Procedural Content Generation for Computer Games

Dissertation

Presented in Partial Fulfillment of the Requirements for the Degree Doctor of Philosophy in the Graduate School of The Ohio State University

By

Yinxuan Shi, B.S.

Graduate Program in Computer Science and Engineering

The Ohio State University

2016

Dissertation Committee:

Dr. Roger Crawfis, Advisor

Dr. Yusu Wang

Dr. Eric Fosler-Lussier

Dr. Neelam Soundarajan
Copyrighted by

Yinxuan Shi

2016
Abstract

Procedural Content Generation (PCG) is no new concept for the gaming industry. From early games like Rogue (1980) and The Sentinel (1986) to more recent games like Diablo III (2012) and Path of Exile (2013), PCG is heavily used in dungeons, quests, mini bosses and even storyline creation. The advantages PCG offers is not just limited to empowering game designers with fast content prototype/creation, but can also provide in-game adaptation to player’s response and small memory footprint. While there is much research on PCG, few results contribute to the evaluation: Does the generated content makes the game more interesting/fun? To answer this question, we examine two applications of PCG. One is level creation and another is visual content creation such as crowds.

For level creation, the existing techniques mainly focus on map/terrain generation. In games where the player either avoids or engages in combat against hostile targets, the player’s experience involves other aspects such as enemy and resource placement and navigation. The problem of creating a fun level can be formulated into searching for a good combination of these aspects. This leads to two problems: 1. How to evaluate the fun of a level? 2. How to constrain/sample the parameter space to produce a viable result in limited
We tackle the first problem by placing a pseudo player into the level. A damage function is proposed to encode the flux of damage at every point in space throughout the level. For a shooter game, 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). For a strategy game, we assume there is an optimal strategy for choosing paths for a small team to cross the damage field. With three different metrics which we defined, we are able to analyze a level by analyzing the optimal path(s). However, this search is NP. For the second problem, consider a level with a given terrain and entry and exit positions. All the possible configurations for enemies and resource placement are infinite. To better sample the parameter space, we lay down \( n \) candidate locations for enemies/resources. The problem is then transformed into a combinatorial problem. We divide the level by a grid and solve each grid cell for a fun enemy and cover combination. Rather than finding the optimal configuration out of \( 2^n \) possibilities, we treat each grid as a tile, with a precomputed tile set, we are able to obtain a fun level by finding a fun ‘tiling’ representation.

The second application for PCG is the visual content. Visual realism and plausibility are the top criteria for assessing immersive experience in games. Here we investigate the representative distribution of body shapes when simulating crowds in games. Achieving representative and visually plausible body-shape variation while optimizing available resources is an important goal. We present a data-driven approach to generating and selecting models with varied body shapes, based on body measurement and demographic data from the CAESAR anthropometric database. With a perceptual study to explore the
relationship between body shape, distinctiveness for bodies close to the median height and girth, we found that the most salient body differences are in size and upper-lower body ratios, in particular with respect to shoulders, waists and hips. Based on these results, we propose strategies for body shape selection and distribution that we have validated with a lab-based perceptual study. Finally, we demonstrate our results in a data-driven crowd system with perceptually plausible and varied body shape distribution that can be used in games.
Vita

2006 – 2010
Bachelor of Science,
Microelectronics and Solid-state
Electronics, University of Electronic
Science and Technology of China

2010 – present
PhD student in Department of
Computer Science and Engineering,
The Ohio State University

Publications

Yinxuan Shi, Jan Ondrej, He Wang, Carol O’Sullivan, “Shape Up! Perception Based Body Shape Variation for Data-driven Crowds”, SIGGRAPH Asia, 2016 (under review)


Yinxuan Shi and Roger Crawfis, “Group Tactics Utilizing Suppression and Shelter”. In Proceedings of Computer Games: AI, Animation, Mobile, Multimedia, Educational and Serious Games (CGAMES), Louisville, KY, 2014


Fields of Study

Major Field: Computer Science and Engineering
# Table of Contents

Abstract ................................................................................................................................. ii

Vita........................................................................................................................................ v

List of Tables .......................................................................................................................... ix

List of Figures .......................................................................................................................... x

Chapter 1 : Introduction ....................................................................................................... 1

Chapter 2 : Optimal Cover Placement for Stealth Game ....................................................... 7

2.1 Introduction ..................................................................................................................... 7

2.2 Related Work .................................................................................................................. 10

2.2.1 Level Design ............................................................................................................ 10

2.2.2 Game Experience ...................................................................................................... 11

2.3 Method ............................................................................................................................ 12

2.3.1 Damage Function ..................................................................................................... 13

2.3.2 Optimal Path for a Fixed Origin with Cover Placement $c$ ........................................ 14

2.3.3 Improve the Cover ..................................................................................................... 15

2.3.4 Feasible Covers .......................................................................................................... 16

2.3.5 Useful Design Metrics ............................................................................................... 17

2.4 Implementation ............................................................................................................... 18

2.5 Experiment Result .......................................................................................................... 21

2.5.1 Uniform Enemy Distribution ..................................................................................... 22

2.5.2 Discrete Enemy Distribution ..................................................................................... 26
2.6 Custom Damage Curve ................................................................. 28
2.7 Unity Demo ................................................................................. 30
2.8 Conclusion .................................................................................. 31

Chapter 3 : Group Tactics Utilizing Suppression and Shelter .................... 34

3.1 Introduction .................................................................................. 34
3.2 Related Work ............................................................................... 36
3.3 Setups and Definitions .................................................................. 38
    3.3.1 Cover Location ..................................................................... 39
    3.3.2 Moving Damage Function .................................................. 41
    3.3.3 Adding Suppression Fire ...................................................... 42
    3.3.4 Cover Graph ........................................................................ 45
    3.3.5 Cover State Graph .............................................................. 46
3.4 Optimal Squad Movement ............................................................ 50
3.5 Implementation Details .................................................................. 52
3.6 Evaluation .................................................................................... 56
3.7 Conclusion ................................................................................... 61

Chapter 4 : Large Open Map Construction Using Path Tiles ..................... 63

4.1 Introduction .................................................................................. 63
4.2 Related Work ............................................................................... 65
    4.2.2 Wang Tiling ........................................................................ 66
4.3 Tile Design ................................................................................... 67
    4.3.1 Damage Function ............................................................... 67
    4.3.2 Edge Condition ................................................................... 68
    4.3.3 Cover Edge .......................................................................... 70
    4.3.4 Path through Tiling ............................................................. 72
4.4 Tiling ............................................................................................ 78
    4.4.1 Navigation Tiles ................................................................... 78
    4.4.2 Tiling Algorithm ................................................................. 79
4.5 Path Tile Generation ...................................................................... 81
    4.5.1 Generating Cover Edge ....................................................... 82
4.5.2 Searching for the Best Path Tiles................................................................. 84
4.6 Results........................................................................................................... 87
  4.6.1 Tile Set .................................................................................................... 88
  4.6.2 Tiling with Single Set ............................................................................ 93
  4.6.3 Tiling with Three Tile Sets ................................................................. 95
4.7 Conclusion .................................................................................................. 96

Chapter 5 : Shape Up! Perception based Body Shape Variation for Data-driven Crowds99

  5.1 Introduction............................................................................................... 99
  5.2 Related Work............................................................................................ 102
  5.3 Model Selection and Creation................................................................. 104
  5.4 Experiments ............................................................................................ 110
    5.4.1 Result ................................................................................................ 113
  5.5 Conclusions............................................................................................. 118

Bibliography .................................................................................................... 126
List of Tables

Table 5.1: The 27 CAESAR body measurements used to generate our 3D models...... 106
Table 5.2: All significant correlations ................................................................. 116
List of Figures

Figure 2.1 : An arbitrary room 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. ........................................................................................................... 8

Figure 2.2. Example set up of area S where player is marked as blue diamond and enemies are marked as red stars. ........................................................................................................................................ 13

Figure 2.3 Sample images of feasible covers. (a)Screenshot from Deus Ex: Human Revolution. The player is taking cover from the enemies. (b) Paint ball game where the players utilize the covers to hide from each other. ....................................................................................... 16

Figure 2.4 : Flow of our theoretical framework................................................................. 19

Figure 2.5: Results generated for a fixed amount of cover within a circle and equal damage emanating from the boundary of a circle. Cover is indicated by the hatch pattern. 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 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. ................................................................. 23

Figure 2.6: The damage/path length graph by three selected path in Figure 2.5. Red line is the minimum damage path, blue line is the longest path and the black line is the largest standard deviation path. .................................................................................................................. 25

Figure 2.7: Result generated from a fixed distribution of enemy inside S. Purple points indicate the enemies. Same as Figure 2.5, cover is represented by the hatch pattern. Damage function has high values mapped to bright yellow and low value mapped to black. (a)(b)(c) are the cover configurations’ damage maps with optimal paths........................................... 27
Figure 2.8: Damage/Path Length by three chosen paths in Figure 2.5 ........................................ 28

Figure 2.9: The screenshot of the application that supports custom damage curve. The left panel indicates the beginning setup for the scene, with enemies denoted as magenta cylinders and covers as wood blocks. The damage map is visualized via the shade of yellow underneath, with the red optimal path. On the right panel, the green line is specified by the user. The red line indicates the damage curve along the optimal path on the left. .............. 29

Figure 2.10: Screen shot of a stealth game using our cover placement algorithm. Predetermined turret placement (camouflage items) corresponds to the enemy locations. Large trees are used for cover. ................................................................. 30

Figure 2.11: 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............. 33

Figure 3.1: (a) Screenshot from Call of Duty: Black Ops. (b) Screenshot from X-COM: Enemy Unknown. ....................................................................................................................... 35

Figure 3.2: Node “I” indicates inside node, “O” means outside node. Pinch location is represented with “N”. Grey nodes are potential ambush points. ........................................... 37

Figure 3.3: Sample setup of four labeled objects. The squad is marked as yellow dots while the enemies are marked as cyan dots. ................................................................. 39

Figure 3.4: (a) damage map generated by enemy locations and object locations. The brightness of magenta indicates the expected damage received at the current location. (b) The candidate cover locations are marked in green...................................................................... 41

Figure 3.5: (a) Right squad member moves while the left one pins down the enemy marked as yellow dot. The damage map is shown after the suppression. (b) The damage map after the left squad member has advanced. ......................................................... 42

Figure 3.6: Part of the squad moves while the rest cover for them. ........................................ 44
Figure 3.7: (a) A complete graph generated by Figure 3.4(b). (b) A cover graph pruned from (a) using the set of rules. ................................................................. 45

Figure 3.8: Partial cover state graph from Figure 3.7(b) ............................................. 49

Figure 3.9: Flow chart for our algorithm. ................................................................. 53

Figure 3.10: White lines indicate the possible enemy suppression. The green line depicts squad member at cover 4 trying to move to cover 7................................................. 54

Figure 3.11: Set up for our test scenario. Enemies are marked with cyan color and objects are textured as wood. ........................................................................ 57

Figure 3.12: 10 steps selected from the optimal squad tactics. Lines between the squad members and the enemies show the suppression strategy from static squad members from current step to next step. Notice how the damage map (shown in magenta) changes are different enemies pinned down with suppression fire. ................................................. 58

Figure 3.13: Greedy approach to move the squad team. Figure (a)(b)(c) show consecutive steps. (d) to (e), (f) to (g) (h)to(j) are 3 sets of consecutive moves. It takes the wrong path and fails its mission................................................................. 59

Figure 4.1: (a) Tile $T_1$. (b) Tile $T_2$. (c) Example of placement of two tiles.............. 68

Figure 4.2 Enemy locations reflected to the left tile...................................................... 69

Figure 4.3: (a) White strip indicates the cover edge. (b) Red line marks the solution path for the left tile given the tiling entrance......................................................... 71

Figure 4.4: (a) Solution path are represented as red lines. (b)Alternative solution path that is also valid. Target zones are depicted as blue segment along the edge................. 74

Figure 4.5: A tile with target zone $Z_1$ on the left edge and $Z_2$ on the right edge. $x_1$ and $x_2$ are two random locations on each target zone. Point $y$ is located outside the target zones. ..................................................................................... 76
Figure 4.6: The list of 6 navigational tiles. The connection between edges are marked as black lines. ................................. 78

Figure 4.7: (a) The blue line represents the path specified by designer. (b) The fitted out designer path with navigation tiles. (c) The complete tiling with path tiles. The yellow line depicts the tile paths. (d) Another designer’s sketch with shorter path. (e) Sketched path in (d) are replaced with navigation tiles. (f) The complete tiling of (d). The cover and enemy locations are not shown for clarity. ......................................................... 80

Figure 4.8: The process of tile creation. ................................................................. 81

Figure 4.9: (a) Cover edges are marked as red rectangular strips. The vertical cover edge overlaps the horizontal one. The enemies’ effective radius is $r$. (b) The corner is marked as blue octagonal shape......................................................... 82

Figure 4.10: Vertical cover edges. The covers are marked by orange squares while the enemies are marked as magenta dots. (a) Vertical cover edge without target zone. (b) Vertical cover edge with target zone. .......................................................... 83

Figure 4.11: (a) Path tiles generated with largest standard deviation metric. (b) Path tiles generated with longest tile path length metric. (c) Plotted damage curve. Red line represents the damage curve for (a), blue one represents for (b). ......................................................... 87

Figure 4.12: The list of path tiles generated with largest standard deviation metric. Enemies are marked with magenta dots; covers are marked as orange squares. Each tile is colored with the damage. The brighter the color is, the higher the expected damage of the area. Each tile corresponds to a navigation tile in Figure 4.6................................. 89

Figure 4.13: Path tile set generated with the longest path length metric. ...................... 90

Figure 4.14: The path tile set generated with the minimum damage metric.................... 91

Figure 4.15: (a) The tiling generated with semantic navigational tiles. (b) The complete tiling with the path tile set generated with largest standard deviation metric in Figure 4.11. .................................................................................................................. 92
Figure 4.16: (a) Tiling with semantic navigational tiles. (b) The complete tiling using all three tile sets. Cameras (used in Figure 4.15) are marked by a logo and its cone of vision is marked in green. ........................................................................................................................................... 94

Figure 4.17: (a) (b) (c) Screen shots from a scene created with the tiling in figure 14(b). Screen shot in (a) is taken in left most camera location of figure 14(b). (b) is taken by the camera in the middle and (c) is taken at the right most location. (next page) .................. 97

Figure 5.1: Pictures of crowds. (a) Crowds portrayed in Hitman Absolution. (b) Real crowd in New York City subway. The crowd in game is more stylistic (too close to the average) than the real crowd............................................................................................................................................. 101

Figure 5.2: The CAESAR database distribution of body heights and girths, with red dots showing the samples we picked to generate our template models; the red cluster in the median group shows the samples used to generate our experimental stimuli. The total number of samples within each quadrant is shown at its top left corner. .................... 104

Figure 5.3 Examples of 3D models generated from the CAESAR sample measurements. ............................................................................................................................................. 107

Figure 5.4 Daz3D screenshot...................................................................................................................................................... 107

Figure 5.5: Top: Close-to-median body shapes with various colored clothes (17 males and 14 females). Bottom: Hour-glass shaped pear-shaped female with V-shaped and block-shaped male............................................................................................................................................. 109

Figure 5.6: Example models used in Experiment 1, sampled near the median: models picked to represent the median group (left); most distinctive and most/least attractive models (right).............................................................................................................................................. 110

Figure 5.7: Survey page used in our experiment. ............................................................................................................................................. 112

Figure 5.8: Distinctiveness and Attractiveness results from Experiment 1: the red dots show the models we chose for the median group. The average Male and Female models were also chosen. ............................................................................................................................................. 117

Figure 5.9: Close-up from the crowd simulation. ............................................................................................................................................. 119
Chapter 1 : Introduction

In the realm of game content creation, good gameplay experience desired by all game makers. A player’s experience is affected by game mechanics, levels, visual presentation, sound effects, the storyline, and social interaction etc. Without a rigorous definition of gameplay experience, the evaluation of such an experience can be quite challenging. Our research focuses on building a framework; it employs metrics based on gameplay experience to evaluate the procedurally generated game content. In this dissertation we only investigate the procedural level creation and visual content creation, as well as their evaluation. For level generation, we model the player’s experience by simulating a virtual player in the game scene. For visual content, we inspect its realism and plausibility.

Level creation and evaluation is an important part of procedural content generation. The existing techniques for game level creation mainly focus on map/terrain generation. Terrain/maps can serve multiple purposes for gameplay. For example, in a strategy game like Civilization, island terrain will emphasize on a navy-oriented strategy while Pangaea terrain will focus more on land-based tactics. The designer relies on terrain to provide desired tactical information. In games strongly related to the story line, the terrain/map also
provides the desired set up for the scene. E.g. in recent games like *Max Payne 3*, one of the main quests requires the player to go through the football field to look for clues. This restricts the terrain/map to be based on a realistic looking football field where objects such as chairs, pillars (Where a player may take cover) are arranged in a symmetric manner. Meanwhile, the player’s experience of a level involves other aspects like enemy placement, resource placement, and navigation etc.

In our research, we focused on procedural level design with object placement. There are two types of objects that we examined.

1. Objects a player utilizes to take momentary cover during the game, such as a crates/pillars. It may shield the player from incoming fire.

2. Hardened enemies that are capable of pining down the player within a certain range.

In chapter 2, we propose a theoretical framework to search for a good configuration of cover placement for stealth-like games. The placement of objects greatly impacts the gameplay and the strategy a player may take towards progressing through the game. For example, given an area $S$, we assume the player will have different plans according to the different placement of objects:

1. If there are no enemies, the player will seek the fastest way to get out.

2. If there are several enemies and covers in the area, the player will take shelter at the covers while progressing to the exit.
During the player’s path to the exit, he/she may experience different incoming fires from the enemies. For instance, while the player is dashing through the area exposed to enemies’ line of sight, he/she may encounter severe fire. This leads to an intensive gameplay experience, which increases the player’s anxiety level. On the other hand, when the player is hiding behind the cover, he/she is in a more relaxed state. This leads to more peaceful gameplay. With different cover placement, the player will undergo different gameplay.

The pacing, known as flow [1], provided during the game affects a player’s experience. With an intermittence of anxiety and relaxation, the player is likely to enter the flow state [2]. By observing the behavior of the virtual player, we are able to infer the gameplay experience for the current game scene. If it fits the design goals, such as desired flow, a good cover placement is found.

Cover placement is also used in strategy games, such as Shadow Run and X-COM: Enemy Unknown. Rather than simulate one virtual player, we have a squad team to progress towards the goal. For future work, we may be able to pick a good cover placement in the same manner by evaluating the AI squad’s strategy,

In chapter 3, we introduce a method to find the global optimal strategy where the total damage to the squad is minimized as they breach towards the objective. When the squad moves, a portion of troops provide suppression fire while the rest of the squad advances to a new cover location safely. Traditional approaches to squad movement apply a greedy based strategy and do not consider suppression fire. To better understand this problem and provide a firm foundation for other squad movement strategies that wish to employ
suppression fire cover, we have developed a graph theoretic approach that solves for the globally optimal movement. We show that while this approach is PSPACE in terms of the number of cover locations, there are ways to mitigate this cost. We compare and contrast this approach to common greedy methods for squad movement, showing configurations where greedy algorithms fail. The optimal squad strategy can also be used as comparison for other game AI.

Tiling is widely used in computer graphics to produce a continuous representation of the texture, noise and geometry etc. With various shape and form of tiles, there is a large amount of literature dedicated to explore the questions of tile assembling (if the tile set can cover a plane) and periodicity (if a repetitive pattern can be found during the tiling) [3] [4]. Introduced by Hao Wang, Wang Tiles are defined as a set of square tiles with colored edge [5]. He conjectured that there must exist a periodic tiling when using a finite set of Wang tiles to tile a plane. Refuted later by Berger [6], a finite set of Wang tiles, as little as 13 tiles, can tile a plane aperiodically [7]. Wang tiling is preferred over other tiling methods since 1. The square shape avoids the complication of tile assembling. 2. The aperiodicity offers more variety with limited tiles in a tile set.

In Chapter 4, we employ our framework to procedurally generate and evaluate the large open game scene. We borrow the idea of Wang tiling to create a non-repetitive large scale open map with a precomputed set of tiles. We define those tiles as path tiles. Every path tile is constructed with covers and hardened enemies. Each tile encodes a potential path (the minimal damage path) the virtual player takes through the tile. In a similar manner,
the evaluation for each tile is conducted via analyzing the such path. The configuration of
cover and enemy locations of the tile will guide the player from the desired entry to the
exit. We describe our tile generation algorithm and a tiling algorithm that produces a
desired large open map. With various metrics to analyze the virtual player’s experience
within a tile, we are able to demonstrate and compare a variety of cover tile generation and
several use cases of tiling. With the set of tiles, we are able to support more complicated
design patterns.

In chapter 5, we examine the visual aspect of the procedural content generation, specifically
for crowd. In this case, the gameplay experience relies on visual realism and plausibility.
For the crowd simulation, it includes the plausibility for each individual avatar’s
appearance and behavior for the close-up shot and the realistic variety and the distribution
of the crowd for the long shot. While modeling each individual would offer great variety
for crowd simulation, the cost would be prohibitive. Therefore, finding and visualizing
template avatars that exhibit the variety and distribution of a real world crowd is our goal.

Based on the perceptual studies on human appearance and motion [8] [9], the variety of the
crowd’s appearance is more salient than the variety of crowd’s motion. Disregarding the
clothes and color variation, the most salient features of crowd’s appearance are people’s
shapes and sizes, with their relative distribution.

We examined the Civilian American and European Surface Anthropometry Resource
Project (CAESAR) database [10] for the 3D scan, measurement and demographic data.
CAESAR database contains subjects sampled base on their age, gender, ethnicity across
US and Europe. The most salient body shape feature are height and girth. However, with the subjects closer to the median height and girth, other factors dominate the saliency of the body shape. Therefore, we conducted two perceptual studies to evaluate the distinctiveness for median avatars and investigate these factors. The result shows that when the subject is close to median height and girth, his/her upper to lower body ratio is the dominating effect for distinctiveness. Based on that, we are able to select the template avatars to efficiently increase the variety of crowd simulation (with clothes and color variation).
Chapter 2 : Optimal Cover Placement for Stealth Game

2.1 Introduction

Action or action-adventure games that involve navigating a space, typically contain combats where a player engages hostile targets using melee or ranged weapon. Levels are used to divide the game into sections. Level design can be crucial for determining whether the game is mundane or exciting [11]. Level design includes map/navigation area generation, tactical object and resource placement, aesthetics, story line design, quest generation, etc.

Stealth games, as a sub-genre of action or action-adventure games, usually have more emphasis on avoiding enemies by taking cover rather than actively engaging with enemies. Mentioned by Rogers [12], taking cover can be viewed as a combat element in a game and is one of the basic game mechanics, giving it a significant impact on the level design.

Consider the scenario in Figure 2.1, the player starts at position A and has as a goal to make it to position B alive. In the absence of cover or nearby cover, the player will likely take the shortest path (in this case, a straight path) to the destination. Given the enemies in the
area, the player will have to ‘run and gun’ also known as ‘shmup’ [13], which yields an intensive gameplay. However, when provided with cover, as shown in Figure 2.1 (b) and (c), the player may take a stealthier, smarter or safer route towards the destination. This gameplay is also known as ‘stop and pop’.

![Figure 2.1](image)

Figure 2.1 : An arbitrary room 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.

Note that in Figure 2.1 (c), in between the two covers, the player may experience ‘run and gun’. The interleaving gameplay of the ‘run and gun’ and the ‘stop and pop’ creates a more stimulating game experience. And with a good cover placement, game designers will be able to create a desired variety of gameplay.

Pacing, also known as flow, is a concept that describes the player’s perception of the level [14]. A well-paced level is a mix of action/peaks and calm/troughs. In games such as *Dishonored* and *Deus Ex*, each level contains a different pace. Being in a trough for too long will lead to tedious and boring gameplay, while being in peaks for too long will
desensitize a player to repetitive motion. Therefore, game designers must balance the peaks and troughs in order to create a good flow for the player, one where he/she feels enjoyment and control in an autotelic activity [1].

There has been a number of research papers for level design of games. Many of which are geared towards procedural map generation. There is relatively little research on object placement, which is another crucial aspect of level design. Game designers/testers are still heavily reliant on manually tweaking game assets to provide cover and enemy placement. However, there are other aspects to consider in game design, such as the architectural feasibility and aesthetics. For example, a building design might be based on the Parthenon Temple where the pillars are intended for cover. In this case, the pillars need to remain symmetric rather than being tweaked solely to optimize gameplay. To solve this, enemy placement may be optimized to favor gameplay. Even for outdoor scenes, the designer will still have a hard time creating a configuration that optimizes gameplay due to limited time and play testing.

With this in mind, our goal is to develop a method to generate optimal cover for stealth games. First we examine the problem of finding an optimal path given a cover and enemy (NPC) distribution with their given behaviors. There is a duality between optimizing the cover for a fixed enemy/NPC distribution and optimizing the NPC distribution for a fixed cover. This chapter will focus on the former, but can easily support the latter in our framework. Both enemy and cover distribution will affect the optimal path.
Second, we look into how the placement of cover affects optimal paths and propose a framework for searching for an optimal cover. Finally, we examine how designers can utilize the framework by specifying the desired flow curve for which the cover should be optimized towards.

The remainder of this chapter is arranged as follows: Section 2.2 outlines the related work. In section 2.3, a theoretical framework is proposed for paths, cover distributions and enemy distributions. In section 2.4, implementation details of our approach to creating possible candidate covers and selecting the optimal cover is discussed. In section 2.5, an analysis of our approach is provided along with examples and use cases that show the advantages and flexibility of our approach.

2.2 Related Work

Procedural content generation significantly lowers the cost for game production and yields almost infinite replay value. As such, it has drawn increasing attention in the research field.

2.2.1 Level Design

There are several level design techniques which have been developed. Guttler and Johanson [15] introduced the spatial principles of multiplayer FPS level design. They defined "Collision Point" as the place for teams to encounter each other and "Tactical Choice" as the best way for the team and the player to perform the encounter. Rectangular areas are used in their research for cover location in tactical planning. However, this research does not offer any clear guidance to procedural content generation for FPS games.
Game design researchers have presented a taxonomy of FPS design patterns. Hullet and Whitehead [16] introduced several design patterns for a FPS game that includes sniper location, strong hold, gallery etc. Yet, they also do not propose a clear usage. Cardamone et.al [17] proposed a method for evolving maps for a shooter game. The novelty of their research is to combine the search-based solution to evolve maps based on player’s average fighting time.

2.2.2 Game Experience

Game experience is considered as the Holy Grail of game design. There are several terms to describe game experience: immersion, presence and flow. Introduced by Csikszentmihalyi, the flow state is an optimal state of intrinsic motivation, where the person is fully immersed in what he/she is doing [1]. The hallmark of flow is a feeling of spontaneous joy, even rapture, while performing a task. Desired by most game designers, it is still not obvious how to achieve flow state in game creation.

Nacke and Lindley [18], utilized electroencephalography, electrocardiography, electromyography, galvanic skin response and eye tracking equipment to measure a player’s physiological change during a FPS game. They also collected questionnaire responses from the participants yielding a subjective measurement of game experience. A positive correlation between subjective and objective indicators of gameplay experience was found. However, due to the cost of game testing and the long processing time, it is not feasible to apply this technique for iterative game design.
Rather than measuring a player’s physiological traits during the gameplay, Sweetser and Wyeth [2] developed GameFlow, a model for evaluating player’s enjoyment in the game. Based on the definition of flow, they developed sets of criteria of its eight elements: concentration, challenge, skills, control, clear goal, feedback, immersion and social interaction. According to gathered expert reviews of high-rating and low-rating strategy games using the criteria, GameFlow criteria is able to successfully distinguish between high-rated and low-rated strategy games and analyze the cause. However, the player’s experience is measured after the game production, it is not helpful for level design. With this idea, Sorenson and Pasquier [19] developed framework for automatically generating levels using a genetic algorithm. For each sample game, they proposed different fitness functions to measure fun. Our approach uses a similar idea, we proposed three different metrics to evaluate the fun for a cover configuration.

2.3 Method

We examined the player’s probable path. Intuitively, the player will lean towards the path with least resistance. To find such a path, the resistance needs to be formulated.
Figure 2.2. Example set up of area $S$ where player is marked as blue diamond and enemies are marked as red stars.

### 2.3.1 Damage Function

Assume a level in the stealthy/shooter game like *Dishonored* contains a closed area $S$, which the player needs make it across. As depicted in Figure 2.2, enemies marked as red stars are distributed in $S$. The player at location $\vec{x}$ is represented by the blue diamond. The vector $\vec{v}$ illustrates the direction from the player to one enemy. Without other distractions, we can assume all the enemies in $S$ will target the player provided the visibility of the player. Note that $|\vec{v}|$ indicates the distance between the enemy and the player, which affects the shooting accuracy from an enemy at position, $\vec{x} + \vec{v}$. Intuitively, the further a player is away from the enemy, the less likely he/she will be shot. This is conveyed as a function of $|\vec{v}|$. At a given time $t$, the amount of damage the player receives from the enemy can be denoted as $damage(\vec{x}, \vec{v}, t)$. The total damage the player receives in $S$ is computed as the integral of damage received all enemies’ directions (Equation 2.1).

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

(2.1)
When the player is exposed to a large amount of enemies, the player will suffer more damage as more and more enemies target him/her (perhaps in a non-linear fashion to represent a stun affect). This feature may represent the game mechanics when the player is slightly stunned after being hit and has more trouble defending or seeking covers for himself/herself. The damage function can be biased to penalize the player from going into enemy condensed area.

Assume the player’s path across $S$ is denoted as $\text{path} = x(t)$. The damage function can be formulated with respect to $t$ as follows:

$$
\text{Damage}_{\text{path}}(t_1, t_2) = \int_{t_1}^{t_2} \text{total}(x(t), t) \, dt
$$

Equation 2.2 indicates the overall amount of damage the player received from time $t_1$ to time $t_2$.

### 2.3.2 Optimal Path for a Fixed Origin with Cover Placement $c$

Based on our assumption, the *optimal* path is the path with the least resistance. More formally, optimal path is the path with minimum damage among all possible paths connecting A to B, over all time periods:

$$
\text{Optimalpath} = \min_{\text{paths}} \{ \text{Damage}_{\text{path}}(t_1, t_2) \}
$$

This function associates optimal path with the damage function. An obstacle can act as covers to block the enemies’ line of sight, this will reduce the damage received by the
player. The damage function is thus related to all the covers in $S$. Therefore, modifying the cover locations in $S$ changes the optimal path and the flow of damage across the path accordingly.

If we fix all covers’ locations, we can use the same equation to find the optimal enemy placement. There is a duality between enemies and covers. In this chapter, we only examine the search for cover placement. Next, we examine optimal cover and our search process.

### 2.3.3 Improve the Cover

Given an enemy distribution, assume we have a set of possible covers, $C$, where each cover set $c \in C$, has an optimal path corresponding to it. We denote that path as $OptimalPath_c$. Therefore, we define optimal cover as the cover which minimizes some metrics. For example, the metric along the optimal path can be used to minimize the overall damage. Hence, we have:

$$OptimalCover = \min_{c \in C} \{OptimalPath_c\}$$

This biases the design towards making the level easier (or allowing more complex enemy distributions). Note that if the amount of the cover varies, the design is biased towards more cover area. In section 2.3.5, additional metrics will be discussed.
2.3.4 Feasible Covers

Figure 2.3 Sample images of feasible covers. (a) Screenshot from *Deus Ex: Human Revolution*. The player is taking cover from the enemies. (b) Paint ball game where the players utilize the covers to hide from each other.
The coverage in the scene is typically dictated by the designers, typically selecting the exact locations for the covers. For example, in Figure 2.3(a), the designer may only want to place only one or two units of cover in the scene, given the room layout. In Figure 2.3(b), the paint ball game is in the open space, the designer may require a lot more covers to shield the players. If the designer limits the total volume of cover, denoted as $max_{cover\_volume}$, we can constrain the set of feasible covers to be the following:

$$\sum_{j=0}^{n_l} Volume(c_{ij}) \leq max_{cover\_volume}, \forall c_{i} \in \{C\}$$ (2.5)

In general, the constraints for the cover may be set by the total volume or the total amount of units as well.

### 2.3.5 Useful Design Metrics

Instead of choosing the optimal cover based on minimal path damage, we can adopt other metrics to optimize towards. If the designer desires longer gameplay through this fixed area, we may pick the cover with longest optimal path length. Similarly, if the designer requires a drastic change through the gameplay (i.e. from peaceful to agitated or vice versa), the cover set with the largest standard deviation for the damage along the optimal path may be selected. Note that the optimal path refers to the least resistance path that a player can go through in a scene. With the one to one mapping between the optimal path and cover placement, the metrics are used to evaluate cover placement via its optimal path. The minimum damage metric is used to select one optimal path with the least accumulated
damage among all other optimal paths. Similarly, the longest path length metric examines the path length for all cover placements’ optimal paths.

Our method also allows the designer to specify the flow, which dictates the damage curve for the desired optimal path. Each point on the desired damage curve is the expected amount of damage the player received along the desired optimal path. Hence, we are able to use least squares to estimate the error between the damage curve of any optimal path and the desired optimal path. This allows the algorithm to search for the optimal cover that yields the damage curve that fits the best for the flow along the optimal path.

\[
\text{OptimalCover}_c = \min_{c} \left\{ \text{DesiredProfile}_{\min \text{path}} \left( \left\{ \int_{t} \int_{R} \text{damage}(\dot{x}, \dot{v}, t) \, dv \, dt \right\} \right) \right\}, \forall c \in \{C\} \quad (2.6)
\]

Next, we will discuss the software architecture of our framework for finding optimal cover placement.

2.4 Implementation

As shown in Figure 2.4, our process is comprised of three components: 1. Generate candidate cover sets; 2. Find the optimal path for given cover configuration; 3. Select the best cover configuration based on the metrics of the optimal path.
Figure 2.4: Flow of our theoretical framework.

For the first component, our goal is to generate a set of feasible cover configurations that satisfy designer’s constraints. To achieve this, we sample the area with a set of $n$ possible cover locations, then the set will be verified by the constraints. Only the valid cover set will be added into candidate cover sets. Note that the total number of feasible covers are infinite. There are various ways to sample the area $S$, such as Poisson disk sampling, random sampling and probability sampling etc. Here we use random sampling for simplicity.

The second component produces an optimal path for the given cover configuration. For time-varying distributions of enemies or damage, the problem gets overly complex. Given constraints on the player’s movement speed and the constraint for fixed enemy locations, we can ignore time and use a shortest path algorithm to compute the optimal path. We take each discretized grid cell in $S$ as a graph node with each node connected to either its four or eight neighbors. The edge weight is computed as the average damage of the current node and neighboring node, of which we built the adjacency matrix/list. Then the optimal path is the minimum weight path with a pair of given start and end locations.

To use A* for searching the correct optimal path, the heuristic function is required to be admissible. Common heuristic such as Euclidean and Manhattan distance are used for
estimating the distance from the current location to the end. The heuristic function increases as the distance from the current location to the exit increases. However, in this scenario, the path weight may approximate to zero from some location to the exit. Other locations may appear to be closer the exit but of higher path weight. Therefore, Dijkstra’s algorithm is used instead of A*. The performance of the algorithm is related to the size of the discretized grid.

The last part is selecting the best cover configuration. For each optimal path, we are able to obtain its score from the designer specified metrics. In this case, the metrics can be formulated as functions, such as longest optimal path length, minimum accumulated damage along the optimal path or largest variation of the damage along the path. As mentioned above, DesiredProfile can be a part of the metric as well. In equation 2.6, the DesiredProfile can be applied to the optimal path to obtain a score to measure similarities with damage profile curve.

SelectOptimalCover(C, Paths)
    bestScore ← 0
    bestCover ← ∅
    for each p ∈ Paths
        score ← Metrics(p)
        if score > bestScore
            bestScore ← score
            bestCover ← cover
    end
return bestCover
SelectOptimalCover takes input $C$ as a set of feasible cover configurations, and $Paths$ as the set of corresponding optimal paths for each cover configurations in $C$. For each path $p$ in $Paths$, we measure the score from the Metrics function. The optimal cover configuration bestCover is selected based on the highest score.

2.5 Experiment Result

We implemented our algorithm to generate several cover configurations for a given set of cover constraints. In our simulation, we assume that enemies are distributed in $S$. To speed up the simulation, the enemy flux, which is the damage from an enemy, is considered time-invariant. To encourage the player to take cover, high flux areas that are exposed to most enemies are penalized. In gameplay, the enemy is prone to attack the nearby player. To model this phenomenon, we define a kernel function that serves as a weight factor to the damage function. If the player is too far away, the kernel function will return zero as in the enemy will not attack the player. The distance between the shooter and the target dictates the likelihood for shooter to hit the target. To factor this in, the kernel function can be modelled as a simple distance decay.

The term Damage Map is introduced to indicate the damage of any location within the area. A discretized grid represents the amount of damage received when the player is on the center of the grid cell. In our experiment, we discuss two use cases, one with a uniform distribution of infinite enemies along the perimeter of $S$ (Figure 2.5), another with a discrete enemy distribution inside of $S$ (Figure 2.7).
2.5.1 Uniform Enemy Distribution

Our first use case adopts an analytical computation of the damage function. In a heavily ambushed large scale scene, such as boss fight with enemies patrolling, the player will be likely to receive incoming damage from all directions. We can simulate this level by placing infinite amount of enemies around the perimeter. In this case, we only consider the enemies that are uniformly distributed along the perimeter of $S$. There are $n$ covers, each of which is represented as a circular area with radius $r$. Covers are non-overlapping and constrained to be contained in $S$, which is also circular with a radius $R$.

Since the enemy is uniformly distributed, the damage for each discretized grid in $S$ can be represented as a percentage of the perimeter. Enemies can only attack a player when the player is exposed to them. Thus, if the line of sight from the enemy to the player is occluded by a cover, the player will not receive any attack from that direction. Consequently, the damage at that point can be computed as the percentage of the perimeter that is not occluded by the covers.
Figure 2.5: Results generated for a fixed amount of cover within a circle and equal damage emanating from the boundary of a circle. Cover is indicated by the hatch pattern. 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 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.

As stated in 2.3.1, the player is likely to be stunned in dense enemy areas, which could result in instant death. Therefore, we encourage the player to take covers and move cautiously. We achieve this feature by biasing every damage map. If the area in the map is exposed to over 70% of the circle, the damage of the area would be increased over the player’s total health, which results in an instant death.

In an area with approximately zero damage, the optimal path may display a meandering pattern, which is of the damage as the shortest distance path. However, this pattern is unnatural in most gameplay: the player is prone to choose the straight way to the destination rather than wandering aimlessly. To prevent this, we add a small amount of path length penalty to the edge weight. Now the optimal path will also incorporate the distance factor.
The entry location is at the top of the area, while the exit is at the bottom. As depicted in Figure 2.5, three different cover placements are produced using different metrics.

From 1000 generated covers, Figure 2.5(a) shows the cover configuration where the optimal path has the least amount of damage compared to the rest of the cover configurations. Covers are presented as circular areas with hatch patterns in the area $S$. The optimal path is illustrated with a red curve. The darker region indicates less damage when the player is at that location. In Figure 2.5(b) and (c) the center part in $S$ is much brighter due to a large open area - nothing occludes the enemies and falls above 70%. On contrary, in the center of area $S$ in Figure 2.5(a), it appears darker compared to Figure 2.5(b) and (c), given a larger percentage of the perimeter is occluded.

The longest path length cover is displayed in Figure 2.5(b). This cover configuration corresponds to the longest optimal path in the candidate cover configuration set. Another metric might be the optimal path with the largest standard deviation of the damage across the path, shown in Figure 2.5(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 travelling to the center of $S$, he/she may have to fight to the exit. This flow also demonstrates an element of surprise to the gameplay.

Note that all of the paths in Figure 2.5 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 2.5(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.

Figure 2.6: The damage/path length graph by three selected path in Figure 2.5. Red line is the minimum damage path, blue line is the longest path and the black line is the largest standard deviation path.

Figure 2.6 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 (black line). The amount of damage received along the path is depicted with the $x$-axis as the path length and $y$-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 a precipitous slope on the curve in Figure 2.6. This curve has a flow of restful to sudden danger. The longest path is backwards; it starts out intense then gets restful.
For Figure 2.5(a), this type of cover can be used as a relatively easy level for a player to pass. In Figure 2.5(b), more gameplay can be provided in a fixed area of $S$. As to Figure 2.5(c), the cover configuration may bring the player a surprising game experience. A damage/path profile can also be utilized to select an optimal cover as in equation 2.6. Just like in Figure 2.6, the designer will draw the desired damage curve and a most similar curve will be chosen from the candidate configuration set. To support the varying damage from enemies, a bias function may be employed to help designer to specify the high or low density of the enemies along the perimeter.

2.5.2 Discrete Enemy Distribution

For small groups of enemies stationed at certain locations within the area, we model this situation as our second use case with a discrete enemy distribution in $S$. For this use case, we adopt a room’s blueprint as area $S$. Then ten non-overlapping enemies are specifically placed inside the area. We constrain the cover so that in total there are five covers, and no overlaps between each other or enemies. The entry and exit points are placed on the boundary of $S$, at the bottom left and middle right respectively. As shown in Figure 2.7, the optimal paths connecting the entry and exit points are marked as red lines.

We sample 10,000 configurations. Cover configuration with the minimum path damage is displayed in Figure 2.7(a). Again, the darker area in the image indicates less damage. The brightest part is where the enemy is located (magenta point). The damage from an enemy falls to zero based on kernel function.
The longest path length cover is displayed in Figure 2.7(b) while the largest standard deviation of damage across the path is depicted in Figure 2.7(c). Note that the enemy distribution is predetermined by the designer, who can specify some areas as heavily guarded areas (high enemy population) and some areas as patrol areas (low enemy population).

![Image](Figure 2.7: Result generated from a fixed distribution of enemy inside S. Purple points indicate the enemies. Same as Figure 2.5, cover is represented by the hatch pattern. Damage function has high values mapped to bright yellow and low value mapped to black. (a)(b)(c) are the cover configurations’ damage maps with optimal paths.)

As shown in Figure 2.7, all the optimal paths start with low damage then wind their way to the exit with minimal total damage. The covers in Figure 2.7(a) were placed close to enemies on the top and left, thus creating a low damage zone for its optimal path. The damage of the optimal path is portrayed as a red curve in Figure 2.8. Compared to Figure 2.7(a), covers were placed between the enemies at the bottom, forcing the optimal path to take a longer route in Figure 2.7(b). Denoted as a blue line in Figure 2.8, this configuration’s optimal path has a long path length. In Figure 2.7(c), the player has to
progress through two areas with high damage (the two high peaks of the black curve in Figure 2.8), while the rest of the path has a damage near zero.

![Graph showing damage vs. path length](image)

Figure 2.8: Damage/Path Length by three chosen paths in Figure 2.5

### 2.6 Custom Damage Curve

To further demonstrate the extension of our framework, we also implemented the user interface to allow the designer to specify the desired damage curve. With the given scene and enemy distribution, the designer will also need to determine the number of covers to search for. The desired damage curve is depicted as the green line shown in Figure 2.9. Given the damage curve, the simulation runs $n$ iterations to find the cover placement that provides the optimal path that matches closest to the damage curve. The number of maximum iterations can also be specified by user. Then the final cover configuration with a damage map and an optimal path is portrayed on the left panel. The corresponding damage curve for the searched result is depicted as the red curve on the right panel (in
Figure 2.9). Note that, before comparing the current damage curve against the desired damage curve, both curves are normalized (The length is mapped to be the 0-1 and the maximum value is scaled to 1). Then similarity is computed as the sum of least squared distance between each point in current damage curve and its corresponding point on desired damage curve. However, even though there are infinite possibilities for cover configurations, there may or may not exist an exact match with desired curve.

Figure 2.9: The screenshot of the application that supports custom damage curve. The left panel indicates the beginning setup for the scene, with enemies denoted as magenta cylinders and covers as wood blocks. The damage map is visualized via the shade of yellow underneath, with the red optimal path. On the right panel, the green line is specified by the user. The red line indicates the damage curve along the optimal path on the left.
2.7 Unity Demo

Figure 2.11 illustrates several different cover placements and their damage profiles generated using a fixed discrete enemy distribution in a circular area $S$ and a constant number of cover amount. For each image, the top part shows the damage map while the bottom part shows 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 four column four, the cover placement will provide an interesting flow to the game since its damage bar follows a pattern of interleaving peaks and troughs.

Figure 2.10: Screen shot of a stealth game using our cover placement algorithm. Predetermined turret placement (camouflage items) corresponds to the enemy locations. Large trees are used for cover.

In Figure 2.10, we show a sample level of a stealth game with this cover configuration created in Unity. In this setting, trees as placed as covers and turrets as enemies. While the
player is running through the scene, the turret will turn to fire at the player on sight. This shot is taken while the player is hiding behind the tree (cover).

Our experiment is implemented with C++ and visualized using OpenGL. The Unity game is implemented with C#. The experiment was run on an Intel Core 2 Duo chip set. For the 10,000 cover configurations generated, the average speed to compute 500 configurations takes approximately 2 minutes, which is around 4 cover configuration per second. For interactive level design, our code needs to be optimized and multi-threaded. We may also use cloud computing. With multiple virtual machines running different sets of cover configurations, the results can be recorded. This may significantly improve our performance.

2.8 Conclusion

A framework for optimal cover placement under fixed enemy positions has been presented for stealth games. Our contributions include (1) a definition of the damage flux, its calculation and considerations for biasing 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 stealth related game to obtain various styles of gameplay as desired by the designer. We presented two simulations to aid in developing and proving the applicability of the framework. Both simulations ran with a discretization of the damage function. One focuses on enemy locations outside a 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 situation. The latter simulation allows the cover to be tweaked in the case of hardened or static enemy placement. The metrics allow for longer gameplay 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).
Figure 2.11: 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.
Chapter 3 : Group Tactics Utilizing Suppression and Shelter

3.1 Introduction

Believable group tactics is an important aspect of game AI. A squad’s behavior should include cooperation under adversity. Sterren [20] introduced two types of squad tactics: a centralized approach and a de-centralized approach. A centralized tactic means one squad leader makes most decisions. Whereas a decentralized tactic indicates that the interaction between squad members determines the squad behavior. In a game like Call of Duty, a decentralized companion tactic is employed. As shown in Figure 3.1(a), using a decentralized tactic the player does not directly control his companions. Allies follow the player as he/she moves. In strategy games like X-COM: Enemy Unknown, a centralized group tactic is used, where the player dictates each squad member’s movement. In Figure 3.1(b), as a centralized tactic, the player controls all the units’ movement.
Figure 3.1: (a) Screenshot from *Call of Duty: Black Ops*. (b) Screenshot from *X-COM: Enemy Unknown*. 
In this chapter, we mainly discuss centralized squad tactics under certain constraints. In centralized squad tactics, there are two general categories of command styles, listed as authoritarian command style and coaching command style [21]. The former represents the absolute obedience of squad members. For instance, if the player orders one squad member to go on a task to distract the enemies, the squad member will execute the mission without blinking an eye. Coaching command style allows the squad member to execute the task at his/her best capability. In this case, the squad member will inform the player that he/she cannot execute the task. Our research is focused on an authoritarian command style.

3.2 Related Work

The common technique for squad tactics includes two steps. First the squad leader chooses the next best location for the squad based on tactical information. Second, the whole team moves to a nearby location in a flocking manner.

Influence maps (IM) were introduced by Tozour [22] for tactical information. This technique discretizes the map to compute the safety and threat locations. This technique requires high CPU computation and memory consumption if given a large scale map. Later on, Heckel et al [23] extends IM to a navigation mesh, which significantly reduced the memory consumption. Daneilsiek et al [24] introduced a way to couple flocking and IM to obtain a believable result.
Figure 3.2: Node “I” indicates inside node, “O” means outside node. Pinch location is represented with “N”. Grey nodes are potential ambush points.

Lidén [25] demonstrates another method of gaining tactical information by pre-calculating a visibility waypoint graph to determine sniping locations, flanking locations and pinch locations for non-player characters (NPCs). Pinch locations can be understood as the connection point between inside and outside, as shown in Figure 3.2. This enables quick and economical calculations for tactical information of a large number of nodes and enemies. White [26] also adopted waypoint graphs in searching squad tactics; however, both algorithms suffer from tedious manual placement for waypoints. Straatman et al [27] combines IM, line of sight and a waypoint graphs to obtain the tactical information.

Hierarchical Task Network (HTN) planner is a popular approach in finding group tactics. Gorniak and Davis [28] presented an application to generate collaborative squad tactics in real-time. Another implementation of HTN planner on a real-time strategy game Stratagus
is demonstrated by Muñoz-Avila and Aha [29]. In spite of that, most games are domain-dependent and require optimization to make it suitable for real-time gameplay.

Genetic Programming (GP) is also used in believable squad behavior. Doherty and O’Riordan [30] presented a GP based approach of utilizing shared perception on the evolution of effective squad behaviors.

Recent techniques for generating squad tactics employ learning algorithms. Tan and Cheng proposed a hierarchical learning framework [31] for obtaining tactical behavior for the team. They construct a tree structure to simulate the chain of command and learning process purely based on previous learning. Avery and Louis introduced a method of coevolving IM [32]. However, their algorithm was not widely used in the industry due to complexity and unpredictable behavior caused by the noise of input data from the learning process. Our approach solves the global optimal for squad tactics in different settings, which can be used to evaluate the other tactics in the learning-based algorithms.

3.3 Setups and Definitions

We are given an area $S$ with several enemy locations $(q_1, q_2 ... q_l)$. A squad with members $(p_1, p_2 ... p_n)$ are on a mission to move from some location $A$ to an objective location $B$. A set of objects are located in $S$ that squad members can utilized as covers. Figure 3.3 shows a sample setup with four objects marked as rectangles, four enemies marked as cyan dots and two squad members marked as orange dots at the starting location $A$. The objective is the green door sign, $B$. 38
Figure 3.3: Sample setup of four labeled objects. The squad is marked as yellow dots while the enemies are marked as cyan dots.

3.3.1 Cover Location

The previous chapter demonstrated a way to generate a damage map for a scene with covers and enemies. Each grid in the damage map can be understood as the expectation of a squad member being shot if he/she is at the location. We continue to use the idea of damage map and compute potential cover locations based on it.

After discretizing, the expected damage received at each grid location is computed based on the visibility of enemies. The visibility function returns 1 if the point $\hat{x}$ is visible from enemy $q_j$. Similar to the kernel function in Chapter 2, the weapon, location and skill or behavior of $q_j$ determines $w_j$, the expected damage caused by $q_j$. 
\[
\text{damage}(\tilde{x}) = \sum_{j=1}^{l} w_j \ast \text{visible}(q_j, \tilde{x})
\] (3.1)

Therefore, each grid point can be evaluated for its eligibility as a cover location. Consider this common scenario: a squad is likely to take cover in locations with low expected incoming damage and close to the locations from which it is easy to shoot enemies. In this situation, cover location $C$ exists if the following two properties hold:

- Provides safety: $\text{damage}(\tilde{x}) < \sigma$
- Allows suppression: $\cup \text{Suppression}(\text{neighbor}(\tilde{x})) > \tau$

Here $\sigma$ stands for the maximum expected damage the squad member takes while staying in cover. The constant $\tau$ is the minimum expected damage that a squad member can cause to the enemies when shooting from his/her own and neighboring locations. We can filter the damage map to obtain a set of $m$ cover locations $C$ at $(c_1, c_2 \ldots c_m)$.

From the example setup in Figure 3.3, the damage map is computed in Figure 3.4(a), where the darker the magenta is, the safer the grid is. The grids marked grey are inaccessible areas that are occupied by objects/enemies. Depicted as a green dot, each cover location provides a safe place for a single squad member that also allows him/her to pin down or disable an enemy when peeking out (In Figure 3.4(b)). This way, each cover location may be understood as a stronghold for planning tactics.
Figure 3.4: (a) damage map generated by enemy locations and object locations. The brightness of magenta indicates the expected damage received at the current location. (b) The candidate cover locations are marked in green.

### 3.3.2 Moving Damage Function

The damage caused by an enemy $q_j$ as a squad member moves from location $\bar{x}_1$ to $\bar{x}_2$ is proportional to $q_j$’s sight. Therefore, the expected damage along the path for the moving squad is the accumulation of each enemies’ line of sights. We denote total damage from $\bar{x}_1$ to $\bar{x}_2$ for a squad member as:

$$MoveDamage(\bar{x}_1, \bar{x}_2) = \int_{x=\bar{x}_1}^{\bar{x}_2} \sum_{j=1}^{l} w_j * \text{visible}(q_j, path(\bar{x})) \, dx \quad (3.2)$$

Note that in equation 2.2, our accumulated damage along the path is the integral starting at time $t_1$ and end at time $t_2$. Here, since the enemy distribution remains the same, we are able to denote the damage with location $\bar{x}$. 

41
3.3.3 Adding Suppression Fire

Figure 3.5: (a) Right squad member moves while the left one pins down the enemy marked as yellow dot. The damage map is shown after the suppression. (b) The damage map after the left squad member has advanced.

In military tactics, a common maneuver performed to protect squad members is providing suppression fire for each other. Here we consider suppression fire as provided by fellow non-moving team members. Enemies that are “pinned down” are unable to fire upon the moving squad member. We assume the one-on-one matching between the non-moving squad members and enemies. One squad member can only target one enemy at a time and one enemy can be fully suppressed by one squad member.

Consider squad members in Figure 3.5(a). They are hunkered down around object 1, marked as orange dots. The right squad member is ready to head to the cover location around object 2. There are two enemies that are visible at his/her teammate’s location, the rightmost one and second to the left one. The damage he/she will receive when moving
along the path is the aggregated damage from all enemies except the one that will be suppressed. In this case, the teammate will suppress the enemy marked as a yellow dot, who casts the most damage along the path. This action will minimize the damage the moving squad member receives. Our goal is to find a match between squad members that stay under cover and enemies to minimize the damage for the moving squad member. This can be formulated into a maximum matching problem.

For suppression fire, the matching between the squad member \( p_k, 1 \leq k \leq n, k \neq i \) (\( p_i \) is the moving squad member) and the enemy \( q_j, 1 \leq j \leq l \), is represented in equation 3.3:

\[
\text{suppress}(q_j, p_k, C, \text{path}) = \begin{cases} 
1 & \text{provide cover} \\
0 & \text{otherwise} 
\end{cases}
\]  

(3.3)

This can be understood as a \( \delta \) function. The moving squad member’s path is denoted as \( \text{path} \). If enemy \( q_j \) is visible to squad member \( p_k \) and \( p_k \) suppresses \( q_j \), the function returns 1. If \( q_j \) is not in \( p_k \)’s line of sight or \( p_k \) suppresses other targets or just stay down, the function returns 0.

We build a graph between squad members and enemies. Each edge connects squad member \( p_u \) and enemy \( q_v \) is weighted by \( \int_{\vec{x}=\vec{x}_1}^{\vec{x}=\vec{x}_2} w_v \cdot \text{visible}(q_v, \text{path}(\vec{x})) d\vec{x} \). Note that, not all enemies are connected to squad members. This is because some enemies are out of squad members’ visibility range, due to enemies being by objects or just being too far away.

This produces a weighted bipartite graph. By finding the maximum weight matching on the bipartite graph, we are able to obtain the maximum amount of damage the squad can
suppress. Using the suppression tactic will ensure the moving squad member receives minimum damage from enemies while he/she advances. The damage function for the moving squad member \( p_i \) with suppression fire can be represented as

\[
damage(\vec{x}_1, \vec{x}_2) = \sum_{j=1}^{l} \left\{ \sum_{k=1, k \neq i}^{n} \left[ 1 - \text{suppress}(q_j, p_k, c, \text{path}) \right] \times \left[ \int_{\vec{x} = \vec{x}_1}^{\vec{x}_2} w_j \times \text{visible}(q_j, \text{path}(\vec{x})) \, dx \right] \right\}
\]

(3.4)

Note that in military scenarios, shown in Figure 3.6, when a squad/platoon advances, half of the team advances at a time [33]. This can be achieved by substituting the \( p_k \) with a portion of \( h \) members \( p_{k_1}, p_{k_2} \ldots p_{k_h} \) advancing and the rest suppressing the enemies.

Figure 3.6: Part of the squad moves while the rest cover for them.
3.3.4 Cover Graph

In this section, we will build a graph to better describe the movement. This graph represents the possible movements of any squad member. We name it *Cover Graph*, as a squad member moves from cover to cover locations. In most tactical games, it is reasonable to assume that a squad team travels utilizing cover as much as possible. In Figure 3.4(b), a squad member is able to move between any two cover locations, resulting in the complete graph in Figure 3.7(a). However, not all movements are permissible.

![Figure 3.7](image)

Figure 3.7: (a) A complete graph generated by Figure 3.4(b). (b) A cover graph pruned from (a) using the set of rules.
We prune this graph based on the following rules to obtain the cover graph $G$ in Figure 3.7(b).

1. A squad member will not bypass any chance to take cover. In Figure 3.7(b), if a squad member intends to go from cover 1 to exit B, he/she will choose a route by stopping at cover 7.

2. A squad member can only travel a limited distance in a certain time. Long edges such as from entrance $A$ to exit $B$ are not allowed in the cover graph.

We eliminated the edge from $A$ to $B$, based on rule (1). According to rule (2) $edge(1,B), edge(2,B), edge(3,B) ... edge(A,7)$ are discarded.

If the edge weights of the cover graph remain constant, one may find the next best location for the squad team through a greedy approach. Thus, the squad members are assigned nearby locations. However, edge weights are correlated with the covers. Since we allow the squad members to suppress enemies when they remain hunkered down, the actual edge is calculated using equation 3.4. Depending on the cover locations of the hunkered squad members taken, they may pin down different enemies. The edge weight will then be changed. Hence the naive search for minimum damage strategy of the whole team on the cover graph is not sufficient.

3.3.5 Cover State Graph

We introduce a new state representation which encodes the location information of the squad. Each squad member must start and end at a valid cover location (no Rambo or
Colonel Braddock), so our set of states is a pairing of squad member $p$ to cover location $c$, where $C$ stands for the set of all cover locations:

$$S_t = \{(p_1, c_{i_1}), (p_2, c_{i_2}) \ldots (p_n, c_{i_n})\}, \ (c_{i_1}, c_{i_2} \ldots c_{i_n} \in C) \quad (3.5)$$

This representation indicates which squad member is taking which cover. Differentiating between squad members will allow us to assign different personality, different weapon/gear and responsibilities to individual squad members. However, memory consumption to encode all the possible states in the cover state graph takes $m! \over (m-n)!n!$, where $m$ is the number of cover locations and $n$ is the number of squad members.

We consider the scenario where all units are identically equipped with rifles. In this case, a squad member has only two options: 1. Move to a new cover location or 2. Provide suppressive fire while hiding. As far as we are concerned, each squad member is fundamentally the same.

Under this premises, we can leave out $p_i$ from equation 3.5 in the squad state representation:

$$S_t = \{c_{i_1}, c_{i_2} \ldots c_{i_n}\}, \quad \text{where} \ (c_{i_j} \in C) \quad (3.6)$$

The total number of states from our cover graph is now restricted to $m! \over (m-n)!n!$. If we have 10 cover locations and 4 squad members, the previous state representation will yield 5040 states compare to 210 states in the new state representation. In our situation, each state is
represented as a string of size \( n \), indicating cover locations from the set \( C \) selected by the squad in a lexicographical order.

A new graph \( G_1 \) is generated encoding the state of the squad. For each \( v \in V, G_1 = (V,E) \), \( v \) stands for the state of the whole squad. An edge \( e(i,j) \) in \( G_1 \) represents the state change between \( S_i \rightarrow S_j \). An edge only exists between the states where a single squad member moves from one cover location to an unoccupied cover location, provided there is a permissible edge \( e' \) in the cover graph \( G \) between those two cover locations.

\[
S_i \rightarrow S_j = \{c_{i_1}, c_{i_2}, ..., c_{i_n}\} \rightarrow \{c_{j_1}, c_{j_2}, ..., c_{j_n}\}(c_{i_1}, c_{i_2}, ..., c_{i_n} \in C) \quad (3.7)
\]

In equation 3.7, if \( S_i \) and \( S_j \) share the same cover \( c_k \), \(( c_k \in S_i, c_k \in S_j )\), the squad member at cover \( c_k \) remains at the same location when the state changes. In Figure 3.7(b), we have a cover state graph generated from the cover graph (in Figure 3.8) with only two squad members. This cover state graph is not complete since some edges such as \( (5,6) \rightarrow (6,7) \) is not shown in Figure 3.8 due to limited space. The entrance \( A \) and exit \( B \) (on the bottom of cover state graph), are treated as outside of the arena and allow multiple squad members to occupy this location.
Figure 3.8: Partial cover state graph from Figure 3.7(b)

The edge between (1,3) to (2,3) represents one squad member moving from cover location 1 → 2 while the other squad member remains at the same cover location 3 to provide suppression fire. The edge weight is then computed as the damage in equation 3.4, with $\vec{x}_1$ as location 1 and $\vec{x}_2$ as location 2. For the squad member at location 3, he/she will suppress the enemy that casts most damage onto the path between location 1 and 2 within his/her line of sight.
3.4 Optimal Squad Movement

To find the globally optimal squad tactic for the cover graph $G$, we simply find the shortest path on $G_1$ from the initial state to the goal state. Dijkstra’s algorithm can be used to find the shortest path in this situation. Yet, the performance of searching can usually be improved using A* search.

The global optimality of A* is ensured if the heuristic function $h(n)$ always underestimates the actual cost from the node $n$ to the goal node. In this case, graph $G_1$ may have edge weights equal to zero, if the static squad members can suppress all the enemies that potentially harm the moving squad member. Therefore, finding an admissible heuristic function is impossible on $G_1$ alone.

To combat this, we go back to the cover graph to speed up the shortest path search by over estimating the suppression fire. Consider the cost for each member to reach the exit in cover graph $G$. If there is no suppression fire from squad members, the cost of moving the whole team to the exit is always greater than or equal to the sum of the minimal cost for each squad member moving to the objective. Therefore, for the current state $S_i$:

$$h(S_i) = \sum_{k=1}^{n} \text{MinimalCostToExit}(c_{ik}, G), (\forall i, 1 \leq k \leq n, c_{ik} \in S_i) \quad (3.8)$$

The equation is formed into how to estimate minimal cost for one squad member to reach the exit. If the locations of a squad member are at least one step away from the exit B, we have:
If the exit $B$ is directly reachable from $c_l$, $w_{\text{min}}(c_k, B)$ stands for the minimum cost travelling through $\text{edge}(c_k, B)$ in $G$. The minimum edge weight is the edge weight excluding the $(n - 1)$ largest damages on it. This can be understood as the $(n - 1)$ enemies that cast largest damage on the edge are suppressed. We can maximize the amount of suppression fire provided by fellow static squad members. In other words, for an $n$ member squad team and $l$ enemies, when one squad member travels between cover $c_{ik}$ to $c_{jt}$, we only count the damage from the $l - (n - 1)$ enemies that have the least influence on the moving squad member.

$$w_{\text{min}}(c_{ik}, c_{jt}) = \sum_{x=1}^{l-(n-1)} \{\text{edge}_x(c_{ik}, c_{jt})\}$$

where $(\text{edge}_1(c_{ik}, c_{jt}) < \text{edge}_x(c_{ik}, c_{jt}) < \cdots < \text{edge}_l(c_{ik}, c_{jt}))$

For the minimum edge weight $W_{\text{min}}$ among all the edges in $G$, it can be written as:

$$W_{\text{min}} = \min\{w_{\text{min}}(c_{ik}, c_{jt}) \forall \text{edge}(c_{ik}, c_{jt}) \in G\}$$

This $h(S_i)$ function is guaranteed to be admissible in an A* algorithm with suppression fire. If the squad member is one step away to the exit, the cost of moving him/her to the exit can be estimated via maximum suppression fire. For the rest of the team, it takes at least two steps to make it to the exit. Each step is guaranteed to be greater than $W_{\text{min}}$. Assume $\text{Cost}(S_i)$ stands for the actual minimum cost for the squad team to get to the exit, we have $\text{Cost}(S_i) \geq h(S_i)$. 

\[ \text{MinimalCostToExit}(c_k, G) = \min\{w_{\text{min}}(c_k, B), 2 \times W_{\text{min}}\} \] (3.9)
However, in most of the cases, \( W_{\text{min}} = 0 \). For example, if we have one edge in graph \( G \) that only receives damage from 3 enemies. Given a squad size of 4 with all the static squad members suppressing enemies, then \( w_{\text{min}} \) of this edge is 0. This causes \( W_{\text{min}} \) to be 0. In this case, the A* search will degenerate into \textit{Dijkstra}.

Now instead of considering the steps from the current cover location to the exit, the squad member needs to take the shortest path to the exit. Each edge in the graph \( G \) is assigned with a new weight: \( w_{\text{min}} \). Therefore, the \( h(S_i) \) can be rewritten into:

\[
h(S_i) = \sum_{k=1}^{n} \text{ShortestPath}(c_{ik}, G), (\forall i, 1 \leq k \leq n, c_{ik} \in S_i) \tag{3.12}
\]

The path each squad member takes has a \( \text{Cost}(S_i) \) which is greater than or equal to the shortest path with maximum suppression, since not all \((n - 1)\) enemies can be suppressed. Therefore, \( \text{Cost}(S_i) \geq h(S_i) \). Also, \( \text{ShortestPath}(c_{ik}, G) \geq 2 \times W_{\text{min}} \).

Hence this heuristic provides a more accurate estimation. We can pre-compute the \( \text{cost}(S) \) using an all-pair shortest path algorithm on the small graph \( G \), then \( h(S) \) can be computed with constant time look up, which makes this heuristic function a better fit than the previous one.

### 3.5. Implementation Details

Figure 3.9 depicts our process for generating the optimal squad tactics. With the game scene as input, we compute the accessible area over the scene, within which a damage map
will be computed. Then based on the damage map, a set of cover locations are selected by equation 3.2. We then examine all the cover locations and build Cover Graph $G$. A vector of edge weights will be computed for $G$ as damages projected by each enemy’s line of sight. Some edges are discarded if they are too long.

Figure 3.9: Flow chart for our algorithm.

Instead of pre-computing the cover state graph $G_1$, we can compute the state change as we search. As search progresses, the estimated minimum node will be expanded. $g(S)$ stands
for the actual cost for changing state. It can be estimated by the sum of the vector of edge weights on the cover graph $G$ that is not suppressed by the static squad members. Since the choice for suppression fire is binary, a bipartite graph can be built between static squad members and enemies in Figure 3.10.

![Figure 3.10: White lines indicate the possible enemy suppression. The green line depicts squad member at cover 4 trying to move to cover 7.](image)

In Figure 3.10, we have the bipartite graph with each edge (white line) connecting squad members (orange dots) and enemies (cyan dots). This is a relative simple example with only two squad members’ presence. The squad member at cover location 3 has two choices of enemies to suppress (the other two enemies are blocked by covers). The edge $e(p_l, q_j)$ in the bipartite graph indicates that the squad member $p_l$ is able to suppress enemy $q_j$. The
edge weight of $e(p_i, q_j)$ measures the amount of damage enemy $q_j$ would inflict on the moving squad member.

Hungarian algorithm, also known as Kuhn-Munkres algorithm, is a combinatorial optimization algorithm that solves the assignment problem [34]. Here we use it to find the maximum weighted matching on the bipartite graph, for the optimal suppression strategy. Given the cost for $n$ workers and $m$ assignment, the algorithm works as follows:

1. Build $n \times m$ matrix $C$ as cost matrix. Each entry $C(i,j)$ represents the cost for worker $i$ to complete assignment $j$. Let $k = \min(n, m)$.
2. Find the smallest value in a row and subtract it from the rest elements in the row and repeat for all rows.
3. Similarly, find the smallest value in a column and subtract it from the rest elements in the column. Repeat for all columns.
4. Find the minimum amount of vertical and horizontal lines to cover all zeros in $C$.
   
   If the number of line equates $k$, we found the solution. If the number of lines is less than $k$, continue to step 5.
5. Determine the smallest entry that is not covered by any line. Subtract it from each uncovered rows, then add it back to each covered columns Then follow the step 4.

In Figure 3.10, if the left edge is selected, the cost for the state change $(3,4) \rightarrow (3,7)$ is the summation of the damage received from the rest of the enemies. Hence, the state change cost is computed as the summation of all enemies’ damage reduced by optimal suppression fire.
Heuristic function $h(s)$ is estimation based on the cover graph $G$. In our implementation, we pre-calculate the heuristic for all the nodes in the cover graph $G = (V,E)$. Then we construct a graph $G' = (V,E)$, which has the same vertices and edges as $G$, but has different edge weights, according to equation 3.12. The vector of edge weights in $G$ can be sorted so that the top $(n - 1)$ damages are not counted towards the edge weight in $G'$. Then we run a single source shortest path algorithm on $G'$ with the exit as the source, to obtain the minimum damage for a squad member to go from each cover location to the exit with maximum suppression fire. Then the $h(S)$ is the summation of all squad members’ minimum damage to the exit at the current state $S$.

3.6 Evaluation

We examine the scenario shown in Figure 3.11 to test our algorithm. The entrance and exit are marked as green door signs in the figure. Valid cover locations are represented as green grids in the area.

In this scenario, we placed nine objects in the scene. Six enemies are spread around the game scene and a 3-man squad team travels from the left to the right side (marked as a door sign). We consider the entrance and exit outside of the arena where the squad members are unable to provide suppression fire. The squad in total has 300 health points (HP), once the accumulated damage exceeds that in tactics, the tactics will fail.
The magenta color shows the amount of expected damage each grid receives. The green dots indicate a total of 22 valid cover locations where the squad members can hide and still be able to pin down enemies.

Figure 3.11: Set up for our test scenario. Enemies are marked with cyan color and objects are textured as wood.

Using our algorithm, the total damage taken is 91.18386 taking 13 steps. Figure 3.12 shows ten consecutive steps from step 2 to step 12 of how squad members move and suppress enemies for the optimal movement. To better visualize their movement, each squad member is marked with a different color: orange, blue and pink. In Figure 3.12(c), the pink squad member is behind the leftmost object, the orange squad member is at one of the cover locations on top. The blue squad member stays at the entrance ready to move.
Figure 3.12: 10 steps selected from the optimal squad tactics. Lines between the squad members and the enemies show the suppression strategy from static squad members from current step to next step. Notice how the damage map (shown in magenta) changes are different enemies pinned down with suppression fire.
Figure 3.13: Greedy approach to move the squad team. Figure (a)(b)(c) show consecutive steps. (d) to (e), (f) to (g) (h)to(j) are 3 sets of consecutive moves. It takes the wrong path and fails its mission.
The pink squad member pins down the enemy on top of the largest object. The orange squad member pins down the enemies at the top left corner. Both pinned down enemies are marked with in yellow. The background shows the damage map after pinning down these enemies. In comparison to Figure 3.11, the top left of the damage map is significantly darker, which means when the blue squad member moves close to the orange squad member, he/she will encounter significantly less damage. In Figure 3.12(d), the blue squad member successfully advanced to his/her new location.

In Figure 3.12(a) to (f), the pink squad member can be considered the ‘sniper’ of the team, since he remains at an unexposed cover location and pins down enemies at a distance to secure advancement of the rest of the team, who eventually land on good spots where a large group of enemies can be suppressed. Once the team arrives at the right location as in Figure 3.12(f), the “sniper” is able to safely move as shown in Figure 3.12(g) to (j).

In comparison, Figure 3.13 shows a greedy strategy for the squad. The greedy approach always chooses the next state with the least damage. Unfortunately, the squad failed in their mission. Figure 3.13(a) also starts from step 2. Figure 3.13(a) to (c) shows 3 consecutive states where the pink squad member goes down to explore, leaves the rest of the team behind until step 9, shown in Figure 3.13(d). Then the pink squad member pins down enemies at the bottom, allowing the orange squad member to unite with him. The same goes for Figure 3.13(f) to (g), where the blue squad starts to move towards the exit.

As shown in Figure 3.11, the cover locations at the bottom left corner has almost no benefit in connecting to the exit. The squad member who made it to the edge of the bottom left
corner will have to retract to the entrance in order to reach the exit. Not until step 74, in Figure 3.13(h) has the squad exhausted all the stats in the bottom left corner. Then in next state, Figure 3.13(i), the squad starts to move back on track.

In Figure 3.13(j), the damage received at this point exceeds the maximum HP. If we disregard the restriction of maximum HP, it takes 1943 steps to finally arrive at the exit, while the number of states in the cover state graph is 2073. This greedy approach will result in higher damage on the squad team since it only considers the local maximum. It fails to detect the safer path or dangerous location even one step ahead.

### 3.7 Conclusion

In order to find an optimal squad tactic, we model the squad member’s behavior of taking cover and suppressing enemies. After building a cover graph by examining the damage map of the scene, we look into the state representation of the squad, which is used to build the cover state graph. The problem is formulated into finding the shortest path on the cover state graph. Instead of doing a simple Dijkstra search on the cover state graph, we discussed two heuristics that help speed up the process.

Our algorithm allows the squad to move on a cover graph with a minimal total damage. The algorithm is deterministic and guarantees global optimality since we adopted an admissible heuristic. However, the computational cost makes it unsuitable for real time. Therefore, it can be served as a design/scripting guide for other AI algorithms. We compared our algorithm with the greedy approach, which indicated several flaws of the
greedy approach such as: dragging a squad member to an enemy surrounded area and leaving one squad detached from the rest of the team.
Chapter 4 : Large Open Map Construction Using Path Tiles

4.1 Introduction

In chapter 2, we proposed a way to procedurally place covers in a game level to help a designer achieve their desired game flow [2]. Consider the following example: we have an area $S$ and we wish to populate $S$ with enemies and covers. Assume the enemy’s line of sight oversees a surrounding area with radius $r_1$. This is also defined as the enemy’s effective radius. If the player is within $r_1$ and is visible to the enemy, he/she may be pinned down or killed. We space out the enemies so that they are at least $r_2$ away from each other. As for the covers, we assume their size is bounded by $2d$ for computation simplicity. To prevent overlapping, the covers will be placed at least $2d$ apart.

Given this, our goal is to find a set of covers and enemies in $S$ that optimize design goals, such as, a configuration of covers and enemies that ensures a player has increasing resistance when crossing $S$. 
In the chapter 2, given $m$ enemy locations, we used Monte Carlo search to sample cover configurations with $n$ covers to optimize the design goal. The number of all possible cover locations is infinite. If we use a set of precomputed $N$ Poisson points for cover locations, the search space for $n$ covers can be constrained to $\binom{N}{n}$.

In this chapter, we are also searching for the set of enemies. Ideally, we could have two Poisson disk samplings: one contains $N_c$ points for the covers with radius $2d$, the other contains $N_e$ points for enemies with radius $r_2$. Assume we have the set of enemy’s locations $E$ and the set of cover locations $C$. Since we do not allow covers and enemies overlapping, we have $\forall e_i \in C, c_j \in C, 0 \leq i \leq n, 0 \leq j \leq m, \|e_i - c_j\| \geq \frac{r_2}{2} + d$. This indicates the distribution of the enemy locations also affects the choices of cover locations, the initial search space is in the range of $[\binom{N_e}{m}\binom{N_c-m}{n}, \binom{N_e}{n}\binom{N_c}{m}]$.

If a single precomputed $N$ Poisson Point distribution is used as candidate locations for both enemies and covers, the search space can be reduced to $\binom{N}{m}\binom{N-m}{n}$. The radius $r$ of the $N$ Poisson points is required to be at least larger than $\max(2d, r_2, d + 0.5r_2)$, which guarantees no overlapping between enemies or covers.

For instance, given 100 different candidate locations in $S$, assume we want to populate 20 covers and 30 enemies in $S$. The number of all possible configurations is thus $\binom{100}{20}\binom{80}{30}$. If each configuration takes 1ms to evaluate, it would take $1.5 \times 10^{32}$ years to go through all possible configurations. To find an optimal one, AI search techniques could help, but in
an arena with an arbitrarily large number of candidate locations, finding an optimal configuration is NP.

To minimize the search space, we adopt the idea of tiling. Area $S$ can be divided into small grids. Assume the maximum number of candidate locations in each tile is 10, ($10 \ll N$). Given the desire for 3 covers and 5 enemies in a single tile, the number of all possible configurations is only 2520. If a finite set of tiles can be constructed to represent different design requirements, we will be able to synthesize an arbitrary arena using tiling. It is described that each tile can be computed independently. This allows us to pre-generate the set of tiles.

In this chapter, we will talk about how to create a tile set that contains different requirements as well as their application in tiling. We will examine and compare different tilings with different tile sets. Other benefits of using tiling will also be discussed, such as giving the artists/designers the freedom to specify the flow of the game. Tiling also offers the capability of catering a large variety of gameplay.

### 4.2 Related Work

The mixed initiative method is also a popular technique that allows designer’s input to aid content generation. As in chapter 2, we have the designer to provide a curve for the desired flow of the level. Liapis et.al [35] provided the designer with a map editor to place resources in the beginning. During the process, the tool will test the playability of the
design. This creates an interactive experience for the designer. In our tiling algorithm, the
designer can supply the sketch of a major path for the tiling.

Maung et.al [36] proposed a set of maze tiles that have the semantic meaning for navigation.
Each tile has connections between different edges to indicate a pathway between these
edges. With a properly tiling algorithm, they are able to obtain a maze. We adopt this idea
to allow a designer to use the set of maze tile for sketching.

### 4.2.2 Wang Tiling

Named after Hao Wang [37] [5], Wang tiles are squares with each edge *colored*. The
matching edges in a Wang tiling shares the same *color* (The color here is any N-
dimensional vector). When using Wang tiling, tiles cannot be reflected or rotated. In 1996,
Culik II [7] proved that a set of 13 Wang tiles is able to tile an infinite plane non-
periodically.

The simplicity and aperiodicity makes Wang tiling widely used in various fields. Cohen et
al [38] adopts Wang Tiling into 2D texture synthesis, 2D Poisson distribution and 3D
geometry. They generated each Wang tile based on four sub-images from the source image.
Then with the set of Wang tiles, they proposed a stochastic tiling algorithm to synthesize
texture that is visually consistent with the source image.

In Poisson disc sampling, a point in the corner of a Wang tile may have impact on the other
three adjacent tiles. This problem is referred as the corner problem. A simple fix is to
enforce each corner (and hence the ends of each edge color) to a small set of values,
typically one. Lagae and Dutré introduced Wang corner tiles to avoid this problem [39]. In our research, we construct a set of path tiles whose edge equate the color in Wang tile. With this concept, we are able to create an infinite non-repetitive game level using path tiles.

4.3 Tile Design

We define path tiles as a set of Wang tiles with covers and enemies for a player to go through. In chapter 2, we encode the amount of damage in the tile by the flux of incoming damage. The damage function can be defined as follows.

4.3.1 Damage Function

The amount of damage a player receives at any location in the tile is the combination of all visible enemies. In this function, $\mathbf{x}$ is the location of the player, $C$ is the set of covers and $E$ represents the set of $m$ enemies, whose has impact of $w_e$ towards the player.

$$damage(\mathbf{x}, C, E) = \sum_{e=1}^{m} w_e * visible(\mathbf{x}, e, C)$$  \hspace{1cm} (4.1)

The damage affects the player’s choice of path as well as game experience. As shown in Chapter 2, this allows higher level semantics to be expressed through specifying the desired damage along the player’s path. To provide the player with a consistent game experience in a large open map, the damage should be the same when the player is crossing the edge between two adjacent tiles in the tiling.
4.3.2 Edge Condition

Figure 4.1: (a) Tile $T_1$. (b) Tile $T_2$. (c) Example of placement of two tiles.

Consider two tiles, tile $T_1$ shown in Figure 4.1(a) and tile $T_2$ in (b). Each of those tiles is treated as a square scene in Chapter 2. Enemies for both tiles are marked with magenta dots and the damage map is portrayed as the brighter color indicates the higher damage. Note that the bright area around the enemy is dictated by his/her line of sight $r_1$. Assume the player is at the right border of $T_1$, on the red line. With the information only from the left
tile, the player will presume he/she is at a safe location. However, if we place two tiles together, as shown in Figure 4.1(c), when crossing the edge, the player will find himself/herself in a dangerous location exposed to two enemies.

To avoid this situation, we need to guarantee the damage is the same along the edge for all adjacent tiles. Let $damage_T(x, C_T, E_T)$ represents the damage the player receives at location $x$ in tile $T$, with cover set $C_T$ and enemy set $E_T$. Then we have:

$$\forall x \in edge_{T_1, T_2}, \|damage_{T_1}(x, C_{T_1}, E_{T_1}) - damage_{T_2}(x, C_{T_2}, E_{T_2})\| < \varepsilon \quad (4.2)$$

Here $\varepsilon$ stands for a small threshold. We can ensure consistency along the edge by requiring all enemies’ locations to be reflected across the edge, if they are close to the edge (Figure 4.2). Thus the Wang color in this case are equal.

![Figure 4.2 Enemy locations reflected to the left tile.](image)
As defined in chapter 2, the optimal path is the minimal damage path, given entry and exit points. Let the entry point be \( x_{in} \), the exit point be \( x_{out} \), the cover set be \( C \) and the enemy set be \( E \).

\[
OptimalPath(x_{out}, x_{in}, C, E) = \arg\min_{\text{path}} \left\{ \int_{x_{out}}^{x_{in}} damage(path(x), C, E) \, dx , \text{path} \right\}
\] (4.3)

Given an individual cover tile, we can compute its optimal path given a pair of entry and exit points. As depicted in Figure 4.1(c), the red line is the pre-computed optimal path of the left tile (not taking into account the reflected enemy). Compare to the Figure 4.1(d), the optimal path changes due to incurred higher damage from reflected enemies. It defeats the purpose if we have to compute the optimal path every time its neighbor changes.

To tackle this problem, the neighboring tile’s influence needs to be considered as a part of the Wang tile construction.

### 4.3.3 Cover Edge

Enemies’ effective radius \( r_1 \) indicates an enemy can have influence on a neighboring tile at most distance \( r_1 \) away. We define a cover edge as a rectangular strip with width of \( 2r_1 \) which contains a set of covers and enemies. As shown in Figure 4.3(a), the cover edge is depicted as the white rectangular overlay for both tiles. Enemies located beyond the right side of the cover edge have no influence on the left tile, while the enemies within the cover edge impacts both tiles.
Figure 4.3: (a) White strip indicates the cover edge. (b) Red line marks the solution path for the left tile given the tiling entrance.

In this way, if two adjacent tiles in the tiling share the same cover edge, these two tiles are consistent across the adjacent edge. As shown in Figure 4.3(b), in this simple 1x2 tiling, the left tile $T_1$’s right cover edge shares the same covers and enemies as the right tile $T_2$’s left cover edge. Note that, the damage function along the adjacent edge for each tile is calculated including the cover edge. In other words, for equation 4.1, $C_{T_1} = C_{T_1} \cup C_{coverEdge}$, $E_{T_1} = E_{T_1} \cup E_{coverEdge}$. This way we are able to guarantee the consistency of the damage function along the adjacent edges. This is a basic Wang tile matching.
Establishing a fixed set of cover edges before the tile construction ensures the damage at any location in an individual tile remains the same when tiled. In this case, cover edge equates to the edge color in Wang tile. As defined in Chapter 2, the optimal path for an individual tile is dependent on being given the entrance and exit of the tile. If we allow the player to exit anywhere along the edge, and let the player to go to the exit with minimal damage, the player may take the unwanted shortcut and exit the tiling too early. In the next section, this problem will be examined.

### 4.3.4 Path through Tiling

Returning again to Figure 4.3(b), the tiling’s entrance $x_{in}$ locates at the left side of the left tile $T_1$ and the exit $x_{out}$ at the right side of the right tile $T_2$. When the player enters $T_1$, intuitively he/she will search for the easiest way out. The player will head towards $x'_{out}$ at the bottom of $T_1$. Since $x'_{in}$ is on the adjacent edge, the path does not lead to the path in $T_2$. Therefore, even though the damage across tiles is consistent, it may not be guaranteed to be playable. To resolve this, we need to look at the possible paths the player takes for each tile.

In equation 4.3, the optimal path is the lowest damage path for a pair of entry and exit locations. Yet, in an open area, it is not clear where the entry and exit locations of each tile are. Once the player enters a tile at $x_{in}$, intuitively he/she will seek the exit of the tile which incurs minimum damage. We define such path as the solution path.
\[ \text{SolutionPath}(C, E, x_{in}) = \text{ArgMin} \left\{ \min \left\{ \int_{x_0}^{x_{in}} \text{damage}(\text{path}(x), C, E) \, dx \right\}, \text{path} \right\} \forall x_o \in \text{tile edges} \] (4.4)

The solution path of \( x_{in} \) is the minimum damage path given the entrance \( x_{in} \). In relation to the optimal path, the solution path can be understood as the minimal damage path of all the optimal paths sharing the same entrance. Now if we examine Figure 4.3(b) again, the solution path of \( T_1 \) (marked as a red line) goes to \( x'_{out} \) at the bottom of the tile. It never makes it to the \( x_{in} \) of \( T_2 \).

In some games, the designer may want discourage the player from taking a short cut. In Chapter 2, we placed walls around the scene to block the player from exiting elsewhere. Obstacles such as buildings, trees, mountains etc. also used in the game to guide the user. Sometimes the designer wants to hide the edge of the world or simply not spoil the next level. By forcing the player onto a certain path, the designer is also able to setup the storyline and even a boss fight for the player.

Rather than putting physical objects such as a wall or mountain to block the path, a large number of enemies may be placed there to avert the player. This way, the player can peek through the enemies and still have an illusion that the short cut is explorable. Then the player can take the alternative route to explore the area. To satisfy this feature, our approach is to construct a target zone around the desired entrance and exit of a tile. The designer may want to line up the desired exit to the neighboring tile’s desired entrance, which can be achieved ensuring the matching between target zones for the adjacent tiles.
4.3.4.1 Target Zones

Figure 4.4: (a) Solution path are represented as red lines. (b) Alternative solution path that is also valid. Target zones are depicted as blue segment along the edge.

As we limit the entrance and the exit to a single point, those points need to be connected to ensure playability. This could be enforced, however, it will severely limit our design space and may lead to repetitive tiles. As shown in Figure 4.4(a), the exit point of the left tile does not connect to the entry point of the right tile. This causes the tiling to be disqualified.
However, the blue segment between the two entry/exit points along the adjacent edge in Figure 4.4(a) has almost zero expected damage. This indicates if the player follows the solution path of the left tile to the adjacent edge, he/she can travel along this segment with minimal damage. In this case, the player is able to continue along the solution path in the right tile, producing a consistent tiling.

We define a target zone as a segment along the edge, $Z \subseteq \text{edge}$, such that points inside the target zone are valid entry/exit locations. Enemies are not allowed within the target zone. When the player travels within the target zone, he/she endures minimal damage. For the edges without a target zone, we currently desire those edges to be unpassable. An unpassable edge implies significant damage for the player to go across.

4.3.4.2 Tile Path

There are multiple solution paths which may go through the target zone. As shown in Figure 4.4(b), an alternative solution path also lands in the target zone. Since we guarantee that travelling within the target zone is close to free, we are able to represent the cost of travelling from one target zone $Z_1$ to another target zone $Z_2$ across a tile with one solution path. We define that solution path as the tile path. Let $x_1$ and $x_2$ be the points on the target zone $Z_1$ and $Z_2$ (Figure 4.5).

$$TilePath(C,E,Z_1,Z_2) = \underset{\text{path}}{\operatorname{ArgMin}} \left\{ \min \left\{ \int_{x_1}^{x_2} \text{damage}(\text{path}(x), C, E)dx \right\}, \forall x_1 \in Z_1, x_2 \in Z_2 \right\}$$ (4.5)
Figure 4.5: A tile with target zone $Z_1$ on the left edge and $Z_2$ on the right edge. $x_1$ and $x_2$ are two random locations on each target zone. Point $y$ is located outside the target zones.

Now we have a tile path to evaluate the expected damage in travelling from $Z_1$ to $Z_2$. Zone damage is defined as the accumulated damage along the tile path for target zones $Z_1$ and $Z_2$.

$$
ZoneDamage(C, E, Z_1, Z_2) = \int_{TilePath(C, E, Z_1, Z_2)} damage(x, C, E)dx
$$

(4.6)

Zone damage indicates the minimum damage to cross the tile. To ensure the player goes through the target zones, we guarantee that the zone damage is strictly less than travelling outside the target zone. In Figure 4.5, point $y$ is a random point located outside the target zone, we have:
In equation 4.7, $\int_{x_2}^{y} damage(path(x), C, E)dx$ determines the minimum damage for the player to go from any point $x_2$ in target zone $Z_1$ to any point $y$ in the non-target zone. By enforcing this rule, the player will endure less damage going into the target zone as compared to going to anywhere in the non-target zone. By thresholding the cost difference between travelling from the target zone to outside the target zone and travelling within target zones, the player will be punished outside the target zone. This prevents the player from veering off the desired path and eliminates unwanted short cuts in the tiling.

With the definitions above, we conclude the following matching conditions for cover edge:

1. The matching cover edges use exactly the same set of covers and enemies.
2. The intersection between the matching cover edges’ target zones is non-empty.

We call these path tiles. When used in a tiling as Wang tiles, it provides the consistent optimal path tiling we desire. The next section discusses how to construct a tiling using these path tiles.
4.4 Tiling

Our goal is to use a set of path tiles to construct a tiling on a large open map with a width \( w \) and a height \( h \). The tiling should contain a main path from the entry \( X_{in} \) to the exit \( X_{out} \).

To simplify the representation of the path tile, a path tile can be semantically considered as a waypoint tile as it connects different edges with their target zones. We call them navigational tiles.

4.4.1 Navigation Tiles

Figure 4.6: The list of 6 navigational tiles. The connection between edges are marked as black lines.

In Figure 4.6(a), the black line indicates the player is only allowed to travel between top and bottom edges. All path tiles with target zones located only on the top and bottom edges and overlaps with midpoint can be represented by this navigation tile. Therefore, we may have multiple path tiles share the same navigation tile. For example, in Figure 4.6(e), the navigation tile connects the bottom edge to the right edge. Two different path tiles for this are shown in Figure 4.11(a) and (b). The navigation tiles can easily be used to sketch out the path by a designer.
In this chapter, we only focus on linear gameplay. This implies there is only one major path for the tiling. We enforce this property by choosing the path tiles with only two edges containing target zones. Figure 4.6 shows the complete set of six semantic navigational tiles, each containing one path. Note that branching would be easily supported by changing the number of edges contains target zones, but this is not addressed for clarity and easier design goal specification.

4.4.2 Tiling Algorithm

With the set of navigation tiles, the tiling path can be sketched out. We are able to generate a tiling path from a designer specified path. These navigation tiles are then substituted with desired path tiles to obtain the final tiling. In Figure 4.7, we only use the path tiles in Figure 4.12 to replace the navigation tiles.

Figure 4.7 (a) shows a designer sketched curve over a tiling grid, which is then fitted out by the navigation tiles (Figure 4.7(b)). In our final tiling, we substitute the navigation tiles with corresponding path tiles (in Figure 4.7(c)). The optimal path is displayed as a bright yellow curve. Figure 4.7 (d) (e) (f) is another example of the designer’s sketched path. Note that in this case, the solution path is significantly shorter, which makes the rest of the space open for exploring. In the level design aspect, the solution path in Figure 4.7 (f) can be treated as the path for the main quest. The designer may place some resources in the open area to encourage the player to explore. This can also be supported by extending the path tiles with branches.
Figure 4.7: (a) The blue line represents the path specified by designer. (b) The fitted out designer path with navigation tiles. (c) The complete tiling with path tiles. The yellow line depicts the tile paths. (d) Another designer’s sketch with shorter path. (e) Sketched path in (d) are replaced with navigation tiles. (f) The complete tiling of (d). The cover and enemy locations are not shown for clarity.
4.5 Path Tile Generation

Figure 4.8: The process of tile creation.

In Figure 4.8, there are seven steps to generate the path tile, which can be divided into two categories: generating cover edges and searching for the best path tile.
4.5.1 Generating Cover Edge

In section 4.3, a cover edge is defined as a rectangular strip with width of $2r_1$ where $r_1$ is an enemy’s effective radius. We use a dart throwing algorithm to generate a Poisson point distribution with radius $r$ on the rectangular strip as the candidate locations.

With the candidate locations on the cover edge, we are able to differentiate edge colors by assigning different candidate locations to covers and enemies. We do not allow any enemies in the target zone, the target zone may be specified by the designer. For covers, if we allow them to be placed in the target zone, we need to guarantee they do not block the target zone.

Figure 4.9: (a) Cover edges are marked as red rectangular strips. The vertical cover edge overlaps the horizontal one. The enemies’ effective radius is $r$. (b) The corner is marked as blue octagonal shape.
Since the cover edge equates to an edge color in Wang tiles, it inherently has the corner problem. As shown in Figure 4.9(a), if a Poisson point is placed in the overlapped area of the cover edges (marked as dark red), the point will influence both cover edges. It has impact on the three neighboring tiles as well.

To solve this, we adopted the method from [39]. With pre-generate corner colors as shown in Figure 4.9(b), the corner color is represented by a blue octagonal area. It spaces out the horizontal and vertical cover edge a distance $r$ away to prevent mutual influences. Any enemies within the new vertical cover edge will have no impact on the newly defined horizontal cover edge.

![Figure 4.10: Vertical cover edges. The covers are marked by orange squares while the enemies are marked as magenta dots. (a) Vertical cover edge without target zone. (b) Vertical cover edge with target zone.](image)

Here we use one corner color for all edges. We have two vertical cover edges shown in Figure 4.10. In Figure 4.10(a), the cover edge is depicted with no target zone. The blue
color indicates a high damage area in the cover edge. This will discourage the player from passing through. While in Figure 4.10(b), the target zone is situated between two covers. The darker blue area shows a relatively safe zone in the cover edge, which allows the player to pass through.

We generated multiple cover edges for each direction, horizontally and vertically. Knowing candidate locations for all edges, the interior candidate locations of the tile can be computed using the same dart throwing algorithm. However, this does not guarantee a safe path connecting any target zones. The next important part in our algorithm is thus searching for the best set of path tiles.

4.5.2 Searching for the Best Path Tiles

To have a complete set of path tiles, we need to have a path tile $P_i$ correspond to each navigation tile $N_i$. This is done by specifying the cover edge colors then searching for the valid path tiles. After we weed out the invalid path tiles, the best path tiles will be selected using metric functions.

4.5.2.1 Validity

In section 4.3.4.2, the zone damage is guaranteed to be smaller than the damage travelling through a non-target zone. Similar to chapter 2, we are able to discretize the tile and compute a damage map.
In equation 4.5 and 4.6, to compute zone damage, we need to sample through all of the $N$ points within target zones. Using equation 4.4, we search for the solution paths for those points. In our implementation, for each of the $N$ points, we use it as the starting point to run Dijkstra’s algorithm on the damage map. Then the solution path is obtained by finding the minimal accumulated damage for all the shortest paths with this starting point. After this process, $N$ solution paths are collected corresponding to the $N$ starting points. If the minimum damage solution path exists outside the target zone, then the configuration is invalid. The tile is then rejected and the search continues. If the minimum damage solution paths lands inside the target zone (starting from one target zone and ends in another target zone), then the accumulated damage on that solution path is our zone damage. The tile is considered as a valid tile. Valid tiles are further processed as discussed below. Note that by searching single source shortest path for all starting point is equivalent of doing pair-wise shortest path for all points in the target zones, if we only care about the accumulated damage between target zones. The zone damage is the minimal damage between all possible pairs from one target zone to another.

To compute the damage from the target zone to non-target zone, we just consider the paths that land outside the target zone. The minimum accumulated damage of those paths is the damage from the target zone to the non-target zone. Then we compare this and the zone damage. If the zone damage is significantly less than the minimum damage from the target zone to the non-target zone, the player is very likely to stay on the solution path. The difference between this and the zone damage measures how easily the player may deviate from the tile path.
4.5.2.2 Metrics

Similar with chapter 2, metrics are used to evaluate cover configurations. We currently have the following metrics on the optimal path: 1) largest standard deviation of damage on the path; 2) path length; 3) total damage on the path. We use the metrics to evaluate path tiles.

Just like in chapter 2, we can have a designer specify the flow of the game, using a damage curve for the tile path. Based on the accumulated damage for path tiles, we may have multiple path tiles of different difficulty levels that correspond to the same navigation tile. In this case, path tile metadata with different difficulty can be generated to fit out the desired flow. For example, design goals such as increasing resistance through tile paths may be achieved by placing low damage/easy tiles in the beginning of the path, followed by medium damage tiles and ending with high damage/difficult tiles. Depending on the design goal, we may also generate different path tile metadata, such as a transition path tile. Such tiles may serve as a transition from low damage to high damage tile.

Figure 4.11 shows the two path tiles with their damage curves. Notice that they use the same set of points, but different choices are made as to whether the point is a cover location, an enemy or neither. Both Figure 4.11 (a) and (b) are generated with same cover edges. In Figure 4.11(c), the blue line indicates the damage curve for the path tile generated with the largest standard deviation metric shown in (a). For Figure 4.11(b), its damage curve is represented by the red line. The x axis in Figure 4.11(c) represents the tile path length and y axis displays the damage received along the tile path.
4.6 Results

We implemented our algorithm to generate three sets of path tiles using the same set of cover edges with two vertical colors and two horizontal colors. The three sets of path tiles are selected optimizing for the largest standard deviation metric (Figure 4.12), the longest path length metric (Figure 4.13) and the minimum damage metric (Figure 4.14). For each tile set, we need at least 6 tiles to represent all navigation tiles in Figure 4.6. Depending on
the parameters, the generation tile process may exhaust all possible configuration to find one valid solution, which may take hours. However, the tile set only need to be computed once before tiling.

We will discuss two use cases in tiling, the first one only uses tiles from Figure 4.12. and the second one uses all the tiles to achieve a design goal. Tile sets will be discussed in the following section.

4.6.1 Tile Set

As shown in Figure 4.12, the six tiles generated optimizing the largest standard deviation metric correspond to the semantic navigation tiles in Figure 4.6. Take Figure 4.12 (d) for instance, the two target zone areas (at the left edge and the bottom edge) present low damage with darker blue while the middle part of the tile shows higher damage, providing some indecision and stress. This is due to the large standard deviation along the tile path.

Note some tiles also show interesting branching features. In Figure 4.12 (b), the path tile consists of target zones on the left and top edges. When the player is at the middle of the tile, he/she is facing two potential ways to exit the tile. Both ways provide shelter along the path, but the one heading to the bottom will clearly lead the player to a dangerous area. The player will have to backtrack before he/she dies.
Figure 4.12: The list of path tiles generated with largest standard deviation metric. Enemies are marked with magenta dots; covers are marked as orange squares. Each tile is colored with the damage. The brighter the color is, the higher the expected damage of the area. Each tile corresponds to a navigation tile in Figure 4.6.
Figure 4.13: Path tile set generated with the longest path length metric.
Figure 4.14: The path tile set generated with the minimum damage metric.
Figure 4.15: (a) The tiling generated with semantic navigational tiles. (b) The complete tiling with the path tile set generated with largest standard deviation metric in Figure 4.11.
Figure 4.13 shows a path tile set generated with longest path length optimization. Note that in Figure 4.13(c), by placing more shelters towards the bottom of the tile, the tile path is guided to have a more circuitous route compared to Figure 4.12 (c). These tiles are generated with the same cover edges, but provide us with more turns and longer gameplay.

Figure 4.14 depicts the path tile set generated optimizing for the minimum total damage along the path. Note that almost all the candidate locations are assigned to covers. This configuration offers the maximum amount of shelter for the player. This set of tiles can provide the player with a more peaceful gameplay, enriching the variety of the overall game experience.

### 4.6.2 Tiling with Single Set

Our first use case of tiling only applies the tile set generated with largest standard deviation metric in Figure 4.12. With the tiling of semantic navigational tiles generated by the designer sketched path in Figure 4.15(a), the complete tiling in (b) is created by substituting the semantic navigational tiles with corresponding path tiles in Figure 4.12. Note that there are two empty tiles in Figure 4.15(a). The two empty tiles can then be replaced with the tiles surrounded by unpassable cover edges. In this case, any path tile with the corresponding unpassable edges will work. In Figure 4.15(b), a random tile of that kind is chosen to fill in the empty spaces. The unpassable cover edge prevents the player from entering the tile.
Figure 4.16: (a) Tiling with semantic navigational tiles. (b) The complete tiling using all three tile sets. Cameras (used in Figure 4.15) are marked by a logo and its cone of vision is marked in green.
4.6.3 Tiling with Three Tile Sets

For this use case, three tile sets are used to create a tiling with richer content. The tiling is arbitrarily set to have 45% of the tiles with largest standard deviation, 40% of the tiles with longest path length and 15% of the tiles with minimum damage. We also require the tiles with minimum damage to be spaced out such that it serves as a buffing area for the player to relax.

As shown in Figure 4.16, we have a snaking path with the navigational tiles. Then with our algorithm, path tiles are selected based on the proportion of the design requirement. There are 5 empty tiles in the navigational tiling. To avoid visual repetition, we select different filler tiles with four unpassable cover edges. In the future, these could be replaced with dead-ends connected to a branching tile.

We marked three camera locations on the tiling in Figure 4.16(b) to display the 3D first person view of the scene (Figure 4.17). In Figure 4.17, zombies are enemies and giant barrels are covers. The camera locations are marked by a camera logo in Figure 4.16(b) with cone of vision colored in green. In Figure 4.17(a), this shot corresponds to the leftmost camera location. There is a clear pathway for the player to advance and the zombies are relatively further away. This creates a more relaxing game experience. In Figure 4.17 (b), it depicts a safer area in-between the barrels. However, the zombies around increase the risk for the player to dashing to a safer area. This brings a more intense gameplay. For Figure 4.17(c), there are two possible paths for the player to choose, one on the left and one on the right. If the player choses the one on the right, it will lead the player into a high
damage area. In that case, the player will have to backtrack to the branching point then select another path. This creates a short dead end, which adds a variety into the gameplay.

4.7 Conclusion

An algorithm for generating a path tile set and a tiling scheme that captures designer’s input are proposed for creating a large open map for a stealthy action game. Path tile is a form of Wang tile and shares the same properties. For a valid Wang tiling, a definition of matching conditions for a path tile’s edges comprises enforcing matching both cover edges and target zones. By constraining the entry and exit locations within the target zone, a tile path is determined based on the damage flux of the configuration of the tile. Each cover and enemy location is assigned with a precomputed Poisson point distribution. Therefore, it allows us to enumerate all possible configurations of the interior tile to examine the damage curve for their tile paths. Just as Chapter 2, three metrics are utilized to analyze the configuration by its tile path and select optimal PCG content.

For the tiling process, our algorithm provides a tiling with semantic navigational tiles that takes the input of the major path sketched by the designers. The designer goal is able to further specify the choice of path tiles in substituting the semantic navigational tiles. We demonstrate the path tile sets generated with largest standard deviation metric, longest path length metric and minimum damage metric. The interchangeable and tileable path tile sets allow for better control of game flow. With the precomputed tile sets, the tiling can be completed efficiently.
Figure 4.17: (a) (b) (c) Screen shots from a scene created with the tiling in figure 14(b). Screen shot in (a) is taken in left most camera location of figure 14(b). (b) is taken by the camera in the middle and (c) is taken at the right most location. (next page)
Figure 4.17 Continued.
Chapter 5: Shape Up! Perception based Body Shape Variation for Data-driven Crowds

5.1 Introduction

Simulation of crowds is needed for many applications, which tend to fall into two main categories: those that need as accurate a representation of a population as possible, and those for which visual realism and plausibility are the most important criteria. For example, for games and movies, the most important criterion is usually that the crowd is perceived to be realistic and to behave plausibly, whereas when designing a city, stadium or theme park, the planners will need to be confident that the movements, behaviors and body sizes of the agents are representative of the real humans who will actually populate the future environment. However, for many planning applications a realistic visual representation of the crowd is also important as it helps to better understand the real-world data being visualized; and often for games and movies, the more representative the virtual crowd is of a real population, the more realistic and plausible the results will be. Therefore, a data-driven approach that takes real-world demographics into account will be useful in both cases.
One of the most important factors for perceived realism of crowds is the level of variety in the 3D models used to visualize the virtual humans present. The perception of varied human appearance and motion have previously been studied in this context [8] [9], but to date, body shape variation has received less attention. However, two of the most functionally relevant and visually salient features of any crowd are the shapes and sizes of the people in it, and their relative distributions: i.e., under normal circumstances most people will have shapes close to the median of the population, and there will be decreasing numbers of more atypical bodies present.

For the majority of practical applications, it is not possible to individually model each member of a crowd, so a fixed budget of 3D models is used and replicated to generate a large crowd. Therefore, strategies are needed for choosing the optimal set of characters and distributing them realistically, while minimizing the risk of perceiving visual clones.

Approaches to varying body shapes for crowd simulation have been presented, and our work is very complementary to such approaches. For example, Thalmann and Musse [40] generate models with different body types that may be distributed according to any available statistics. However, the resulting models in such systems have tended to be more stylistic representations rather than physically accurate (Figure 5.1), and the perception of body shape differences or motion variation has not been considered. In the field of psychology, body perception studies have mainly been focused on gender and attractiveness (see [41] for a recent overview) and not on the distinctiveness of different body shapes. To our knowledge, this is the first study to perceptually evaluate the effects
of body shape on distinctiveness and the first to use body shapes derived from real data to explore attractiveness.

Figure 5.1: Pictures of crowds. (a) Crowds portrayed in Hitman Absolution. (b) Real crowd in New York City subway. The crowd in game is more stylistic (too close to the average) than the real crowd.
In this section, we present a data-driven approach to creating a crowd with representative and varied body shapes. We base our model generation and distribution strategies on body measurement and demographic data from the Civilian American and European Surface Anthropometry Resource (CAESAR) anthropometric database. From preliminary studies and observation, it was obvious that the most salient body shape differences are due to the height and girth (i.e., waist circumference) of the captured actors. However, closer to the median, other types of body shape differences are increasingly salient. As there are more individuals in this group than any other, more replications of each template model will be needed. We were interested in the following question and conducted two perceptual studies to explore them: What are the most salient factors that affect the perception of body shape distinctiveness for bodies close to the median height and girth? We found that the most salient body differences are in size and upper-lower body ratios. Based on our results we strategically select of a number of models which, with clothing variation, can be used to optimize crowd variety with a limited budget of 3D models.

5.2 Related Work

McDonnell et al. [42] [8] have studied the effects of different types of appearance and motion variation on the perceptibility of visual clones in crowds. Pražák et al. [9] demonstrated how only three different human motions replicated evenly through a crowd produces enough variety comparing to having different human motions across all crowd. Hoyet et al. [43] evaluated the distinctiveness and attractiveness of different types of human motion and found that average motions were the least distinctive and most attractive,
consistent with findings in face perception [44]. Johnson and Tassinary [45] [46] studied the effects of both shape and motion on the perception of sex, gender, and attractiveness. Using mainly stylized silhouettes of human body shapes with varying waist to hip ratios, from exaggerated hourglass to tubular, they found that both shape (especially waist to hip ratio for women), and motion information contributed to participants’ judgments of attractiveness. In particular, the waist-to-hip ratio (WHR) and hip sway are important for sex categorization and female attractiveness, while shoulders and their sway are most salient in men. Wellerdiek et al. [47] recently explored how body shapes and postures affect perceived strength and power of male characters. See [41] for a thorough overview of recent research in body perception. Apart from sex categorization, we are not aware of any studies where the distinctiveness of different body shapes has been explored.

Realistic body shape generation usually starts from real data, e.g. body scans [48] and measurements [49]. To generate a variety of body shapes, a body shape manifold or statistic model is typically learned from shape data [50] [51] [52] [53] [54] [55] [56]. Many assume a low dimensional manifold and dimensionality reduction techniques such as Principle Component Analysis (PCA) have been used to learn such representations. Once the manifold is learned, controlled body generation can be achieved by specifying part of the parameters and inferring the remainder. The parameter specification can be either done through sampling or user input [57] [54]. We use a commercial system (Daz3D) to generate our crowd characters, but our results would be applicable to models generated by any of these methods. With respect to crowd generation, our work is complementary to that of Thalmann and Musse [40], who generate models with different somatotypes (endomorph,
mesomorph and ectomorph). These mainly vary based on the distribution of body fat, muscle and WHR. Then, these body types may be distributed according to any available statistics. Our study could be used to improve the realism and representativeness of such a system.

5.3 Model Selection and Creation

To generate a set of 3D virtual human models for our data-driven crowd system, we need to select a representative sample of real body shape measurements and then create a realistic visual representation for each. Using a set of templates to be repeated throughout the crowd is efficient in terms of both artist time and computational resources, as hardware instancing can be used to replicate selected models.

![Figure 5.2: The CAESAR database distribution of body heights and girths, with red dots showing the samples we picked to generate our template models; the red cluster in the median group shows the samples used to generate our experimental stimuli. The total number of samples within each quadrant is shown at its top left corner. We use body measurement data sourced from the CAESAR project. The CAESAR data is based on stratified sampling of age, ethnicity and gender [10]. It contains demographic data,](image-url)
3D range scan and measurements for 2391 US residents (52.9% female, 47.1% male, aged 18-65, with varied ethnicity). Height and girth (waist circumference) are used to pick reference samples, as we observe that these are by far the most salient features that differentiate people from each other, especially the more they deviate from the median. Then, we use 29 measurements for each sample (See Table 5.1) to direct the creation of each template 3D human model. To represent the variation of the height and girth distribution as shown in, we divide the height into 5 groups and girth into 7 groups respectively. We combine some of the outlier quadrants with small numbers of samples. Then 3 samples are drawn randomly from each quadrant at least $r$ distance apart to avoid resemblance. After careful examination, one sample is kept to represent each bucket and is used as templates for generating the crowd. Due to the large sample count around the median (row 3, column 2), more distinctive samples are needed to maximize variety. As shown in the top image of Figure 5.5, all the red dots in non-median quadrant are visualized as 3D meshes with fitted clothes.

We used Daz3D to create realistic 3D models as it provides useful tools and morphs for creating body shapes, although any model generation tool with similar properties could be used. A set of mesh deformers $P$ is used to morph the template mesh $T$ to the desired measurement $M_{d}$. Thus we can generate models with measurements that closely match the CAESAR samples (see Figure 5.3).
Table 5.1: The 27 CAESAR body measurements used to generate our 3D models

<table>
<thead>
<tr>
<th>Measurement</th>
<th>Explanation</th>
</tr>
</thead>
<tbody>
<tr>
<td>Acromial Height, Sitting</td>
<td>Vertical difference from the sitting surface to the left acromion (shoulder)</td>
</tr>
<tr>
<td>Ankle Circumference</td>
<td></td>
</tr>
<tr>
<td>Arm Length (Shoulder to Wrist)</td>
<td></td>
</tr>
<tr>
<td>Hand Length</td>
<td>Distance from tip of middle finger to the end of the palm</td>
</tr>
<tr>
<td>Head Circumference</td>
<td>Circumference of the hand across knuckles of the index and little fingers</td>
</tr>
<tr>
<td>Hip Circumference, Maximum</td>
<td>Maximum hip circumference measured parallel to the standing surface</td>
</tr>
<tr>
<td>Hip Circumference, Maximum, Height</td>
<td>Vertical distance from standing surface to level of maximum hip circumference</td>
</tr>
<tr>
<td>Knee Height, Sitting</td>
<td>Vertical distance from foot support surface to the top of knee</td>
</tr>
<tr>
<td>Neck Base Circumference</td>
<td>Circumference of neck measured at the juncture of neck and shoulder</td>
</tr>
<tr>
<td>Shoulder Breadth</td>
<td>Horizontal distance between maximum protrusion of left and right shoulder</td>
</tr>
<tr>
<td>Sitting Height</td>
<td>Vertical distance sitting surface to the highest point of the head</td>
</tr>
<tr>
<td>Weight</td>
<td></td>
</tr>
<tr>
<td>Thigh Circumference</td>
<td></td>
</tr>
<tr>
<td>Total Crotch Length</td>
<td>Distance from front to back of the waist passing through the crotch</td>
</tr>
<tr>
<td>Waist Circumference</td>
<td></td>
</tr>
<tr>
<td>Waist Front Length</td>
<td>Distance from front neck base to front of waistline in the median plane</td>
</tr>
<tr>
<td>Waist Height</td>
<td>Vertical distance from waist to the standing surface</td>
</tr>
<tr>
<td>Weight</td>
<td></td>
</tr>
<tr>
<td>Armscye Circumference</td>
<td>Circumference passing acromion (shoulder) and armpit</td>
</tr>
<tr>
<td>Chest Circumference</td>
<td>Circumference of the torso at nipple level</td>
</tr>
<tr>
<td>Bust/Chest Circumference Under Bust</td>
<td></td>
</tr>
<tr>
<td>Buttock-Knee Length, Sitting</td>
<td>Horizontal distance from foremost point of kneecap to rearmost point of buttock</td>
</tr>
<tr>
<td>Chest Girth at Scye</td>
<td>Max circumference of torso passing under the arms and across upper chest</td>
</tr>
<tr>
<td>Crotch Height</td>
<td>Vertical distance between crotch and standing surface</td>
</tr>
<tr>
<td>Eye Height, Sitting</td>
<td>Vertical distance between sitting surface to outer corner of the eyes</td>
</tr>
<tr>
<td>Foot Length</td>
<td>Maximum distance from the rear of the heel to the tip of the longest toe</td>
</tr>
<tr>
<td>Hand Circumference</td>
<td>Circumference hand passing knuckles of index and little fingers</td>
</tr>
</tbody>
</table>
Figure 5.3 Examples of 3D models generated from the CAESAR sample measurements.

Figure 5.4 Daz3D screenshot.

The template mesh is displayed on the left viewport in Figure 5.4. The top right panel displays the current measurement of the template mesh. Note that some measurements require the avatar to be seated and in these cases we take the measurement from the template mesh by setting the joint rotation of the avatars. In Figure 5.4, the bottom right
panel shows some applicable mesh deformers. We wrote a script to apply the Daz3D mesh deformers \( P = \{p_1, p_2, \ldots, p_n\} \) to morph the template mesh \( T \) using weights \( w_i \). The post morph measurement \( M_t \) of the model is therefore:

\[
M_t = Measure \left( T + \sum_{i=0}^{n} w_i * p_i(T) \right)
\]  

(5.1)

Our script adjusts the morphs iteratively and different measurements are recorded and compared against the desired measurements. To conform the avatar to the subject measurements in CAESAR, we use conjugate gradient descent to minimize the error \( \epsilon = |M_d - M_t| \) between the template mesh measurements and the desired ones. A set of clothes for the template female and male meshes were created by an artist, and morphed in Daz3D to fit each of our selected models. The models are exported to Morpheme where retargeting and animation occurs, and finally they are imported into our crowd system. There they are rendered with color variation to create our final varied crowd simulation.
Figure 5.5: Top: Close-to-median body shapes with various colored clothes (17 males and 14 females). Bottom: Hour-glass shaped pear-shaped female with V-shaped and block-shaped male.
5.4 Experiments

We conducted an online perceptual experiment to investigate the influence of body measurements on the perception of body shapes that are close to the median, as these are the templates that will be most often replicated in a crowd. We are particularly interested in the distinctiveness of the different body shapes, and as previous research shows how attractiveness is closely related, we also explore this metric. We also ran a laboratory-based experiment with a different method and smaller number of participants to gain further insights.

Figure 5.6: Example models used in Experiment 1, sampled near the median: models picked to represent the median group (left); most distinctive and most/least attractive models (right).
**Stimuli:** We select 32 male and 32 female CAESAR samples with girth and height closest to the median (see Figure 5.2) and generate the 3D models that match their measurements, as described in Section 5.3. We also generate average male and female models, as previous research in face and body motion perception has shown that the average is usually amongst the least distinctive and most attractive. In total, we have 33 models for each gender. Some of the models used can be seen in Figure 5.6.

**Method:** In order to recruit as many participants as possible, we created several online surveys and posted the links on Mechanical Turk (MT). The HITs were available only to ‘master MTurkers’, i.e., those who have a good performance track record. The study was approved by an institutional ethics review board and all participants provided informed consent. After first removing the responses of participants whose accuracy was too low (97 out of 479), we analyzed the results of 382 participants (206F/176M). Participants are shown a set of 99 pages, showing either all male or female models depending on the hit. A target model is shown on the left, and three smaller sample models are shown on the right (in Figure 5.7). One of the sample models always matched the target, while the other two distractor models are chosen at random from the remaining 32. The task was to first rate the attractiveness of the target on a 7-point Likert scale from 1 (very unattractive) to 7 (very attractive); then to select the sample on the right that was the same as the target. Three repetitions of each target were shown, and the order of all pages was randomized. Each page was viewed for between 5-10 seconds so the experiment took approximately 15-20 minutes on average.
Figure 5.7: Survey page used in our experiment.
As it would be infeasible to generate all possible combinations of the 33 models as the time and/or number of participants needed would be prohibitive, we created two surveys each for male and female models. This means that each target was compared with a total of 12 distractor models. To ensure that we were not introducing bias with this limitation, we also ran a laboratory-based experiment with 17 participants (8M/9F). We followed a similar procedure to Hoyet et al. [43], where first a group of two side-by-side models was shown, then one target model, and the task was to indicate using one of two keys whether the target was Present or Absent from the previous group. In 50% of cases the model was present. One group of 9 participants (include both male and female participants) viewed the Female models and one group of 8 viewed the Male models. There were 4 repetitions of each target model (2 present, 2 absent) and the distractor models were selected at random.

5.4 Result

Analysis of Variance (ANOVA): To ensure that differences between models were indeed being noticed, we first performed ANOVA on the average accuracy values and attractiveness ratings for the Mechanical Turk experiment. For accuracy (i.e., mean percentage of correct identifications), we found a main effect of Model for the 33 female models ($F(32,7680) = 50.06, p < 0.00005$) and $F(32,7520) = 30.65, p < 0.00005$). For the attractiveness ratings, we also found a main effect of Model for both Females ($F(32,7680) = 134.01, p < .00005$)) and Males($F(32,7520) = 136.83, p < .00005$)).

We also analyzed the factors of participant sex, age group and the self-reported display device used and found no significant effects.
We also performed repeated measures ANOVA on the accuracy results from the Laboratory experiment for Males and Females separately (a between-groups analysis showed no main effect of Model Sex). We found a main effect of both Female ($F(32,256) = 1.88, p < .005$) and Male ($F(32,256) = 1.97, p < .00005$) Models. We also evaluated whether presence or absence detection accuracy were different, and found that the former was significantly higher than the latter ($F(1,8) = 25.70, p < .0005$). However, a two-way interaction between presence/absence and Model for the Female models ($F(32,256) = 1.49, p < .05$) indicated that for some actors, absence detection was as good as presence detection, in one case (90%) whereas for others it was as low as 20%. There was much less variability in the scores for presence detection, with participants scoring above 80% on average.

**Correlation Analysis:** One of the main goals of our experiments was to determine which body shape features have the strongest effect on body shape perception. Hence we wished to assess the correlations between our results and the body measurements that we used to create the 3D models. Based on the observation that absence detection appeared to yield greater variability between models in the Laboratory experiment, we decided to use five metrics based on our results: the Mechanical Turk Hit rate (MT-H), which was calculated for each model as the percentage of times it was accurately matched; the Mechanical Turk False alarm rate (MT-F), i.e., for each model, we calculated the number of times it was wrongly selected divided by the number of times it was seen as a distractor (note that we could not do this on a per-participant basis, as the distribution of distractors was random);
the Laboratory Hit (LAB-H) and False alarm (LAB-F) rates; and the average Attractiveness (ATT) ratings.

From viewing the images of the ranked models, it was clear that the women models fell into one or other of two very visually distinct groups: those with symmetric upper and lower bodies (i.e., hour-glass) and those whose lower body was wider than the top (i.e., pear-shaped). Further analysis of the male images showed that the two main body types were V-shaped (i.e., shoulders larger than the lower body) and Block (upper and lower body widths more or less similar). Furthermore, from the cited literature we know that Waist to Hip Ratios (WHR) in women and Waist to shoulder or Chest Ratios (WCR) in men have been found to be strong predictors of Gender and Attractiveness. In order to determine whether the Females were more Hourglass or Pear-shaped, we also calculated WCR-WHR for the females (i.e., indicates symmetry between the upper and lower body). We also found high correlations with these measurements and factors as shown in Table 5.2.
### Table 5.2: All significant correlations

<table>
<thead>
<tr>
<th>Female Correlations</th>
<th>All Females</th>
<th>Hourglass Females</th>
<th>Pear Females</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MT-H MT-F LAB-H LAB-F ATT</td>
<td>MT-H MT-F LAB-H LAB-F ATT</td>
<td>MT-H MT-F LAB-H LAB-F ATT</td>
</tr>
<tr>
<td>M Turk Hit Rate (MT-H)</td>
<td>-0.5 -0.6 -0.8 -0.7 -0.9</td>
<td>-0.8 -0.9 -0.5 -0.6 -0.7</td>
<td>-0.7 -0.9 -0.7 -0.8 -0.6</td>
</tr>
<tr>
<td>M Turk False Alarm Rate (MT-F)</td>
<td>-0.5 -0.6 -0.8 -0.9 -0.7</td>
<td>-0.5 -0.9 -0.6 -0.7 -0.8</td>
<td>-0.9 -0.7 -0.7 -0.8 -0.6</td>
</tr>
<tr>
<td>Lab Test Hit Rate (LAB-H)</td>
<td>- - - - - - - - -</td>
<td>- - - - - - - -</td>
<td>- - - - - - - -</td>
</tr>
<tr>
<td>Lab Test False Alarm Rate (LAB-F)</td>
<td>-0.6 0.7 -0.8 -0.7 -0.6</td>
<td>0.8 -0.7 0.5 -0.6 -0.8</td>
<td>-0.6 0.7 -0.6 0.3 -0.3</td>
</tr>
<tr>
<td>Attractiveness Ratings (ATT)</td>
<td>0.5 0.6 0.7 0.8 0.9</td>
<td>0.6 0.7 0.8 0.9 1.0</td>
<td>0.6 0.7 0.8 0.9 1.0</td>
</tr>
</tbody>
</table>

| Chest Circumference (mm)              | 0.4 - - - - | - - - - - - | - - - - - - |
|                                       | 0.1 -0.1 0.4 0.8 | 0.8 -0.7 0.4 0.6 0.9 | -0.4 0.7 -0.4 -0.4 0.6 |
| Hip Circumference, Maximum (mm)       | - - - -0.5 - | - - - - - - | - - - - - - |
|                                       | - - - - - - | - - - - - - | - - - - - - |
| Waist to Chest Ratio (WCR)            | 0.2 -0.3 0.1 -0.4 -0.8 | 0.8 0.8 -0.2 0.6 -0.8 | 0.6 -0.6 0.3 -0.3 -0.7 |

### Male Correlations

<table>
<thead>
<tr>
<th>Male Correlations</th>
<th>All Males</th>
<th>VShape Females</th>
<th>Block Females</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>MT-H MT-F LAB-H LAB-F ATT</td>
<td>MT-H MT-F LAB-H LAB-F ATT</td>
<td>MT-H MT-F LAB-H LAB-F ATT</td>
</tr>
<tr>
<td>M Turk Hit Rate (MT-H)</td>
<td>-0.8 -0.5 -0.9 -0.5 -0.7</td>
<td>-0.8 -0.9 -0.5 -0.6 -0.8</td>
<td>-0.7 -0.9 -0.7 -0.8 -0.9</td>
</tr>
<tr>
<td>M Turk False Alarm Rate (MT-F)</td>
<td>-0.8 -0.4 0.6 -0.9 -0.7</td>
<td>-0.5 0.6 -0.6 -0.6 0.6</td>
<td>-0.5 0.6 -0.5 -0.5 0.5</td>
</tr>
<tr>
<td>Lab Test Hit Rate (LAB-H)</td>
<td>-0.4 - - - - -</td>
<td>-0.4 -0.7 0.5 -0.6 -0.8</td>
<td>-0.4 -0.7 0.5 -0.6 -0.8</td>
</tr>
<tr>
<td>Lab Test False Alarm Rate (LAB-F)</td>
<td>-0.5 0.6 -0.6 -0.6 0.6</td>
<td>0.4 -0.7 0.4 -0.6 0.9</td>
<td>0.4 -0.7 0.4 -0.4 0.6</td>
</tr>
<tr>
<td>Attractiveness Ratings (ATT)</td>
<td>-0.4 - - - -</td>
<td>-0.4 -0.7 0.5 -0.6 -0.8</td>
<td>-0.4 -0.7 0.4 -0.4 0.6</td>
</tr>
</tbody>
</table>

| Chest Circumference (mm)              | 0.2 -0.3 0.1 -0.4 -0.8 | 0.8 0.8 -0.2 0.6 -0.8 | 0.6 -0.6 0.3 -0.3 -0.7 |
| Waist to Chest Ratio (WCR)            | 0.2 -0.3 0.1 -0.4 -0.8 | 0.8 0.8 -0.2 0.6 -0.8 | 0.6 -0.6 0.3 -0.3 -0.7 |
Figure 5.8: Distinctiveness and Attractiveness results from Experiment 1: the red dots show the models we chose for the median group. The average Male and Female models were also chosen.

We selected Hourglass and Pear female models from those above and below the median of WCR-WHR; and V-Shaped and Block males similarly, based on the median WCR. In Figure 5.8, we plot the average attractiveness (ATT) ratings against the average distinctiveness (MT-H) for these models. We can see that for the Hourglass group, attractiveness and distinctiveness are positively correlated (0.9), whereas for the Pear group they are not (the only reason there is not a significant negative correlation is due to an outlier pear model who was much thinner and lighter than all the others). Similarly, for the
V-Shaped males, there is a significant positive correlation between distinctiveness and attractiveness (0.8) and a negative one for the Blocks (-0.8). The most distinctive hourglass female was also the most attractive female overall, with the most distinctive pear the least attractive, with the same being true for the male V-Shaped and Block models (Figure 5.6:right).

These results confirm our intuition that these different body types are perceived very differently by participants, and hence we made our choice of the three representatives of the Median groups (Figure 5.6:left) as templates to represent median body shaped human in our crowd simulation. The non-median body shape templates are selected using stratified sampling in section 5.3. By choosing the average Male and Female (both of whom had the median WCR or WCR-WHR and were amongst the least distinctive of all models), along with one of each of the Hourglass, Pear, V-Shaped and Block models (chosen to be not overly distinctive but reasonably different from each other), the median group is perceived to have enough variety.

5.5 Crowd Demo

In our crowd demo, with our selected non-median templates, we replicate each template by the number of samples in their group. For the median group, since there are three templates, we divide the number of samples evenly and replicate each template by the number. There are in total 3 different outfit sets designed for male and 4 outfits for female from artists. Note that once the outfit is made for the template mesh Figure 5.4, we are able to adapt the clothes while we generate avatars with different measurements. Also, varying
color scheme is used on the clothes, hair and shoes to increase the variety perceived by the audience. In Figure 5.9 (a)(b), we demonstrate the crowd in a close shot with people walking or talking to each other. A further away shot shows the people walking around the plaza (Figure 5.9 (c)). Various body shapes can be spotted here (including plus size, extra small), while the majority of people are of regular size.

Figure 5.9: Close-up from the crowd simulation.
5.6 Conclusion

We have found that the most visually salient properties of body shape for models near the median are hips, chest and their ratios with waist size. Furthermore, the types of body that is defined by these ratios, i.e., V-shape or Block for males; Hourglass or Pear for females, are perceived differently. Based on these results, we propose a selection strategy to pick the template models to use to represent the median population in a crowd. Together with the samples we selected from the other body sizes, we generated a set of crowd template models that is both varied and representative of a real population (see Figure 5.5 and Figure 5.9).

This is a joint work with Disney Research with Dr. Carol O’Sullivan, Dr. Jan Ondřej and Dr. He Wang. My contribution includes developing an algorithm to procedurally generate 3D meshes with 1D measurement from the CAESAR database, fitting the clothes to the avatars, helping create online surveys and examining the data from the online survey. I also contributed to the final demo of our work. Jan Ondřej is in charge of creating crowd demo (include retargeting animation and employing different color scheme) and setting up the lab experiment. He Wang contributes to creating the online survey and data analysis, he is also in charge of creating the final demo. Carol O’Sullivan is the head of our team and she helps select and guide the research, as well as set up and analyze both online/lab experiment.

There are limitations to our work, in that we have only explored similarity for a very small group of body shapes, and we did not perform a full confusion analysis due to the nature
of the experimental data. We also did not assess the overall variation perception of the full crowd, which is an interesting direction for future work.
Chapter 6 : Conclusion

In this dissertation, we present a framework to evaluate the procedural generated content for computer games with gameplay experience. We examined two aspects of the procedural content generation: level generation and body shape variation. With each of those aspects, we proposed a way for employ gameplay experience for evaluation. For procedural level generation, a virtual player is placed in the game scene to model the player’s behavior. By studying the virtual player’s experience, we infer the gameplay experience of the level. For procedural body shape variation, gameplay experience is measured by visual realism such as representation and distribution of the body shapes.

With the definition of the damage flux, we define the optimal path as the least resistance path for the virtual player to go through the scene. Under this premise, game scenes with different cover and enemy configurations will demonstrate different optimal paths, which allows us to evaluate the game scene indirectly by evaluating its optimal path. We proposed four metrics as follows. (1) Least damage accumulated along the optimal path. This metric is suitable for finding a level with stealthy gameplay. (2) Longest optimal path length. This metric is useful to maximize the gameplay with given game assets. (3) Largest standard deviation for the damage on the optimal path. This metric is applicable in designing the
game level with changing pacing; (4) Designer specified damage curve for optimal path, which can be applied to shape and tweak the desired flow of the game scene. For a stealth game, we used this framework with an iterative approach to search for the cover placement against static enemy positions. We demonstrate our approach by running two simulations with random sampling of cover placement. By searching for maximum amount of iterations, we were able to find the desired cover placements for each of our metrics.

To evaluate strategy game level, we proposed an algorithm to find the global optimal of squad tactics. Unlike the stealth game, squad members also provide suppression fire to his/her allies when taking covers. After examining the game scene, we constructed a cover graph to simulate the possible paths for the squad to advance. To represent the state of the squad, we then build a cover state graph to capture the state change of one squad member as he/she dashes towards a cover location while the rest of the team stay to provide suppression fire. The optimal squad tactics are obtained by searching the minimal damage path from the starting state to the end state. The optimal squad tactics are deterministic and can therefore be used as a comparison to evaluate how other AI performs in the same setting. For future work, we may evaluate the strategy game level by examining the damage curves for the squad movement with the similar metrics.

To procedurally generate and evaluate a large game scene with random sampling in our framework is cost prohibitive. In this case, we proposed an algorithm to generate a path tile set and a tiling scheme to aid the designer in creating a large open map for a stealthy action game. Path tile set is constructed as Wang tile set. With a matching cover edges and
target zones, a player is ensured to have a consistent gameplay when crossing the adjacent path tiles. For path tiles, we used our framework to create the tile sets with the first three metrics. By fixing the entrance and exit locations within the target zones, we use the tile path to represent the virtual player’s path in a tile. With a Poisson point sample for a tiles’ candidate cover and enemy locations, we enumerate all possible configuration to find the best tiles that suit our metrics. Given the pre-generated path tiles, the designer sketches the path for the game scene. A set of navigation tiles are used to fit out the designer’s sketch. For the final game scene, navigation tiles are swapped with path tiles with constraints to allow the designer to tweak the game flow. In the future, path tiles can be expanded to support branching, which will allow the designer to create more complex gameplay.

To evaluate the body shape variation in crowd simulation, we used a perception based method. By examining the distribution of general population’s measurements, we discovered the most salient features for the non-median crowd are height and waist circumference. We developed an algorithm to procedurally generate the body shapes with measurements. The visual realism depends on the distinctiveness of body shapes. For the median crowd, conducted two user studies to evaluate the distinctiveness of the generated median body shapes and investigate the most salient features other than height and girth. We found the waist to hip ratio (WHR) and waist to chest ratio (WCR) for female, and waist to chest ratio for male are the most salient. Based on WHR and WCR ratios, we selected Hourglass shaped and Pear shaped for females, V-shaped and Block shaped for males to represent the median crowd. For non-median crowd, we selected the avatars with the stratified sampling. By replicating the template avatars proportionally, we are able to
generate a crowd as varied and representative as real-population. For future work, another perceptual study of the examining the visual clones in a whole crowd can be done to validate our approach.

Our framework demonstrates a way to allow the user to plug in various procedural content generation method and evaluation metrics for gameplay experience. For the future, one possible direction is to expand the framework to support human-in-the-loop design process. Instead of simulating a virtual player in the game scene to perform certain task and gather feedback, designer can iteratively guide the search process into the desired result.
Bibliography


