Optimal Cover Placement Against Static Enemy Positions

Yinxuan Shi and Roger Crawfis
Computer Science and Engineering, The Ohio State University, Columbus,OH, United States
shiyi, crawfis@cse.ohio-state.edu

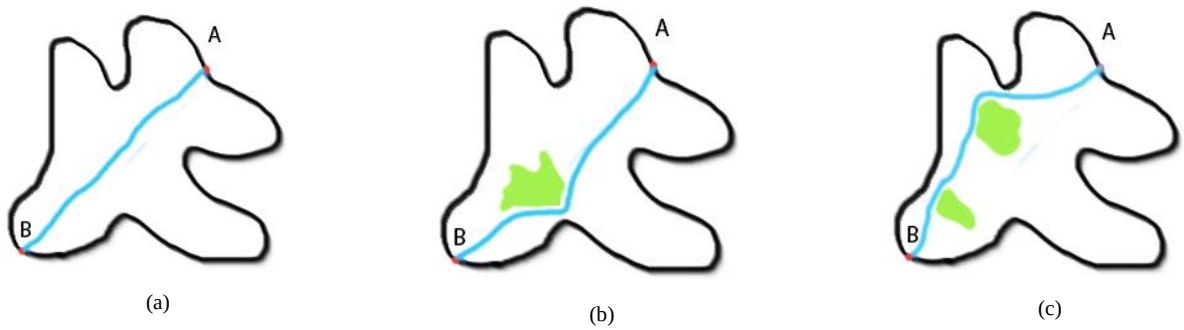


Figure 1: An arbitrary region with different cover constraints (a) Possible optimal path from A to B when there is no cover, (b and c) optimal paths with different amounts and distributions of cover.

ABSTRACT
With the popularity of first-person shooter (FPS) games and role-playing games (RPG), procedurally generated levels are a growing topic of interest in games research. Level design for FPS games aimed at presenting interesting gameplay, involves map generation as well as object and resource placement. Existing techniques mainly focus on map generation. The placement of objects or debris, in which a player can take momentary cover, as well as the locations of hardened enemy positions, greatly impacts the gameplay and the strategy a player may take towards progressing through the game. We propose the damage function to encode the flux of damage at every point in space throughout the level. We work under the premise that there exists a path that is optimal in some sense through this damage field (i.e., there exists a path that would inflict the least amount of damage on the player). We describe how to create a damage function under several use cases and compute this optimal path. Every configuration of enemy locations and cover elements potentially creates a different damage function and hence a different optimal path. We examine various metrics that may be used to compare one configuration to another. Using this framework we are able to search for optimal covers under various metrics.

Categories and Subject Descriptors
K.8.0 [**Personal Computing**]: Games

Keywords
Game design, level design, procedural content, player experience, first person shooter game (FPS).

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, or republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
Submitted to FDG'13, Month 1–2, 2010, Crete, Greece.
Copyright 2013 ACM 1-58113-000-0/00/0010 …$15.00.

1. INTRODUCTION
An FPS game typically contains intense combat where a player engages hostile target using melee or ranged weapon through gameplay. Most commercially released FPS games are separated into multiple areas or maps, called levels. Level design can make the difference between a game that is repetitively boring or immersive fun.

Level design not only includes the map or navigable area generation, but object or resource placement as well. Unlike multi-player FPS games (e.g. *Halo 4* [1] and *Unreal Tournament 3* [2]), single player FPS games (e.g. *Deus Ex* [3], *Dishonored* [4]) usually have more emphasis on taking cover. Since taking cover is one of the basic game mechanics [5] in a single player FPS game, cover placement has a significant impact on the level design.

Consider the scenarios depicted in Figure 1, where the player starts at position A and has a goal to make it to position B alive. In the absence of cover or no cover near the player, the player will likely take a straight line to his/her destination. This will result in an intense game experience where the player has to run and shoot. However, in Figures 1(b) and (c), when provided with cover, he/she may take a safer or stealthy route to the destination, which requires some strategy.

When taking a closer look at Figure 1(c), the player's path may consist of two elements: hiding in a cover location or running to the next best cover location/destination. When the player is moving from one cover location to another cover location, he/she may encounter the Figure 1 situation. With a good cover location placement, the player can experience different styles of game play, which adds variety to the game.

Pacing is a concept that describes the player's perception of a game level [6]. A well-paced level, also known as flow, provides

moments of action/peaks interjected with periods of calm/troughs. Most commercial FPS games, like *Call of Duty: Black Ops II* [7], etc. involve varying pace within the same level. Being in a trough for too long will lead to tedious gameplay and lack of excitement, while remaining in peaks for too long will result in desensitizing a player and repetitive or boring action. Many times, before a "boss fight", the player will have the opportunity to rest and/or acquire some power-ups. Game designers may specifically design a level to create a flow of the game, where the player feels enjoyment and control in an autotelic activity.

Numerous techniques have been proposed for level design of FPS games [8], many of which gear towards procedural map generation. There is relatively little study on cover placement. It mainly relies on manually tweaking by the designer to decide the cover placement, as it can be crucial to the flow of a game.

In actual level design, some consideration is given to the architecture feasibility and aesthetics. E.g. for some game level inside a building that is similar to Parthenon Temple, where several pillars are used as cover, the designer has to keep these pillars symmetric rather than place them based on gameplay. Even for an outdoor level, it is a difficult task for game designers to come up with a configuration of cover that optimizes gameplay due to limited time and play testing devoted to the large space of possible configurations.

With this in mind, our research goal is to develop a method to generate optimal cover for FPS style games. In order to achieve this, we examine several problems. First, we examine the problem of determining optimal paths given a cover and an enemy (NPC) distribution with behaviors. Secondly, we examine how the placement of cover items affects the optimal path and develop a framework for searching for an optimal cover. Finally, we examine how designers can best use this framework by specifying a desired flow that the cover should optimize towards. There is a duality between optimizing the cover for a fixed NPC distribution and optimizing the NPC distribution for a fixed cover. This paper will focus on the former, but can easily support the latter.

The remainder of the paper is organized as follows: Section 2 outlines related work. In section 3, we develop a theoretical framework for paths, cover distributions and enemy distributions. In section 4, implementation details of our approach to creating possible candidate covers and selecting the optimal cover is discussed. Then in section 5, an analysis of our approach is provided along with examples and use cases that show the advantages and flexibility of our approach.

2. Related Work

Procedural content generation for game levels lowers the cost for building FPS games and provides nearly infinite replay value for the player. Hence, procedurally generated FPS game levels have been a growing topic in game research.

2.1 Level Design

Several level generation techniques have been developed. Christian and Troels [9] introduced the spatial principles of multiplayer FPS level design associated with a *Collision Point* where teams confront each other and a *Tactical Choice* by which the team and player may seek to perform the collision in the best possible way. They provide the idea of using rectangular areas as cover location in tactical planning. However, lacking actual formulation and solution to the problem, it can not be applied in procedurally generated FPS level.

Game design researchers have presented a taxonomy of design patterns for an FPS game. Kenneth and Jim [10] provided a set of formal design pattern for FPS games. Yet, they lack clear usages. Luigi et.al [8] proposed a method of evolving maps for FPS games. The novel feature of their work is combining a search-based solution to evolve maps based on a player's average fighting time.

2.2 Game Experience

Game experience, also known as player experience, is considered as the Holy Grail of game design. Terms that describe game experience include: immersion, presence and flow. Introduced by Csikszentmihalyi, the flow state is known as an optimal state of intrinsic motivation, where the person is fully immersed in what he is doing [11]. Flow state is what is desired by the game designer, yet it is not clear how to use it to aid in level design for an FPS game.

Lennart and Craig proposed a method of measuring player experience through electroencephalography, electrocardiography, electromyography, galvanic skin response and eye tracking equipment [12]. However, due to the cost of game testing and the equipment, it is difficult to use this method to help game design.

Instead of measuring the actual player's physiological traits when he or she is playing the game, Penelope and Peta [13] developed a model for evaluating player enjoyment in the game by game flow. Yet since the player experience is measured after the production of game, it is not as helpful in game design.

Based on this idea, Nathan and Philippe [14] applied a fitness function to quantify fun in a game. For the use case they provided, it is not clear if the usage of their fitness function can be applied to an FPS game. In order to solve this, we simulate a player's behavior as a fitness function to evaluate the game level, and help in the level design.

3. Method

In order to simulate a player's behavior, we need to examine the player's probable path. Intuitively, the player will likely take the path of least resistance. In order to find the path with least resistance in a given area, resistance needs to be formulated.

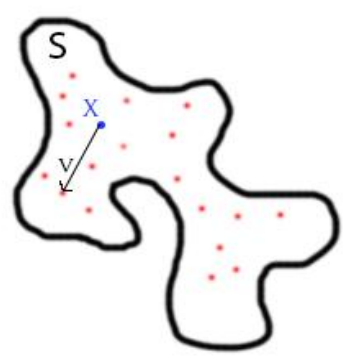


Figure 2: Theoretical set-up. *S* represents a closed game area. The blue point, x, represents the virtual player, red points represent enemies. The vector $\vec{v}$ illustrates the direction from the player to one enemy.

3.1 Damage Function

Consider a player located at point $\vec{x}$ and some distribution of enemies inside a closed area *S*, as depicted in Figure 2. In the absence of additional targets, we can assume that at time *t*, the

player will receive damage (probabilistically) from one or more enemies located in the direction $\vec{v}$ relative to the player. This can be denoted as $damage(\vec{x}, \vec{v}, t)$. In many scenarios, shooting accuracy is affected by the distance between the shooter and the target. This is a function of $|\vec{v}|$, and represents the likelihood of a player getting shot from the position $\vec{x} + \vec{v}$. Using this, total damage received from all directions at time t and location $\vec{x}$, is given by:

$$total(\vec{x}, t) = \int_S damage(\vec{x}, \vec{v}, t) dv \quad (1)$$

When facing a large number of enemies, the player may endure more damage compared to facing the enemies one by one. This may represent game mechanics when a player is stunned slightly after being hit and has even more difficulty fighting back or seeking cover. The damage function may be biased so that the player is penalized when going into enemy condensed area.

Given a path denoted as $path = \overrightarrow{x(s)}$, we model a user's behavior by parameterizing the path by time, t. This gives $s = s(t)$. So the path can be rewritten as: $path = \overrightarrow{x(s(t))} = \overrightarrow{x(t)}$. The damage along the path, or in taking the path, is the integral over the function $total(\vec{x}, t)$, with respect to time. This is denoted as:

$$Damage_{path}(t_1, t_2) = \int_{t_1}^{t_2} total(\overrightarrow{x(t)}, t) dt \quad (2)$$

3.2 Optimal path for a fixed origin with cover placement c

We assume that the ***optimal*** path is the path with the least resistance. Hence, we seek the path with the minimum amount of damage from all possible paths connecting A to B, over all time periods:

$$OptimalPath = min_{\forall paths}\{Damage_{path}(t_1, t_2)\} \quad (3)$$

This gives the optimal path given a specific damage function. The damage function is related to the cover in **S**. Let us now look at how to find the optimal cover within **S**.

3.3 Improve the cover

Before we look at the general case, let's assume we are given some set of covers **C** for a fixed set of enemy positions. This changes the damage function while retaining the enemy positions. For each cover $c \in$ **C**, there is an optimal path corresponding to it, denoted $OptimalPath_c$. We can define the optimal cover as the cover which has the least amount damage along its optimal path (see section 3.4 for other metrics).

$$OptimalCover_C = min_{\forall c}\{ OptimalPath_c\} \; \forall c \in \{C\} \quad (4)$$

This biases the cover towards making the level easier (or allowing more complex enemy distributions). Of course, if the covers in the set C have different *amounts* of cover, then there will be a bias towards the covers with more cover area.

3.4 Feasible Covers

The amount of coverage is typically dictated by the designers. A designer may also want to place cover at certain locations. E.g. some area in the map may be pre-determined as an inaccessible area by the game. If the designer limits the total coverage amount, denoted *max_cover_amount*, we constrain the set of feasible covers to those whose cover amount satisfies the following:

$$\int_S Cover(\omega) d\omega \equiv \text{max_cover_amount} \quad (5)$$

In general, there may be many constraints on the cover. For instance, the minimum unit of cover may be a two foot by six inch wall that is three feet high.

3.5 Optimize cover for other metrics

Rather than picking the optimal cover according to the minimum optimal path damage, we can adopt other metrics. For instance, if we desire longer gameplay through this area, we might pick the cover with the longest path length of the optimal path. Likewise, the cover whose optimal path has the largest standard deviation may lead to a more varied gameplay. In general, we can choose a metric where the optimal path most closely matches a curved specified by the level designer. This would allow a designer to specify the flow, and have the algorithm search for the best cover to match this.

In general, we define a fitness function, $DesiredProfile(OptimalPath_c)$:

$$OptimalCover_C = min_{\forall c} \left\{ DesiredProfile_{min_{\forall path}} \left(\left\{ \int_t \int_R damage(\vec{x}, \vec{v}, t) dv \, dt \right\} \right) \right\} \quad \forall c \in \{C\} \quad (6)$$

Next, we will discuss one approach to determining possible covers, determining the damage function and finding the optimal cover for a particular configuration.

4. Implementation

To simplify the problem, in our implementation we assume that there exists a finite set of covers. If this set can be enumerated, then a simple brute force method is can be applied to compute the set of candidate covers. There are two major parts of our algorithm: generating a feasible candidate cover set and selecting which cover is optimal according to the desired metric.

4.1 Find an optimal cover placement

SelectOptimalCover takes two inputs: C as a feasible cover set and S as a map for the area. First, we obtain a cover from the candidate cover set, and then compute the optimal path based on this cover. *ComputeOptimalPath* takes the current cover location and returns the optimal path for this configuration based on Equation (3).Then the *DesiredProfile* function (Equation 6) will be applied to the optimal path to obtain a score for this cover configuration. As mentioned above, *DesiredProfile* may specified by designer. The cover with the highest score is thus the optimal cover.

```
SelectOptimalCover(C, S)
for each cover ∈ C
        do path = ComputeOptimalPath(cover, S)
           value = DesiredProfile(path)
           if value < bestValue
              then bestValue = value
                   bestCover = cover
return bestCover
```

Note, for time varying distributions of enemies or damage, the problem gets overly complex. Given constraints on the player's movement speed and the constraint that the enemy is fixed, we

can ignore time and use Dijkstra's algorithm to compute the optimal path. We take each discretized grid cell in S as a graph node and each node only connected to neighboring node. For a square grid, the common ones are 4 neighbors and 8 neighbors. In 3x3 grids, for 4 neighbors, center node is connected to the up, left, right and bottom node while for 8 neighbors, center node is connected to every surrounding node. The edge weight is computed as the average damage of the current node and neighboring nodes. Therefore, we can compute the optimal path from a pair of given start and end locations within reasonable time. Unfortunately, the A* search algorithm can not be used here since the traditional heuristic function using Manhattan or Euclidean distance is not admissible.

4.2 Create candidate set

In creating candidate covers, we take two inputs: one is the number of covers to be generated; another is the cover constraint T, which can be a number or a function. First we initialize the set C then, for each iteration, one cover will be generated by *GenerateCover* function. The generated cover is then tested against our *SatisfyConstraint* function to determine whether it satisfies all of the user's constraints (Equation 5). If the generated cover satisfies the cover constraint, it is added to the candidate cover set.

CreateCandidateCover(n, T)

$C \leftarrow \emptyset$

$i \leftarrow 0$

while $i < n$

  do $cover \leftarrow GenerateCover()$

    if *SatisfyConstraint(cover ,T) = false*

      then continue

      else *UNION(C, cover)*

        $i \leftarrow i + 1$

return *C*

5. Experimental Result

We implemented our algorithm to generate several cover configurations for the same cover constraints. In our simulation, we assume that S and the cover are circular areas and enemies are distributed in S. In order to speed up the simulation, we assume that the enemy flux, which is the damage from an enemy, doesn't change over time. In order to encourage players to take cover, high flux areas where most enemy can attack are penalized. Also the path weight is counted to exclude the situation of unnecessary wandering in low damage areas. In addition, a kernel function acted as shooting accuracy is computed as a decay factor for damage, since the larger the distance between the shooter and the target, the less chance that the shooter will hit the target.

We use the *Damage Map* to indicate the damage in the area, where a discretized grid represents the amount of damage received if the player is on that grid cell. Two use cases are discussed, one with a uniform enemy distribution along the perimeter of S (Figure 3), another with a discrete enemy distribution inside of S (Figure 5).

5.1 Uniform enemy distribution

Our first use case uses an analytical computation of the damage function. We consider an infinite number of enemies that is uniformly distributed along the perimeter. There are *n* covers, each of which is represented as a circular area with radius of *r*. Covers are constrained to be contained in S having a radius of *R*, The covers are non-overlapping.

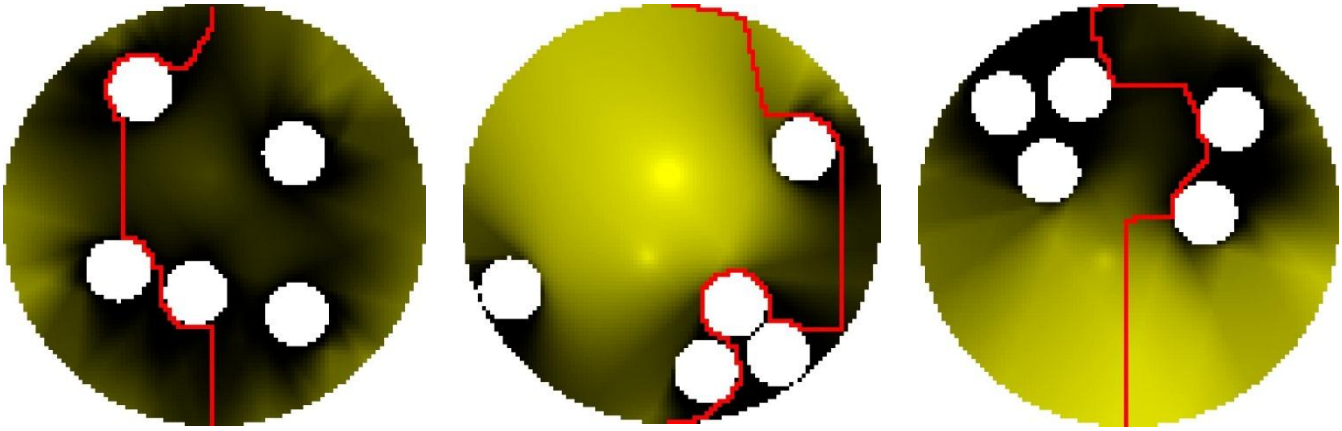
Since the enemy is uniformly distributed along the perimeter of S, the damage for each grid cell can be represented as a percentage of the perimeter. Enemies can only attack a player when they are able to see the player, so whether the ray between the player and the enemy is occluded by a cover determines whether the player is attacked by the enemy. Therefore the damage of a grid cell can be computed as the percentage of the perimeter that is not occluded by cover.

To encourage the cover to be used, we biased every damage map so that if more than 70% of the map was not occluded by cover, the damage would increase. Also a small amount of path length penalty was added when searching for the optimal path for each cover. The entry location is at the top of the area, while the exit location at the bottom of the area. As depicted in Figure 3, three different cover placements are selected by different metrics.

From 1000 generated covers, Figure 3(a) shows the covers where the optimal path has the least amount of damage compared to the rest of the covers. The large circle is the area S, white interior part represents the cover. The optimal path is illustrated with a red curve. The darker the region gets the less damage it receives when the player is at that point. In the center of area S in Figure 3(a), it appears darker compared to Figure 3(b) and 3(c), since a larger percentage of the perimeter is occluded, which falls below our penalized threshold of 70%. On contrary, in 3(b) and 3(c), the center part in S is much brighter due to large open area where nothing occludes the enemies.

The longest path length cover is displayed in Figure 3(b). This cover placement corresponds to the longest optimal path in the candidate cover set. Another metric might be the optimal path with the largest standard deviation of damage across the path (Figure 3(c)). If we consider high damage areas as 'intense' and low damage areas as 'relaxing', this path may provide a more interesting flow to game experience. The player may experience a relaxing and stealthy gameplay in the beginning. After traveling to the center of S, he or she may have to fight their way to the exit, which involves a lot more combats. Combating will provide an intense gameplay, which is contrasts previous relaxing gameplay. Therefore, this flow contains a surprising element of gameplay.

Note that all of the paths in Figures 3(a) - 3(c), exhibit a more circuitous route than a nearly straight path connecting the entry to the exit. This is dictated by the damage function rather than by a simple distance function. In Figure 3(b), the path goes around the one cover rather than in between or below the cover because the path between the two circles is not wide enough for a player to go through.



(a) Minimum Damage Cover **(b)** Longest Path **(c)** Largest Standard Deviation

Figure 3: Results generated for a fixed cover amount within a circle and equal damage emanating from the boundary of a circle. Cover is indicated by the white interior. Damage function has high values mapped to yellow and low values mapped to black. (a)(b)(c) The cover map with computed optimal path. (a) Cover that results in an optimal path with the minimum amount of damage (the easiest configuration). (b) Cover that forces the player to explore more of the area. Note that some of the cover is not used. (c) Cover whose optimal path has regions of heavy damage and little damage. This may provide a nice flow or allow the player to rest and strategize before making it to the exit.

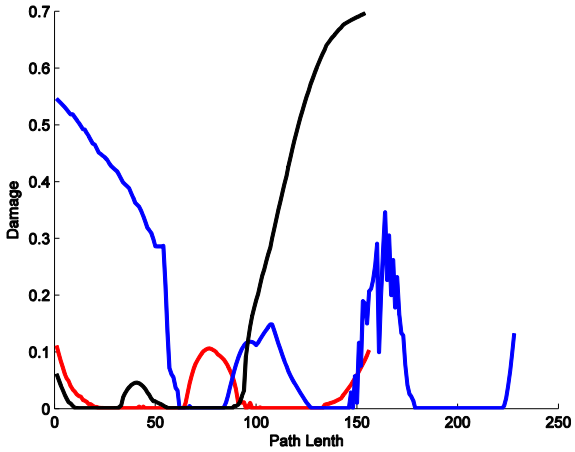


Figure 4: The damage/length graph by 3 selected path in Figure 3. Red line is the minimum damage path, blue line is the longest path, black line is the largest standard deviation path**.**

Figure 4 compares these three different metrics for optimal cover: the minimum path damage (red line), the longest path length (blue line) and the largest standard deviation of the path damage (black line).The amount of damage received along the path is shown, with the *x*-axis as the path length and the *y*-axis as the damage. On the metric for largest standard deviation, it starts with low damage since the beginning of the path is on the top of area. After going back and forth between covers, the path leads to a wide open area. This results in high damage, which is reflected as high values on the curve in Figure 4. This curve has a flow of restful to sudden danger. The longest path is backwards; it starts out intense then gets restful.

For Figure3 (a), this type of cover can be used as a relatively easy level for a player to pass or maybe useful for a more stealthy gameplay element. In Figure 3(b), more gameplay can be provided in a fixed area of S. In Figure 3(c), the cover placement may bring player a surprising game experience. Damage/path profile can also be utilized as selecting optimal cover as in Equation 6. Just like Figure 4, instead of showing the curve for a designer to use, a most similar curve will be chosen from the candidate set based on the designer's input curve.

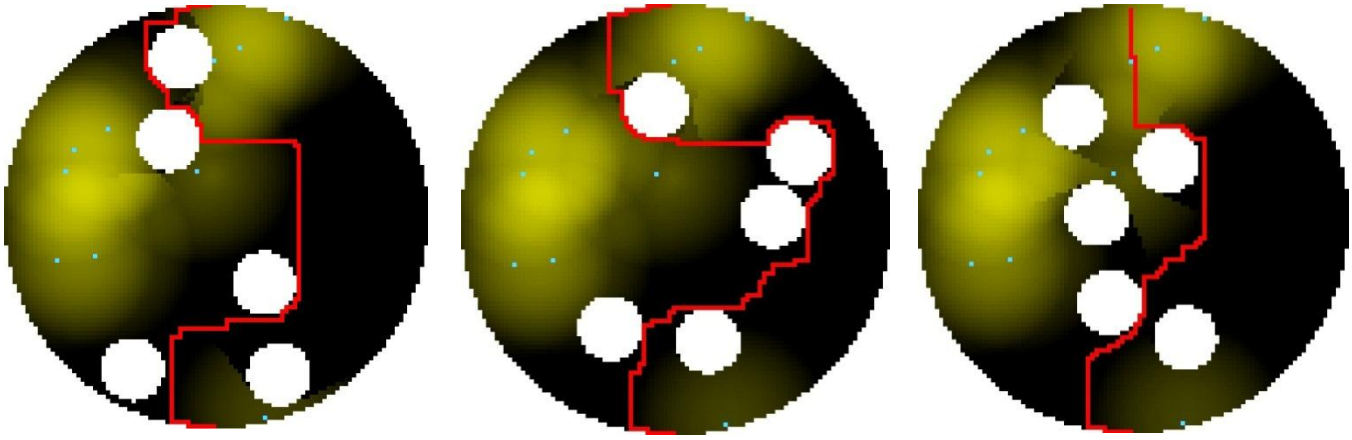
5.2 Discrete enemy distribution

Our second use case has a discrete enemy distribution inside the area. Again we use a circle for the regions but any shape can be used. In this case, ten non-overlapping enemies are randomly placed inside the area. We constrain the cover so that in total there are five covers each, and no overlaps with each other or enemies. The entry and exit points are placed as the first use case, at the top and the bottom respectively. As shown in Figure 5, the optimal paths connecting the entry and exit points are marked as red lines.

Cover with the minimum path damage is displayed in Figure 5(a). Again the darker area in the image receives less damage. The brightest part is an enemy location exists (Blue point). The damage from an enemy fades to zero based on distance, due to a shooting accuracy kernel. In an actual game, it is usually a lot easier to shoot the enemy if they are right in front compared to they are a block away.

The longest path length cover is displayed in Figure 5(b) while largest standard deviation of damage across the path is depicted in (Figure 5(c)). Note that the enemy distribution is predetermined, so in this scenario the designer can mark some area as a heavily guarded area (high enemy population) while some areas as patrol areas (low enemy population)

Figure 6 shows damage depending on three different paths from Figure 5(a) - 5(c). Similar to Figure 4, the red line represents the cover with the minimum damage, the blue line stands for the longest path, while the black line represents the largest standard deviation. As shown in Figure 5, an enemy located at the bottom of the map contributes to the peak in the end of the three paths.



(a) Minimum Damage **(b)** Longest Path **(c)**Largest Standard Deviation

Figure 5: Result generated from a fixed distribution of enemy inside S. Blue point stands for enemy. Same as Figure 3, Cover is indicated by the white interior. Damage function has high values mapped to yellow and low value mapped to black.(a)(b)(c) is the damage map with optimal path.

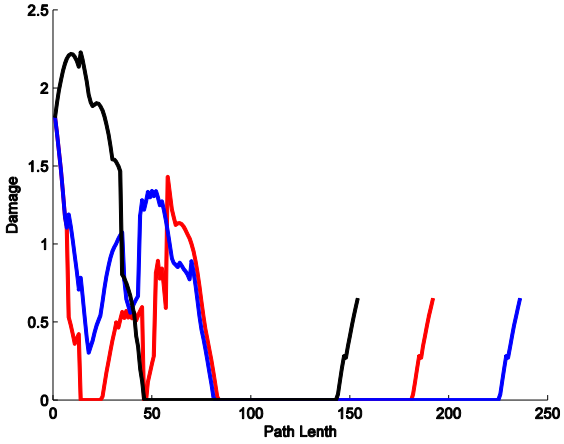


Figure 6: Damage/Length Path by 3 chosen paths in Figure 5.

Figure 8 illustrates several different cover placements and their damage profile generated with a fixed discrete enemy distribution and a constant cover amount. For each image, the top part shows the damage map while the bottom part is a damage bar which is color coded by damage along the optimal path. We can use this visualization to quickly scan the possible cover configurations. For example, on row 4 column 4, the cover will provide an interesting flow to the game since its damage bar follows the pattern of interleaving *peaks and toughs.* In Figure 7, we show a sample level of an FPS game with this cover configuration generated in Unity. For some enemy distribution, it may seem that cover placement with different cover amounts will provide a similar game experience. However, this requires a more complex model to optimize the cover amount for a fixed enemy distribution, which we plan to explore in the future work.

Our result is implemented with C++ and visualized using OpenGL. The experiment was run on an Intel Core 2 Duo chip set. Five hundred cover sets can be computed within 2 minutes, around 4 cover sets per second, which is tolerable for interactive level design.

6. CONCLUSION

A framework for optimal cover placement under fixed enemy positions has been presented for FPS style games. Our contributions include (1) a definition of the damage flux, its calculation and considerations for contrast enhancing it to favor covered or partially covered locations; (2) Optimal path determination based upon the damage function; (3) An iterative approach to generating and searching for optimal cover; and (4) Several metrics that can be used to steer the optimal search. The method can be applied in any FPS or RPG game to obtain various styles of gameplay as dictated by the designer. We presented two simulations to aid in developing and proving the framework. Both work with a discretization of the damage function. One focuses on enemy fire from outside of a local region, while the other considers enemy locations within a region. The former can be viewed as indicating how exposed a player is along a path and may work well for any enemy placement, dynamic enemy placement and/or multi-player situations. The latter simulation allows the cover to be tweaked in the case of hardened or static enemy placement. The metrics allow for longer game play within an area (longest optimal path), better or stronger enemies (minimal damage), or shaping of the flow of the level (optimal path whose damage profile most closely matches a desired profile curve). The simulation is fairly fast, achieving an analysis of over four cover configurations per second.

7. FUTURE WORK

In this paper, we constrain the enemies to static locations. For future work, we would like to model time-varying enemy distributions or time-varying damage flux. This allows for the enemies to move as well as simulates the player eliminating hostile threats as he or she progresses along the path. This will require a complex heuristic for path finding. The addition of other game objects (e.g. ammo and health potions) can be added into our simulation with the addition of rewards as well as damage. The visualization of the cover, its optimal path and the path's damage profile visualization provide a rich and quick synopsis for the designer. Using this in an interactive exploration such as Design Galleries [15] should provide a rich tool for the designer to experiment with different cover configurations, cover amounts and cover shapes. Having knowledge of the amount of damage a player can sustain leads

to an iterative system where the search is expanded to add more and more cover until the overall damage (or peak damage) along a path falls below some threshold.

Figure 7 Screen shot of an FPS game using our cover placement algorithm. Predetermined turret placement (camouflage items) corresponds to the enemy locations. Large trees are used for cover.

8. REFERENCES

[1] 343. Industries, "Halo 4: Waypoint," Microsoft Studios, 2012. http://www.halowaypoint.com/halo4/en-us/.

[2] Epic Games, Inc., "Unreal Tournament 3," Epic Games, Inc., 2006. http://www.unrealtournament.com/.

[3] Edios Montreal, "Deus Ex: Human Revolution," Square Enix, 2011. http://www.deusex.com/.

[4] Arkane Studios, "Dishonored," Bethesda Softworks, 2012. http://www.dishonored.com/.

[5] Sicart. Miguel, "Defining Game Mechanics," *The International Journal of Computer Game Research*, December 2008.

[6] Mark. Davies, "Examining Game Pace: How Single-Player Levels Tick," 2009 http://www.gamasutra.com/view/feature/132415/examining_game_pace_how_.php?print=1.

[7] Treyarch, "Call of Duty: Black Ops II," Activision, 2012. [Online]. Available: http://www.callofduty.com/blackops2.

[8] Luigi Cardamone, Georgios N. Yannakakis, Julian Togelius, and Pier Luca Lanzi. 2011. Evolving interesting maps for a first person shooter. In Proceedings of the 2011 international conference on Applications of evolutionary computation - Volume Part I (EvoApplications'11), Cecilia Di Chio, Stefano Cagnoni, Carlos Cotta, Marc Ebner, and Anikó Ekárt (Eds.), Vol. Part I. Springer-Verlag, Berlin, Heidelberg, 63-72.

[9] Christian Guttler and Troels Degn Johansson. 2003. Spatial principles of level-design in multi-player first-person shooters. In Proceedings of the 2nd workshop on Network and system support for games (NetGames '03). ACM, New York, NY, USA, 158-170.

[10] Kenneth Hullett and Jim Whitehead. 2010. Design patterns in FPS levels. In Proceedings of the Fifth International Conference on the Foundations of Digital Games (FDG '10). ACM, New York, NY, USA, 78-85.

[11] M. Csikszentmihalyi, Flow: The Psychology of Optimal Experience., New York: Harper and Row., 1990.

[12] Lennart Nacke and Craig A. Lindley. 2008. Flow and immersion in first-person shooters: measuring the player's gameplay experience. In Proceedings of the 2008 Conference on Future Play: Research, Play, Share (Future Play '08). ACM, New York, NY, USA, 81-88.

[13] Penelope Sweetser and Peta Wyeth. 2005. GameFlow: a model for evaluating player enjoyment in games. Comput. Entertain. 3, 3 (July 2005), 3-3.

[14] Nathan Sorenson and Philippe Pasquier. 2010. Towards a generic framework for automated video game level creation. In Proceedings of the 2010 international conference on Applications of Evolutionary Computation - Volume Part I (EvoApplicatons'10), Cecilia Chio, Stefano Cagnoni, Carlos Cotta, Marc Ebner, and Anikó Ekárt (Eds.), Vol. Part I. Springer-Verlag, Berlin, Heidelberg, 131-140.

[15] J. Marks, B. Andalman, P. A. Beardsley, W. Freeman, S. Gibson, J. Hodgins, T. Kang, B. Mirtich, H. Pfister, W. Ruml, K. Ryall, J. Seims, and S. Shieber. 1997. Design galleries: a general approach to setting parameters for computer graphics and animation. In Proceedings of the 24th annual conference on Computer graphics and interactive techniques (SIGGRAPH '97). ACM Press/Addison-Wesley Publishing Co., New York, NY, USA, 389-400.

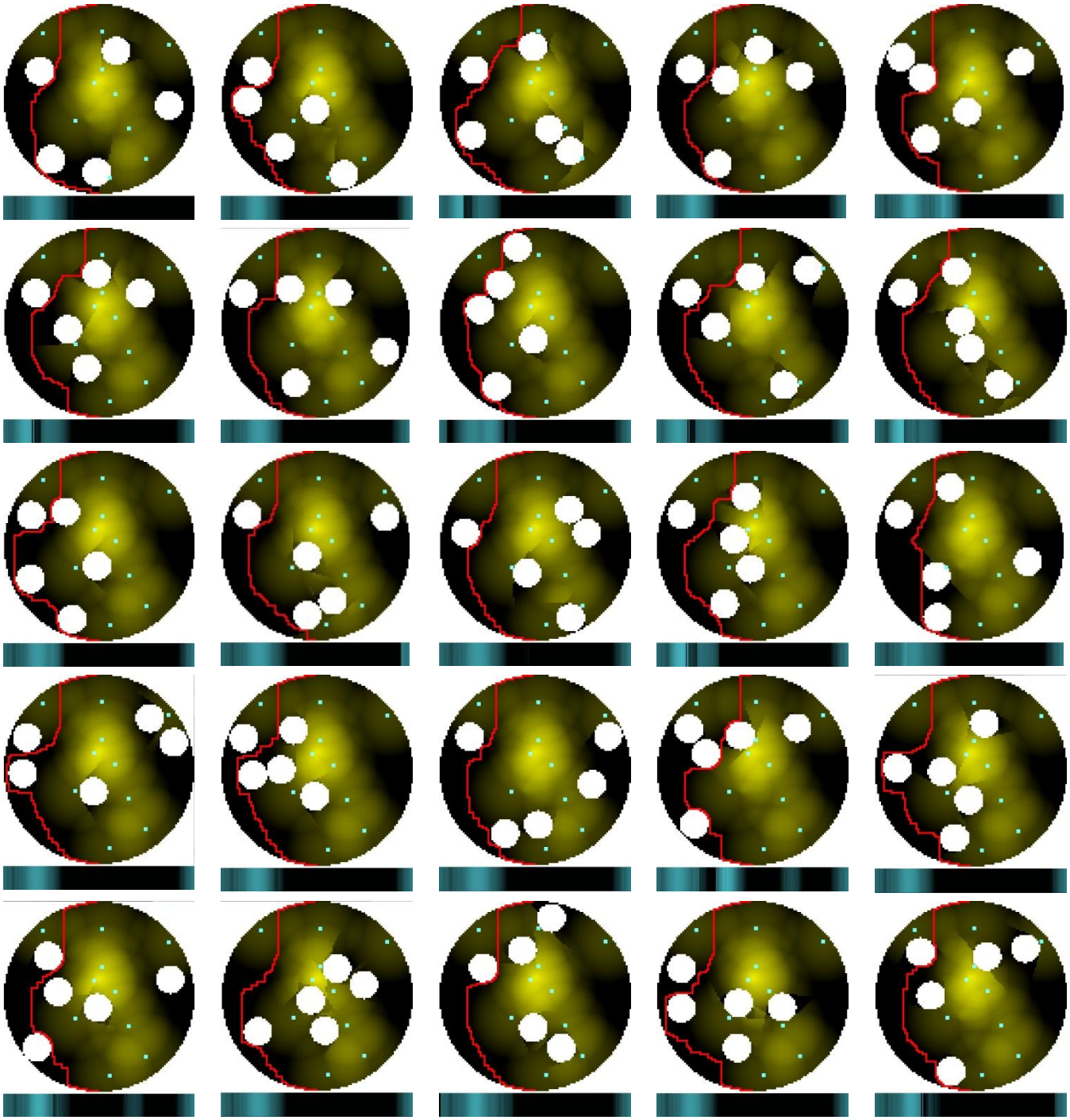


Figure 8: Twenty-five different damage maps and bars. Damage bars are color coded to represent the damage received along the optimal path. The entry point of the optimal path is mapped to the left and the exit point is mapped to the right. Darker areas indicate low damage while brighter areas mean high damage. Enemy locations are marked as blue points on each damage map. The optimal path is marked as a red line for each map. Damage maps are generated with the same enemy distribution and a fixed amount of cover.