Trouble in Gaussian City: When Laser-Tanks use Potential Fields

Kian Kenyon-Dean
260564475

April 11, 2017

COMP 521 Final Project

Prof. Clark Verbrugge

McGill University

1 Introduction

In this project we explore the use of artificial potential fields to create group pathfinding strategies for groups of enemy artificial intelligence agents. In the context of a procedurally generated city environment, we find that these multi-agent potential fields allow for advanced, strategic behaviors to be implemented implicitly via a set of potential generating functions. For example, one implicitly implemented behavior causes enemy units to strategically surround the player in a circular formation according to their laser-gun's shooting range, giving the impression that the agents are communicating with each other to "set up a perimeter".

In our implementation we address the well-known drawbacks of potential fields (especially the problem of local optima) with simple, effective solutions inspired by [1]. Additionally, we observe the following benefits of using potential fields, benefits which are not found in other multi-agent pathfinding algorithms:

i. Dynamic, rapid, group pathfinding around obstacles that does not rely on complex engineering (unlike, for example, hierarchical A* or reservation-based group pathfinding approaches);

ii. Strategic collective behaviors are exhibited through simple functions of generated potentials;

iii. A straightforward, cheap, extension from a discrete implementation to a continuous implementation, which is not normally the case for standard pathfinding algorithms, particularly for those based on graph theory.

It is important to emphasize the fact that potential fields are directly conducive to pathfinding in dynamic worlds where the objective is constantly changing location. This advantage is apparent because potential fields do not require any path-planning to reach their destination, they only necessitate checking the potential-values of the direct neighbors surrounding the agent. This ability to rapidly adapt behavior in a dynamic world is not easily implemented in standard pathfinding algorithms without significant engineering or a waste of computational resources due to obsolete paths, and this difficulty of other pathfinding algorithms is especially exemplified in the context of multi-agent pathfinding. Potential fields automatically and implicitly solve many of the problems faced by other pathfinding approaches, and their own problems are easily resolved by simple solutions.

In general, we propose that potential fields are an excellent tool that should be researched and pursued more rigorously in the context of video game AI, and our results encourage further exploration. In Section 2 we provide an overview of previous approaches to this problem. In Section 3 we describe the context of the game we implement and the procedurally generated environment. In Section 4 we give a precise depiction of potential fields in our discretized environment, the different variants we use for different potential generating objects in the game, and perspectives for a continuous implementation of potential fields. In Section 5 we describe the different experiments we pursued for testing the implementation. Finally, we give concluding remarks and perspectives for future work in Section 6.

2 Background

The concept of a potential field is very general and is applied across domains from gravitational physics and electricity to abstract vector mathematics on manifolds. The concept of an "artificial potential field" obtained practical popularity in the domain of robotics in 1986 [2] and has been pursued, criticized, and expanded upon throughout the domain since then. In particular, the authors in [3] found that the disadvantages of potential fields outweighed their advantages, suggesting that potential field methods should be abandoned altogether in favor of their novel approach. While potential fields methods may not be state-of-the-art in robotics, the video game world provides a very different context which turns out to be quite conducive to potential field methods.

2.1 Background of Potential Fields in Video Games

In [1, 4, 5], the authors explain that potential fields are notorious for having the following main problems in the video game context:

i. There is no guarantee that the most efficient path will be found;

ii. There is no guarantee that the agent will arrive at the desired destination as a result of getting trapped in local optima;

iii. Potential fields are computationally expensive to implement.

At first glance, these seem to be quite concerning problems since we want our pathfinding algorithms to actually work, and to work quickly. We should first note that the first problem is not of particular importance in a video game context because we do not necessarily want or need our enemy agents to find perfect paths, since that may make the experience too difficult for the player; we simply want the agent to arrive at the destination in a reasonable amount of time.

The authors above systematically address each of these problems with effective solutions. One particular solution to the well-known problem of local optima is resolved in an appealing way by using the potential field architecture already present. Namely, whenever the agent moves to a new location that is not an objective destination, create a repellent potential-generator for the agent at that point. We describe this implementation in detail in Section 4, along with several other interesting behaviors that can be exhibited via simple potential functions inspired by those presented in [1].

The advantages of potential fields are clearly observed when considering the results obtained in [1] who use potential fields to control an army of tanks in an RTS (real-time-strategy) 2008 tournament, where they won the tournament, winning 98% of games against other advanced AI implementations by other research teams. Additionally, their implementation was in a more complex environment compared to ours, where they had to handle multiple enemy bases, and, in particular, the fog-of-war. They show that parameter tuning and clever models of potential field generators allows for the behavior and experiments implemented in this project to be extended to the more advanced context of the tournament environment.

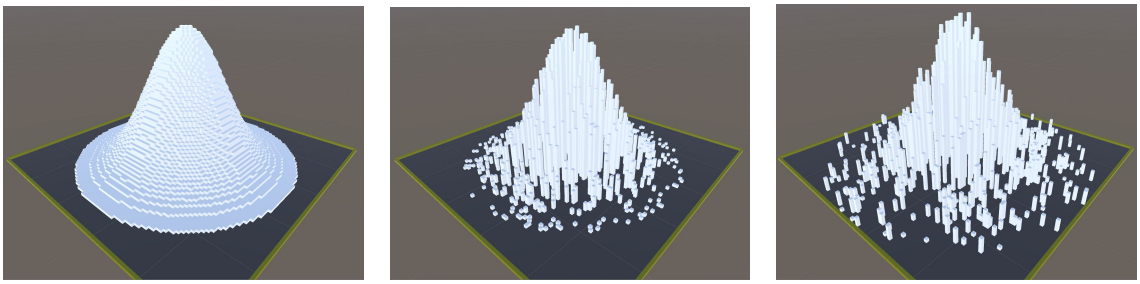
3 Game Setting

In our game, the player controls a single tank from a first person perspective, which shoots lasers out of its barrel. The player must either run away from or shoot down enemy laser-tanks whose potential-field based behavior gives the impression of a collective, strategic attack and pursuit on the player. All tanks have a shield which, when completely drained, causes the tank to be destroyed, although the shield regenerates after five seconds. The environment in which this occurs is a procedurally generated city (Gaussian City) described below in Section 3.1. The player and enemy tanks must pathfind around the buildings in this city, and the player has the freedom to destroy the buildings with her laser, although the larger buildings take longer to destroy. We assume that the enemy tanks have a complete knowledge of the game environment, each other's location, and the player's location; however, we note that the authors in [1] provide a potential-field implementation that accommodates the fog-of-war, which is beyond the scope of the current project.

3.1 Gaussian City

We construct our environment with a model based on a standard 2D Gaussian function centered at $(x_0 = 0, y_0 = 0)$. In Equation 1 we present our Gaussian function with hyper-parameters A and σ, which represent the amplitude and standard deviation of the function, respectively.

$$G(x, y, A, \sigma) = A\exp\left(- \frac{(x - x_0)^2 + (y - y_0)^2}{\sigma^2} \right) \tag{1}$$



(a) City with $p = 1$ and $v = 0$. (b) City with $p = 0.1$ and $v = 0$. (c) City with $p = 0.1$ and $v = 5$.

Figure 1: Gaussian city generated with different parameter settings, all with $A_b = 50$ and $\sigma_b = 20$.

Our city is discretized into an integer coordinate frame for the purpose of constructing the "buildings", which are implemented as simple rectangular prisms, although they could be easily extended to look like real buildings with 3D modelling software. During construction, we do not want to place a building at each coordinate, or else there would be no space to move around, so we add a probability term p which represents the probability of placing a building. Additionally, we add a random variation term v, which is the maximum value allowed for random height variation. We thus derive Equation 2 for the height h of a building at some point (x, y), with A_b and σ_b being predefined hyper-parameters for maximum building height and standard deviation, respectively:

$$h(x,y) = \begin{cases} G(x,y,A_b,\sigma_b) + \text{Random}(-v,v) & \text{probability } p \\ 0 & \text{otherwise} \end{cases} \tag{2}$$

In Figure 1 we observe the city generated according to various hyper-parameter configurations. We first observe the base function that generates the city when $p = 1$, noting its clear discretization. We also find that with the random variation $v = 5$ we obtain a random generation that looks somewhat like a real city city due to its significant variability, unlike when $v = 0$ where each outer ring is completely the same height, which does not create as interesting of an environment to explore.

For the purposes of pathfinding on the ground, it should be noted that the height of the buildings is irrelevant so long as it is greater than zero. However, this procedure creates an enjoyable environment for the player which creates a visualization of an actual randomly generated city that is tall in the center and smooths out by the outskirts. Additionally, the height of the buildings adds an important strategic element to the game since the buildings are destructible, where the larger buildings take more time to destroy.

4 Potential Fields

A potential field can be intuitively understood as a dynamically changing range of hills with a ball rolling downwards, always heading in the direction of greatest slope in order to get to the bottom as fast as possible. More formally, in our discretized representation we approach the pathfinding as an *ascent* problem (like a hiker who always wants to get to the highest point of a mountain), where a tank agent has eight different directions it can move (euclidean movement), and always moves to the neighboring point which has the greatest *potential* in order to head up the "mountain" as fast as possible.

As a dynamically changing system, we require many different functions to represent the potentials emitted by different objects and agents in the world. Our system is characterized by the following set of *cumulative potential functions*, which compute a floating point number representing the potential at a certain coordinate (x, y):

- **Semi-Static**: this function $M(x, y)$ is stored as a pre-computed 2D array of floating point values which holds the potential values of buildings and their immediate surroundings (i.e., of the map); it is semi-static because the values are infrequently modified in the occurrence of a building being destroyed (see Section 4.1.1);
- **Collective-Dynamic**: this function $C(x, y)$ dynamically computes the sum of each function belonging to a *collective potential generating object*; i.e., the sum of the potential functions for each player and each enemy tank (see Sections 4.1.2, 4.1.3);
- **Individual-Dynamic**: this function $I_T(x, y)$ dynamically computes, for an individual tank, T, the sum of each function belonging to an *individual potential generating object* for each object created by this individual tank; i.e., the sum of all of the repellent potential functions it has previously created along the path it has followed (see section 4.1.4).

Our final function, F, which computes the potential at any point (x, y), for some agent tank T, is therefore formally expressed by Equation 3:

$$F(x, y) = M(x, y) + C(x, y) + I_T(x, y) \tag{3}$$

4.1 Potential Generating Objects

Our potential generating objects provide the basis for the behavior of our tank artificial intelligence, and when combined into the cumulative potential functions as explained above, we observe strategic group behaviors.

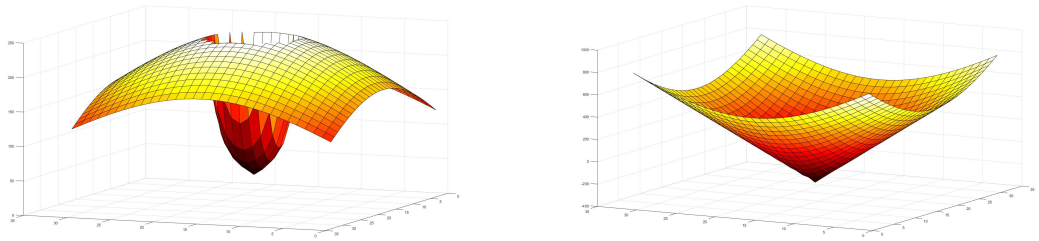
4.1.1 Obstacles

Our approach to obstacle potentials is straightforward. For each coordinate (x, y), add a potential of -1 to $M(x, y)$ for each surrounding obstacle, and if the coordinate is an obstacle, set $M(x, y) = -250$. For example, if a point (x_1, y_1) were surrounded by three obstacles, it would have potential $M(x_1, y_1) = -3$. In practice this proved to work well even though the change in surrounding potential is seemingly quite small; if there were no negative potentials around obstacles then it would be easy for the AI to get trapped in a concave enclosure, and if the negative potentials were too high then the AI would never go near obstacles, which would not be desirable because the player will likely be weaving in and out very close to obstacles. We found that the proposal of the authors in [1] was not as useful as our more simple approach; their approach is based on the distance d between (x, y) and a single obstacle, where $M(x, y) = \frac{-80}{d^2}$ if $d > 0$, otherwise -80. Indeed, the authors in [1] have to do a substantial amount of engineering (concave filling, path clearing, etc.) to get rid of the problems caused by their obstacle potentials.

4.1.2 Player

The player is the most important potential generating object in the game. The player's potential is what attracts all of the enemy tanks to approach and attack her. The player potential field $player(x, y)$ generated is conditioned on the euclidean distance d from the player to (x, y) (the coordinate for which we are calculating the potential), and the range of the AI tank's laser gun, R. Additionally, there are two different potential fields generated depending on which enemy tank T is requesting for the potential value; if T has more than half health, then it is receives the ATTACK field function value for (x, y), otherwise it obtains the RETREAT field function value.

$$player(x, y) = \begin{cases} \begin{cases} G(x, y, 250, 50) & d >= R \\ G(x, y, \frac{250}{R-d+1}, 50) & otherwise \end{cases} & \text{ATTACK} \\ 250(\frac{d}{R} - 1) & \text{RETREAT} \end{cases} \tag{4}$$



(a) Field emitted for requesting tank in ATTACK state. (b) Field emitted for requesting tank in RETREAT state.

Figure 2: Player potential fields according to different states of the tank requesting potential values; note that, in both images, the player is located at the center where the functions have the lowest values. Visualized on a 15 by 15 discretized grid.

In Equation 4 we are using the original Gaussian function (Equation 1) with an amplitude of 250 and standard deviation of 50. There are two cases for the ATTACK field: the top one (if $d >= R$) is the fundamental default attractive field that spreads across the map as a Gaussian. However, as we can observe in the visualization[1] in Figure 2, when $d < R$ the potential rapidly decreases due to the decrease in value of d, which modifies the amplitude according to the equation marked "*otherwise*". We design this ATTACK field to intentionally create the behavior of "setting up a perimeter" around the player for the AI tanks, as they are maximally attracted to the circle of radius R surrounding the player, and will not go closer because the potential rapidly drops off. We note that these functions for ATTACK are *not* differentiable when combined in this way, but future work may involve using the Laplacian of a Gaussian (i.e., the second derivative of a 2D Gaussian defined by polar coordinates) to model this desired phenomenon without relying on two functions.

For the RETREAT field we have a simple cone function that impels T to run as far away as possible from the player. This is desirable because T should not want to be destroyed, so this function is emitted to T until the shield is regenerated after five seconds without taking damage. We would have preferred to use a negative Gaussian for the RETREAT field[2], but we found in practice that strange behaviors were exhibited, where T would sometimes act as if impelled by a selfless delusion of grandeur and sacrifice itself, which, while admirable, is often not as strategic as retreating.

4.1.3 AI Tanks

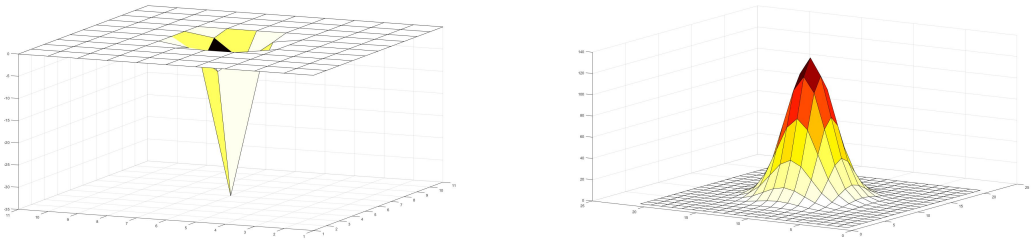
Our AI tanks emit minor potential fields corresponding to whether they are in the ATTACK or RETREAT state. We observe in Equation 5 (and in its visualization in Figure 3) that when the tank is in its ATTACK state it emits a negative Gaussian potential function around a small radius which corresponds to the standard deviation 0.75, with its overall strength corresponding to the amplitude 35. In practice, this functions to push tanks away from each other so that they don't all follow the same path, thus causing the tanks to establish a spatial perimeter around the player.

$$AItank(x,y) = \begin{cases} -G(x,y,35,0.75) & \text{ATTACK} \\ G(x,y,35,2.75) * Protection * \frac{50-h}{50} & \text{RETREAT} \end{cases} \tag{5}$$

In the RETREAT state we observe that the tank emits a positive potential field according to a *Protection* constant (7.5, in the implementation) and the tank's current shield strength h. The

[1]These visualizations (and all subsequent field visualizations) were generated using MATLAB.

[2]We thought it would be interesting (with no mathematical justification) to use the same distribution to model everything in the game, from the potentials to the city itself, but sometimes we needed to use non-Gaussian functions to get desirable behaviors.



(a) ATTACK field emitted.

(b) RETREAT field emitted with $h = 25$.

Figure 3: The potential fields emitted by an AI tank depending on what state it is in; note that the emitting tank is located at the center, the extremes of both functions. Unlike the fields emitted by the player, the requesting tank's state does not determine what field it receives, this is solely conditioned by the emitting tank. Visualized on a 10 by 10 discretized grid.

tank's max shield is 100, and the RETREAT state is activated only when the tank's health goes to less than half, so this field will always be positive. We want this field to be positive because we want to encourage the other AI tanks to move towards the tank in order to protect it from the player by blocking the player's path, or at least making it more difficult for the player to hunt down the damaged tank. Additionally, we have this field as a function of the tank's current shield strength (h) in order to allow for the tanks to implicitly prioritize protecting the more damaged tanks who need the assistance more. Lastly, we increase the standard deviation for this field because we want to have a larger radius of attraction for the other tanks to come and protect it, but we have a relatively small standard deviation for the ATTACK field because we want the tanks to be close enough to each other to make it difficult for the player to escape them after they set up a perimeter.

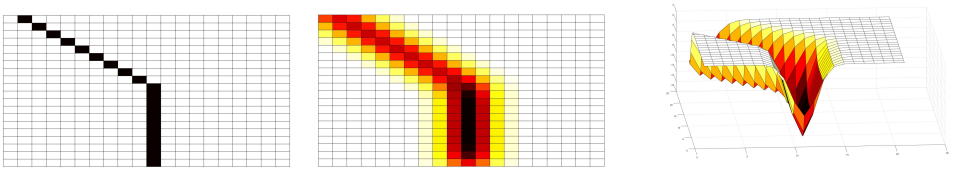
4.1.4 Repellent Potentials

In order to avoid the problem of getting trapped in local minima we have the tanks emit repellent potential generating objects for each coordinate they visit (described as a "pheromone trail" in a blog post[3] by the first author in [1]). These are *individual potential generating objects*, as described above in Section 4, meaning that they only effect the tank that emitted them.

$$repellent(x, y) = \begin{cases} -G(x, y, 5, 1.75) & t < 0.5 \\ 0 & \text{otherwise} \end{cases} \tag{6}$$

In Equation 6 we observe that the repellent potentials are modelled by a negative Gaussian function conditioned the amount of time the repellent object has existed, t seconds. This condition that $t < 0.5$ thus means that the repellent object only emits a potential for half a second until it zeros out. We put this timer on the repellent potentials because the player constantly is changing position, so it is sometimes necessary for the tank to quickly backtrack to previous positions. Additionally, the repellent potentials will still fulfill their intended purpose because the longer a tank ventures around a certain area, the more the negative potential of each of its repellents will accumulate, and thus the more impelled will the tank be to leave this area. In the implementation, we do not have the tank emit repellent potentials if it is shooting at the player because if we did then the tanks would never stay still. In Figure 4 we observe an example of the summed repellent potentials along an arbitrary path travelled by an AI tank.

[3]See `http://aigamedev.com/open/tutorials/potential-fields/`.



(a) Path travelled. (b) Repellent field emitted. (c) 3D visualization of repellent field.

Figure 4: The individual potential field determined by the sum of the repellent potentials emitted by objects created by an AI tank at each point travelled upon in the path. Visualized on a 10 by 10 discretized grid.

4.2 Running Time

Running time is an essential aspect of any algorithm, especially for video games where slow running times make the game unplayable. To properly analyze the running time of our game, we must consider the n potential generating objects for an individual tank and d, the length of the shortest path to our destination. A standard breadth-first-search algorithm would cost $O(8^d)$, where 8 is our branching factor, the number of neighbors around a single point. This is very expensive, but the A* algorithm can improve this significantly, although even with a strong effective branching factor from a good heuristic (some $b < 8$) we still have exponential running time, $O(b^d)$.

Potential fields, however, only need to look ahead one step at a time. This means only the 8 neighbors have to be checked, so, on the surface, we seem to have a constant time algorithm, $O(8) = O(1)$. However, because each of our n potential generating functions $f(x, y)$ must be summed together for computing the potential of the neighbors' coordinates (x, y), we actually have an $O(n)$ path-finding algorithm for a single tank, which is linear in the number of potential generating objects. This is still much faster than the basic exponential time algorithms described above, but we still have to deal with the limitations of potential fields. Additionally, we find that even with the complex-to-compute Gaussian functions and the square roots in euclidean distances we experience no observable loss in performance.

We find that the group pathfinding is an implicit feature with potential fields, where more tanks just linearly increases the number of our $O(n)$ computations, so if there are m tanks, we perform $O(mn)$ computations. Our rapid group-pathfinding stands in contrast to the significant amount of complex engineering that has to pursued in order to develop group path-finding algorithms with the more standard methods, and the desired group behaviors (such as surrounding the player in a perimeter) would have to be explicitly encoded, whereas in potential fields we can do this entirely implicitly just based on their simple potential functions.

4.3 Perspectives for Extension to Continuous Representation

Although time did not permit an exploration into the practical details of a continuous implementation, here we propose a theoretical extension of discrete potential fields to a continuous setting. In this continuous representation, by relying on principles of vector calculus, we no longer need to even iterate over neighbor coordinates, instead we would compute the total gradient of all of our *potential generating functions* and move in the direction of that gradient each frame.

In general, we treat the entire set of potential generating functions as a summed composition of functions which we are attempting to ascend. The principle of gradient ascent is directly applicable here since the gradient of a function[4] points in the direction of greatest rate of change of the function, thus meaning that we will move in the desired direction. Additionally, the magnitude of the gradient,

[4]The vector of partial derivatives of a function; i.e., $\nabla f(x, y) = \langle f_x, f_y \rangle$.

$||\nabla f(x, y)||$, represents the intensity of the slope.

Formally, at each frame, for each of our n potential generating objects i with potential generating function $f_i(x, y)$, compute the values of the gradient vector $\nabla f_i(x, y)$ at that point (x, y), sum all of the gradients together according to a pre-determined (or learned with some variant of reinforcement learning) weighting scheme w_i, and move in that direction with a velocity equal to the magnitude of the resulting vector. In Equation 7 below we observe the mathematical formalization for computing our *heading vector* $\vec{h}$:

$$\vec{h} = \sum_{i=1}^{n} w_i \nabla f_i(x, y) \tag{7}$$

This continuous extension would have several benefits. The most significant is that the agents would now move around much more smoothly, as opposed to the rigid, unnatural movements that result from the discretized implementation. Additionally, this continuous implementation would remove the necessity of two architectures in the implementation, whereas in the discretized version there has to be translation back and forth between the internal discretized representation and the virtual continuous world.

However, new considerations and problems would have to be addressed. The question of how to compute the gradients of each function arises, whether they should be approximated or analytically engineered for each function. The problem of the semi-static field representation also arises, because now each obstacle will require a repellent differentiable function associated with it, rather than the 2D array of constants (which would differentiate to zero). One option would be to simply have, for each obstacle, a gradient vector associated with it that points directly orthogonal from it, whose weight we would tune accordingly in the testing, parameter-tuning stage of development.

5 Experiments

The majority of experimentation for this project involved tuning the many hyper-parameters of our potential fields, particularly the amplitude and standard deviations of each of our Gaussian functions. We found it necessary to significantly decrease the standard deviation of the repellent fields from what is presented in Section 4.1.4 down to a small value of 0.15. This is because having the standard deviation higher would cause the tanks to move around confusedly without heading toward their destination, especially if they would have to go through obstacle-filled environments. All of the other values presented in Section 4 are the same in the final implementation.

We also performed a set of stress tests, observing high performance (over 60 frames per second) until we reach about 130 AI tanks, whereupon performance tends to degrade as more tanks are added. This is very desirable performance since this game was made for a player to battle against a set of tanks, so we would likely never have more than 25 tanks, which would already be very difficult for the player. The code could also be more optimized and thus accommodate significantly more tanks, where one future perspective would involve having AI tank wars between large teams of tanks.

In Figure 5 below we observe behaviors exhibited by our AI tanks, where the red lines are the lasers they are shooting at the player. In sub-figure 5(a) we can clearly observe the desired behavior where the tanks strategically surround and set up a perimeter around the player. This occurs because of the player potential described in Section 4.1.2, and because the AI tanks emit the negative potentials described in 4.1.3, which repel each other enough to spread them apart along the circle, but not too much as to cause them to out of range of the player. We observe this behavior develop quickly in a period of a few seconds when the player is near all of the tanks. In sub-figure 5(b) we observe that the tanks still encircle the player in a relatively dense obstacle environment, although we note that the one tank in the far left corner had a bit of trouble reaching the player.

However, potential fields are not perfect. We found that the protection field emitted when an AI tank retreats only occasionally attracts tanks to defend their wounded comrade; future work would involve improving the likelihood of this desired behavior through more parameter tuning. In sub-figure 5(c) we observe that the tanks had difficulty following the player into the small crevice in this dense obstacle environment. Even after one minute of the player not moving, we observe in sub-figure 5(d) that they could not adequately reorient themselves to have more than two of them attacking the player. This is due to the negative potentials they emit to each other, their negative individual repellent potentials, and the negative obstacle potentials, which all accumulate to make a very "hilly" potential field with many local extrema. This may not be an undesirable behavior, however. The game is quite fast-paced and intense, so the player might actually drive into a crevice for the purpose of avoiding the tanks, or for the purpose of being able to fight them individually since they are so strong collectively. Additionally, this seemingly undesirable behavior can have the interpretation that the tanks are setting up a perimeter around the player's hideout, which is true, because any direction the player goes to leave will be greeted by at least one tank.

6 Conclusion and Future Work

We have presented an approach to group pathfinding in an obstacle-filled environment using artificial potential fields. We have shown that, with a certain amount of parameter tuning, one can create a very efficient potential field system that allows for hundreds of AI agents to perform group pathfinding and exhibit strategic group behaviors entirely implicitly as a result of the potential field functions generated by the different objects and agents in the world. We believe that the benefits of potential fields (i.e., rapid dynamic group-pathfinding with strategic behavior) strongly outweigh their disadvantages (i.e., the tendency to get stuck in local extrema), and we provide solutions that help to minimize the impact of these disadvantages.

In future work we would like to extend these principles of potential fields to a continuous representation, as described in Section 4.3. Additionally, we would like to experiment with large AI tank wars between hundreds of tanks which all are directed by these potential fields, in which additional complexities are added to the environment, such as an incentive to destroy buildings (i.e., to spawn power-ups), fog-of-war, and an implementation of a more city-like structure with roads and alleys.

References

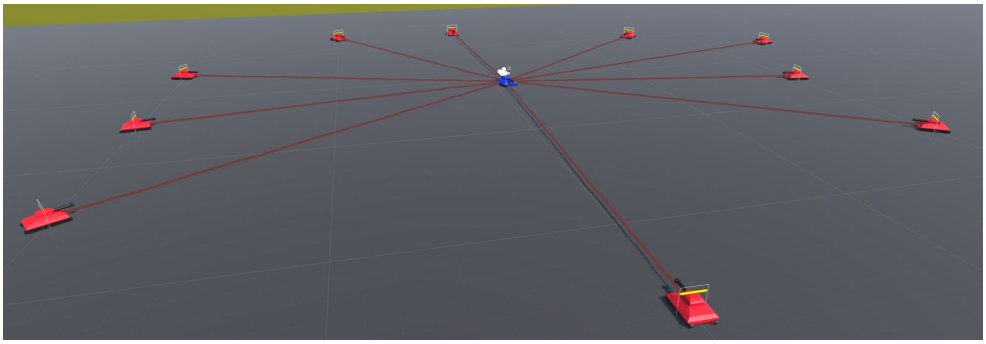
[1] Johan Hagelbäck and Stefan J Johansson. A multiagent potential field-based bot for real-time strategy games. *International Journal of Computer Games Technology*, 2009:4, 2009.

[2] Oussama Khatib. Real-time obstacle avoidance for manipulators and mobile robots. *The international journal of robotics research*, 5(1):90–98, 1986.

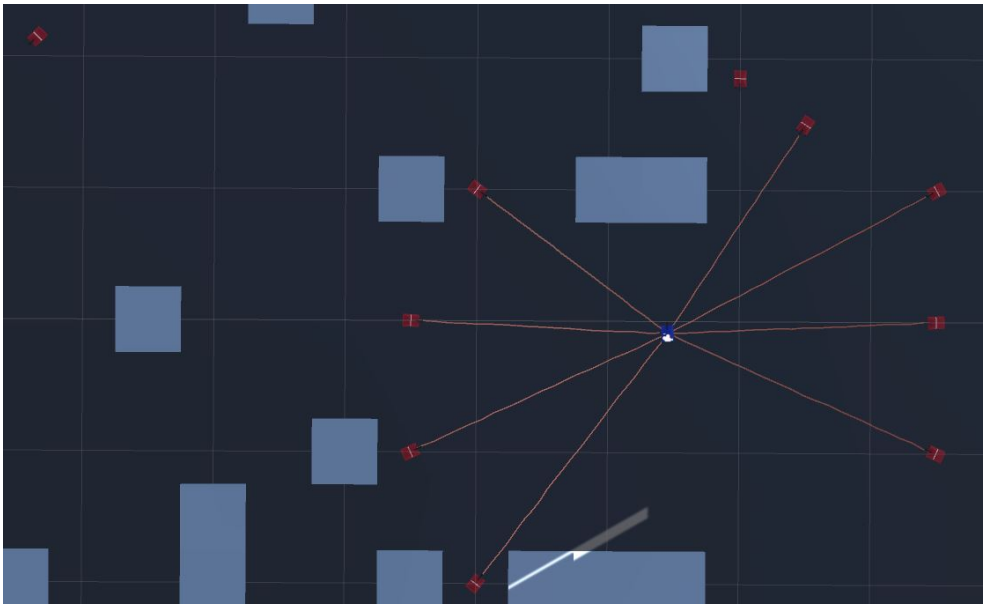
[3] Yoram Koren and Johann Borenstein. Potential field methods and their inherent limitations for mobile robot navigation. In *Robotics and Automation, 1991. Proceedings., 1991 IEEE International Conference on*, pages 1398–1404. IEEE, 1991.

[4] Johan Hagelbäck and Stefan J Johansson. Using Multi-agent Potential Fields in Real-time Strategy Games. In *Proceedings of the 7th International Joint Conference on Autonomous Agents and Multiagent Systems, Volume 2*, pages 631–638. International Foundation for Autonomous Agents and Multiagent Systems, 2008.

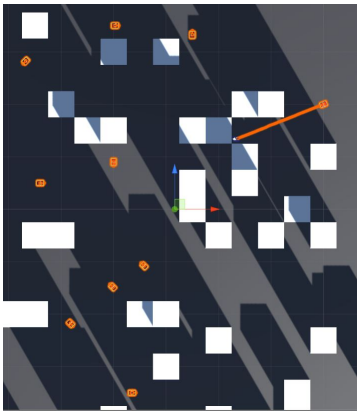
[5] Johan Hagelbäck. Potential-field based navigation in starcraft. In *Computational Intelligence and Games (CIG), 2012 IEEE Conference on*, pages 388–393. IEEE, 2012.



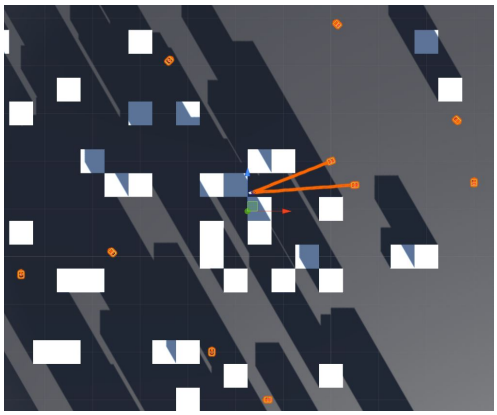
(a) Tanks encircling the player in a perimeter.



(b) Tanks encircling the player in a perimeter around obstacles.



(c) Confused tanks.



(d) Confused tanks after one minute.

Figure 5: Visualizations of different tank behaviors and formations.