed.
- **Dynamic field**. All dynamic objects of the game world such as own and enemy tanks. This field is updated every game frame. For shorter frame times updates for example every 2:nd or 3:rd frame might be more suitable.

We experimented with two main architectures for generating the potential fields…

Pre-generation

The field generated by each object type was pre-calculated and stored in static matrices in a header file. At runtime the pre-generated fields were simply added to the total potential field at correct position. To make this feasible the game world had to be divided into tiles, in our case each tile consisted of 8×8 points in the game world. This approach proved not to be detailed enough and the bot performed badly (2007 years' ORTS tournament). Since the game world was divided into quite large tiles we had problems deciding which tile(s) an object occupied. Consider the unit (orange circle) and base (orange square) in figure 10. Which tile(s) (grey squares) are occupied by the green unit, and which tiles shall be used as boundaries for the base?

This approach can be suitable for games like Wargus which uses a less detailed tile-based navigation system.

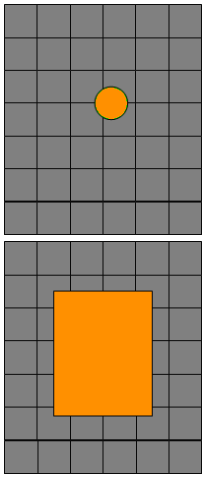


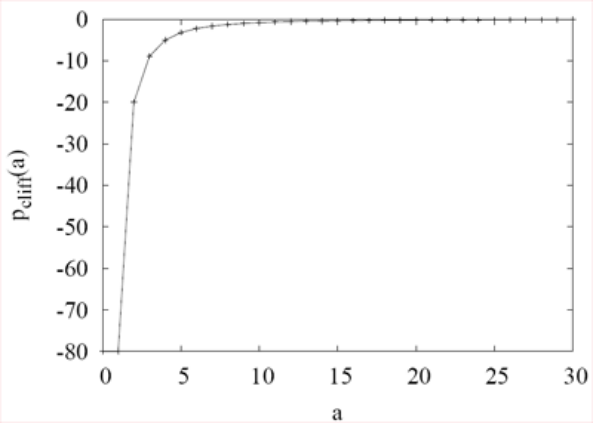
Figure 10: A unit (orange circle) and base (orange square) are placed in a grid representation of the game world. Which tile(s) are occupied by the unit and base?

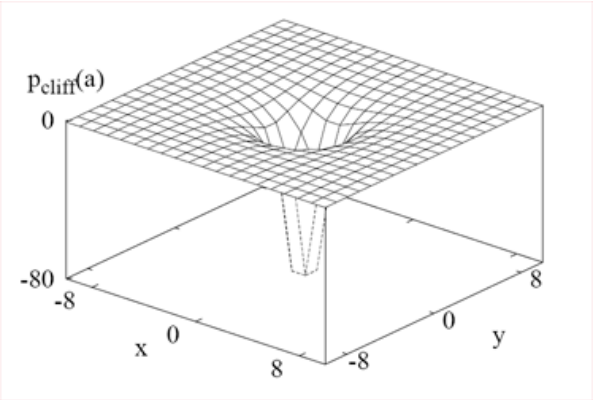
Runtime Calculation

The potentials generated in a point are calculated at runtime by calculating the distance between the point and all nearby objects, pass each distance as a variable to a function and then calculate the generated potentials. This approach actually used less CPU resources in our bot than pre-generated fields due to:

- We only calculated potentials in points that are candidates to move to by unit(s). In the pre-generated approach the potentials for the whole game world were
  calculated (of course this can be optimized in the same manner).
- By first estimating the distance to an object with a quick Manhattan distance calculation, we could skip costly Euclidean distance and potential calculations for a large portion of the objects in the game world.

This approach was used in the second version of our bot (2008 years' ORTS tournament). Equation 2 shows an example of a potential field equation (with 2D and 3D graphs) used to calculate the potential a cliff generates in a point with the distance *a* from the cliff.





$$p_{cliff}(a) = \begin{cases} -80/a^2 & \text{if } a > 0 \\ -80 & \text{if } a = 0 \end{cases}$$

Equation 2: The equation (with 2D and 3D graphs) used in our bot to calculate the potential generated by cliffs at distance *a*.

What about the Local Optima Problem?

One of the most common issues with potential fields is the *local optima problem*. Figure 11a shows an example where this problem arises. The destination E generates a circular field that is blocked by a mountain. The unit (green) move to positions with higher potentials, but end up in a position (as shown in the figure) where the highest potential is where the unit currently stands. The unit hence get stuck.

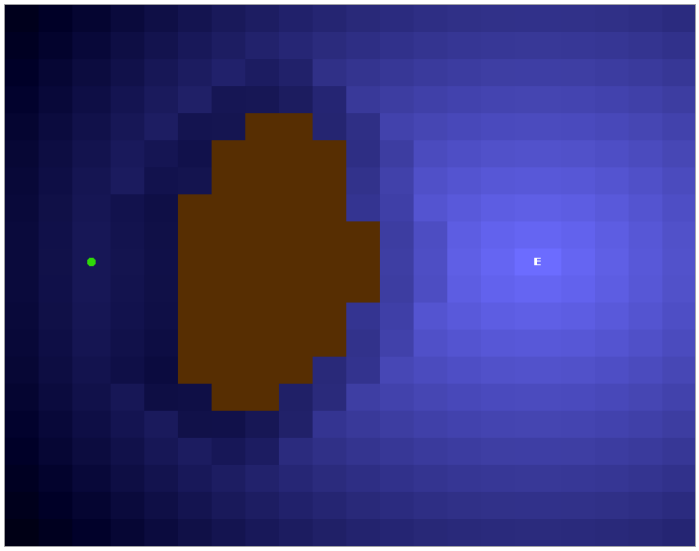


Figure 11a: The local optima problem. The potential field generated by E is blocked by a mountain and the unit (green) is unable to reach its destination.

We solved this by using a trail similar to a pheromone trail as described by Thurau et al. in [4]. Each agent controlling a unit adds a trail at the last **n** positions visited by the unit, as well as the current position of the unit. The trail generates slightly repelling potentials and "pushes" the unit forward. Figure 11b shows how the trail pushes the unit (green) around the local optima (yellow path) and the unit are able to find a path to its destination.

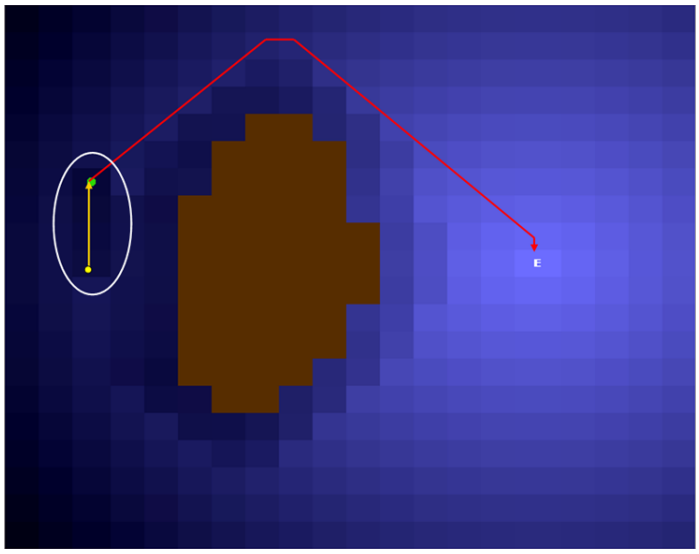


Figure 11b: The trail (yellow) pushes the unit around a local optima.

Even with trails there is a risk that units get stuck in situations as shown in Figure 12a. This can be solved with convex filling of gaps (see Figure 12b).

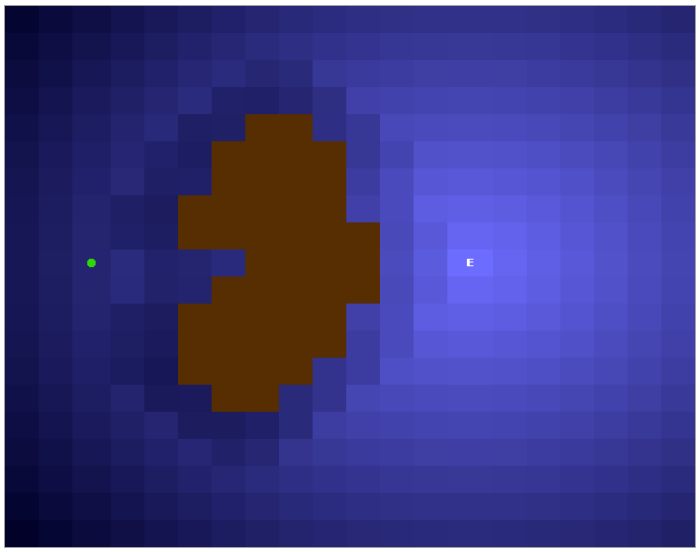


Figure 12a: A unit (green) can get stuck in the gap in the mountain.

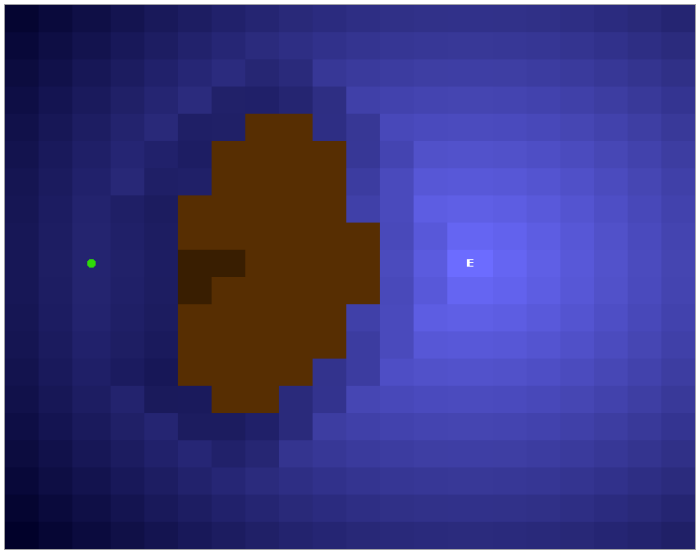


Figure 12b: Convex filling can solve this problem.

Summary

Below we list a number of criticisms against potential field based solutions and our point of view:

- **PF's have issues like local optima that are difficult to solve**. With the use of trails most local optima can be solved. PF's still have problems in very complex terrain, but in those cases pathfinding based methods are better suited. The strength of PF's are in large dynamic worlds with large open areas and less complicated terrain. This is the case for many RTS games of today.

- **PF based solutions use too much resources**. We have in our work shown that PF based solutions can be implemented with the same or better resource efficiency as pathfinding based methods. More work in other environments has to be done to back up this statement. Efficiency can still be a problem, but that can often be the case in pathfinding based systems as well.

- **PF based solutions are difficult to implement and tune**. PF's can be implemented with very simple architectures. Tuning can be difficult and time consuming, but the relative importance between the fields is a great help here (i.e. is destroying a base more important than destroying units?). A graphical representation of the potential field view as the one shown in Figure 8 is also valuable. There is however a need for better tools and methodologies to aid in this matter.

After my talk on AIIDE 2008 I got some interesting comments and ideas. Brian Schwab pointed out the possibility of using the GPU to further improve the performance of field generation. I have not been able to investigate this in more detail mostly due to my lack of knowledge in GPU programming. If anyone knows about an easy framework for this feel free to contact me. Chris Darken came up with an idea of combining pathfinding and potential fields to benefit from both worlds. Pathfinding could for example be used to move units over large distances, while potential fields could be used when engaging the enemy. The idea sounds interesting and I hope I'll be able to try it out soon.

We believe that potential field based solutions can be a successful option to more conventional pathfinding based solutions. After a mediocre ORTS tournament 2007, some tactical and technical problems were addressed and solved and a new bot version was created. In experiments this bot won over 99% of the games against the four top teams from 2007 years' ORTS tournament (see Table 1). The bot also won 98% of the games in 2008 years' tournament, but we must point out that the number of participants was very low. This was quite an improvement from ending at 6:th place of 9 participants in 2007. The results from 2008 can be found on the tournament webpage.

Opponent Team	Win % (100 games)	Avg units left (of 50)	Avg bases left (of 5)
NUS	100%	28.05	3.62
WarsawB	99%	31.82	3.21
UBC	98%	33.19	2.84
Uofa.06	100%	33.19	4.22
Average	99.25%	31.56	3.47

Table 1: Experiment results from our PF based bot against the four top teams from 2007 years' ORTS tournament. Our bot won 99.25% of the games with an average of 31.56 units left after each game (both teams start with 50 units) and 3.47 bases (both teams start with 5 bases).

In a working paper we show that PF based bots can handle complex scenarios such as a full RTS game including fog-of-war, resource gathering, exploration, base building and a simple tech tree. The work might be a scope for a second part of this tutorial. We have also experimented with runtime difficulty scaling and tests conducted by human players.

We are interested in investigating PF based approaches in other RTS environments and we have done some initial testing on the Wargus platform. I gladly accept suggestions on other platforms.

References

Using Multi-agent Potential Fields in Real-time Strategy Games
Johan Hagelbäck and Stefan J. Johansson
International Conference on Autonomous Agents and Multi-agent Systems (AAMAS), 2008.
1. **Download PDF**

The Rise of Potential Fields in Real Time Strategy Bots
Johan Hagelbäck and Stefan J. Johansson.
Proceedings of Artificial Intelligence and Interactive Digital Entertainment (AIIDE), 2008.
2. **Download PDF**

A Multiagent Potential Field-Based Bot for Real-Time Strategy Games
Johan Hagelbäck and Stefan J. Johansson
International Journal of Computer Games Technology, 2009.
3. **Download PDF**

Learning Human-like Movement Behavior for Computer Games
C. Thurau, C. Bauckhage, and G. Sagerer
International Conference on the Simulation of Adaptive Behavior (SAB), 2004.
4. **Download PDF**

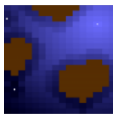
About the Author

Johan Hagelbäck is a Ph.D. student and lecturer in adaptive game AI at Blekinge Institute of Technology, Sweden. You can visit his webpage or contact him at jhg[at]bth.se.

Editor's note: If you'd like to join Johan and myself (Alex J. Champandard) for a **live online masterclass** (public) on Wednesday, February 4th at 21:00 CET Europe, 15:00 East Coast, then go to live.aigamedev.com for more details. You'll not only learn about the practical details behind this article, but also how potential fields fit in with high-level AI strategy, dynamic navigation in crowds, and optimizing traditional pathfinding. Similar sessions are run weekly as part of AiGameDev.com's membership site, so don't miss this free event.

Feel free to also post questions or feedback in the comments below.

Related Articles


Advanced Potential Fields in Practice for Efficient RTS Bots


DEMIGOD's AI from Role-Playing to Real-Time Strategy


Reinforcement Learning in Real-Time Strategy Games Using Case-based

Discussion 8 Comments

William on February 1st, 2009

Hi, interesting revival of this technique! With your experience in the ORTS competition and looking towards use in consumer games, do you feel a need for: - any kind of damping (if the AI continuously attempts to move into the ideal attack spot, any move by an AI can cause a chain of reactions) - other shapes than circles for the attractors/repulsors, for example to handle line-of-sight or static guns that only fire in a certain direction? Furthermore, how would I be able to implement flanking behavior using potential fields? Chris Crawford introduced virtual objectives/attractors between the unit's position and unit's primary objective/attractor and laterally offset these virtual objectives (in his 1982 Tanktics game) to achieve this effect. Would that work here as well?

PaulTozour on February 3rd, 2009

Wow, this is a fantastic article! This is a very practical approach and I appreciate that you list the shortcomings of potential fields and the situations where they're really useful.

d4nte on February 3rd, 2009

Hi, this is a great article ! I am currently working on a RTS, and all these explications will be very heplful. Thanks

red15 on February 4th, 2009

This is pretty much the same technique as using pathfinding imo, where A* pathfinding is even better equipped to deal with non-convex obstacles this PF method requires "hacks" to navigate around them. I guess the cost of this algorithm for a large number of agents would be less than pathfinding each of them. One solution to this is making pathfinding a de-synced process for each agent. So instead of providing all agents a complete path to the point of interest as soon as the point originates you would queue (preferably multi threaded) each agent to look for a path when cpu time is available. This would create a perhaps wanted side effect of similar kind to Darwinia where each agent responds at a slightly different offset instead of a cross-map instantaneous snap to navigate to the point.

alexjc on February 4th, 2009

Red15, It's more than regular shortest-path A* pathfinding. The potential field gives you a sense of strategy (like an influence map does) that A* could not take into account on its own. But it's a good point, we'll discuss it in the masterclass. Alex

johanhagelback on February 4th, 2009

Besides the sense of strategy, potential fields have another major difference from A*. A* takes a "snapshot" of the current game world state, plans a path from A to B and then executes the path. This takes time and if the game world changes the agent must replan its path. Navigation using potential fields is a one-step lookahead. The agent know, by examining the fields, where to go next but has no complete path from A to B to follow. A PF based solution thus directly adapts to changes in the game world. This has some similarities with Darwinia's use of routing nodes. The agent, or rather the node the agent visits, does not know the complete path to the destination but know where to go next to in the end reach the destination.

johanhagelback on February 12th, 2009

The complexity depends on a lot of things. In general you can say it works like: foreach own unit U foreach reachable position P around U foreach game object O generating a field distance d = distance between P and O total potential pt += potential(P,O) end end end This can be optimized in a lot of ways. For example we skip calculating exact distance and potential for game objects located far away which has none or very low influence on the current position. An own unit might also be busy doing something else, for example attacking the enemy, and therefore no potentials around that object are calculated. Reference paper 2 is a good start if you want to dig into more details about the algorithm.

nahelzym on September 4th, 2013

It was really inspiring lecture! I've implemented Johan's ideas with my [URL="https://github.com/ncreated/Potential-Fields"]Potential Fields AS3 library[/URL]. I've also written an article about implementation details and published some demos on my blog (links in the library repository).

If you'd like to add a comment or question on this page, simply log-in to the site. You can create an account from the sign-up page if necessary... It takes less than a minute!