MODELLING PLAYER UNDERSTANDING OF NON-PLAYER CHARACTER PATHS

by

Mengxi Xoey Zhang

School of Computer Science
McGill University, Montréal

August, 2019

A THESIS SUBMITTED TO THE FACULTY OF GRADUATE STUDIES AND RESEARCH
IN PARTIAL FULFILLMENT OF THE REQUIREMENTS FOR THE DEGREE OF
MASTER OF SCIENCE

Copyright © 2019 Mengxi Xoey Zhang
Abstract

Modelling a player’s understanding of non-player character (NPC) movements can be useful for adapting gameplay to different play styles. For stealth games, what a player knows or suspects of enemy movements is important to how they will navigate towards a solution. In this work, we build a uniform abstraction of potential player path knowledge based on their partial observations. We use this representation to compute different path estimates according to different player expectations. We augment our work with a user study that validates what kinds of NPC behaviour a player may expect, and develop a tool that can build and explore appropriate (expected) paths.

From the study, we found that players tend to prefer short simple paths over long or complex paths with looping or backtracking behaviour. We also apply our methods to real game situations to expand on our approach’s benefits and limitations. We ran experiments on real and artificially created scenarios of NPC paths and evaluated the generated pathing behaviours and found that simple shortest paths tend to be most similar to actual NPC paths. We also discuss path simplification, as well as measuring similarity between two paths using discrete Fréchet (and a modified extended version that considers paths around obstacles). We use area and volume to measure level difficulty and we examine the challenges of accurately representing player observation and the endeavour of translating player observations into a valid representation of player knowledge.
Résumé

Il peut être utile de modéliser la compréhension du joueur des mouvements de personnages non-joueurs (PNJ) pour adapter un jeu vidéo à différents styles de jeu. Pour les jeux furtifs, ce qu’un joueur sait ou soupçonne des mouvements de l’ennemi est important pour la manière dont il ou elle va naviguer le niveau pour trouver une solution. Dans cette thèse, nous construisons une abstraction uniforme de la connaissance du trajet potentiel des joueurs en fonction de leurs observations partielles. Nous utilisons cette représentation pour calculer différentes estimations de trajets en fonction des attentes des différents joueurs. Nous renforçons notre recherche avec une étude qui valide les types de comportement des PNJ auxquelles un joueur peut s’attendre et nous développons un outil qui peut créer et explorer des trajets appropriés (et attendus).

D’après notre recherche, nous avons constaté que les joueurs ont tendance à préférer les trajets courts et simples aux trajets longs ou complexes (c’est-à-dire des trajets avec boucles ou backtrack). Nous mettons en œuvre également nos méthodes à des situations de jeux réelles et élaborons sur les avantages et les limites de notre méthode davantage. Nous avons effectué des expérimentations sur des scénarios de trajets PNJ réels et créés artificiellement et avons évalué les trajets générés. Nous discutons également la simplification des trajets, ainsi que de la mesure de la similarité entre deux trajets à l’aide de la méthode Fréchet discret et d’une version modifiée et améliorée qui tient compte des obstacles. Nous utilisons aire et volume pour mesurer le niveau de difficulté du jeu. Nous examinons aussi les défis de la représentation précise des observations dans le jeu, et les problèmes qu’encompasse la translation des observations des joueurs en une représentation valable du savoir des joueurs.
I would like to express my deepest appreciation to my supervisor, Clark Verbrugge, for his invaluable advice, support and guidance. My research and thesis would not have been possible without his insightful suggestions and unwavering patience. I am also appreciative for the financial support provided by the Natural Sciences and Engineering Research Council (NSERC), through their COmputing Hardware for Emerging Intelligent Sensing Applications (COHESA) project and grant, and for McGill’s Grad Excellence Award in Computer Science and Graduate Research Enhancement and Travel (GREAT) award.

I am thankful to my parents, for their continuous encouragement. Special thanks to my sister Lucy, for bringing joy into my life, and to my best friends for always believing in me. Above all, I am most grateful for my husband, Jason Biem, for offering me his endless love, support and encouragements, for fixing my English and for keeping me sane through the toughest times.
# Table of Contents

Abstract i  
Résumé ii  
Acknowledgements iii  
Table of Contents iv  
List of Figures vii  

1 Introduction 1  

2 Background and Related Work 5  
  2.1 Background ................................................................. 5  
  2.2 Related Work ............................................................. 8  

3 Methodology 11  
  3.1 Representing Movement Knowledge ................................. 12  
  3.2 Definitions ............................................................... 12  
  3.2.1 State Space ............................................................ 12  
  3.2.2 Player and Guards .................................................. 15  
  3.2.3 NPC Movements ...................................................... 16  
  3.2.4 Observations .......................................................... 16  
  3.3 Filling in the Gaps ....................................................... 17  
  3.3.1 Level Setup .......................................................... 17
List of Figures

2.1 Screen capture of Dishonored gameplay where the player must hide from the enemy and observe them. ......................... 6
2.2 Screen capture of Styx: Master of Shadows gameplay where the player is perched above the enemies to avoid detection. Image was enhanced (brightened) for clarity. ......................... 7
3.1 Example of a complete path ........................................ 13
3.2 Example of a partial path with different ways to fill in unknown gap segments 14
3.3 Example of a level setup where tiles \{0, 1\}, \{0, 2\} and \{2, 3\} are obstacles. 17
3.4 End result of the labelling algorithm showing each tile labelled with \([l_a, l_b]\).
   Colouring of each tile is based on \(\Delta t = 7\) and shows the maximum amount of time the character can wait on that tile. ......................... 19
3.5 Resulting search space based on increasing the amount of time elapsed without changing the position of start or end of a gap segment. ......................... 20
3.6 Top and isometric view of filling in a gap segment ........................ 21
3.7 Example of the algorithm in action showing the breakdown at every time step. ............................................. 23
3.8 Permutation zones shown on maps with and without obstacles, and how they affect paths found within the level. ........................ 24
3.9 Example of two permutation zones in one gap segment. ................. 25
3.10 Circled in red is a guard seen paused (waiting) in front of an interest point (painting). ............................................. 26
3.11 Example of a gap with $\Delta t = 8$. The two blue paths are equally minimal in length (4), but imply the NPC paused at some point to make up the 4 extra time units. The dotted red path is arguably less realistic in being indirect, even if it implies continuous movement. Turning here is assumed instantaneous.

3.12 Example of two separate observations contributing to the knowledge of the same NPC’s movement pattern.

3.13 Example of two possible paths to fill the gap segment between observed path segments $p_1$ and $p_2$ without wait time.

3.14 A screenshot of an example with unreachable tiles, based on the starting and ending positions of the gap segment.

3.15 Buttons, checkboxes, slider and $\Delta t_{\text{min}}$ labelled in a partial screen capture of the tool interface for clarity.

3.16 A screenshot of the tool built to analyze gap segments. The red, green and blue lines represent generated paths of different styles. Green square is the beginning of gap segment whereas blue square is the end of the gap segment. White dot is used to animate the simulation of the enemy or guard taking the paths.

3.17 Two separate views: top-down and 3D-rotating side view of paths generated using our tool for the gap segment indicated by the green and blue squares.

3.18 Example of shortest simple path, exact steps and interesting paths generated by our tool on a larger map, with $\Delta t = 26$, manually defined segments and a hidden heatmap.

3.19 Representation of two gap segments on the same map. Overlaying heat maps makes it difficult to distinguish the computed path information between the two gaps.

4.1 A screen capture of a question in the survey.

4.2 Survey results when comparing surveyed players’ preferences for simple paths and different complex paths out of 104 respondents.
4.3 Two examples of gap segments, one complex and one simple, both from the same complete path (on the left). A correct segment has been generated for the simple gap, and an incorrect one for the complex gap.

4.4 An example of an NPC moving in a straight line diagonally across the map.

4.5 Examples of different paths that require approximations.

4.6 Example of a curve through multiple iterations of RDP.

4.7 Example of simplifying a multi-segment path using RDP and a modified RDP.

4.8 Illustration of the man and dog analogy. The dog travels along the red line, while the man on the blue. The dotted lines represent leash length throughout the walk.

4.9 Example showing that despite two paths 1 (red) and 2 (blue) being identical, the Fréchet Distance between them is non-zero because path 2 has one more point.

4.10 Modified Fréchet allows us to penalize paths that are not on the same side of an obstacle.

4.11 Example of the same man and dog analogy as Figure 4.8, but using the modified Fréchet algorithm that considers obstacles.

4.12 The full path drawn on the map as a reference.

4.13 The example variations of dropping various small-size segments of the path (i.e. gaps of size \( t = 4 \)).

4.14 Result of filing in the gaps of size \( t = 4 \).

4.15 The example variations of dropping various medium-size segments of the path (i.e. gaps of size \( t = 6 \)).

4.16 Result of filing in the gaps of size \( t = 6 \).

4.17 Partial paths selected for gaps of size \( t = 10 \).

4.18 Visualization of the resulting areas of where the NPC may be during the gap of all the variations of examples 1, 2 and 3.

4.19 Top and 3D view of an example of Fréchet distance calculated in 3D space.
4.20 One of the “Shortest With Waiting” paths generated with $\Delta t = 15$. Results for the set of paths (not indicated in this figure) are: Fréchet Distance 1.80, and average distance 0.90. .......................... .................................................. 73

4.21 One of the “Exact Steps” paths generated with $\Delta t = 15$ and $\Delta t_{\text{min}} = 7$. Results for the set of paths (not indicated in this figure) are: Fréchet Distance 1.41, and average distance 0.76. Distances may seem distorted, caused by the 3D figure and nature of the representation. .......................... .................................................. 74

4.22 An example where the top two images represent the top-down view and 3D view of a gap with a $\Delta t$ of 7, and the bottom two images are views of a gap that has a $\Delta t$ of 8. .......................... .................................................. 76

4.23 Orthogonal view in *Pillars of Eternity*, player sees a portion of the map (bottom image shows the region highlighted in first image), and can observe the movements and paths taken by the NPC enemy. .......................... .................................................. 78

4.24 A sequence of actual gameplay from *Pillars of Eternity* where a gap is created when the guard and characters get too far from each other. Magenta arrows were added to show the observed guard movements. The screen captures were enhanced (in brightness and contrast) for clarity. .......................... .................................................. 79

4.25 A sequence of actual gameplay from *Thief* where the player watches the guard for a full cycle before proceeding. The screen captures were enhanced (in brightness and contrast) for clarity. .......................... .................................................. 80

4.26 A screen capture of the game Dishonored: example of the player observing a guard (circled in red) and his movements from a distance without being detected. .......................... .................................................. 81

4.27 A screen capture of the game Dishonored: example of the player observing multiple guards patrolling an area while staying hidden. .......................... .................................................. 81

4.28 Observed actual gameplay result in very short and obvious gaps. The area of gap segment is 4, and its volume of possible positions is 4. .......................... .................................................. 83

4.29 Screen captures of observed actual gameplay result in very short and obvious gaps. .......................... .................................................. 84
4.30 Screen capture of an instance of actual gameplay where the gap is not necessarily direct but still obvious. Top picture shows the player observing the guard through the keyhole, and the second shows the layout of the room and level once the player enters the room. 

4.31 Path segment representations of what was observed (left), the path generated for the gap (middle), and the 3D representation of it (right), map from the level depicted in Figure 4.30.

4.32 Comparison of area and volume as $\Delta t$ increases for the level from Figure 4.30.

4.33 Comparison of area as $\Delta t$ increases.

4.34 Series of screen captures showing a gap segment created by player distancing themselves from the guard to go around an obstacle before seeing the guard from the other side of it.

4.35 Gap from Figure 4.34 filled in using the tool. Left picture represents what the user observes and the right shows one of the generated paths.

4.36 Results of the level in terms of area ($17 \text{ unit}^2$) and volume ($28 \text{ unit}^3$) in the level depicted in Figures 4.34 and 4.35.

4.37 Screen capture of gameplay in Aragami, where the player needs to avoid multiple guards (on different paths) at once, and must stay in the shadows to prevent being detected. Two guards are visible, while 3 others are in vicinity, but out of sight.

4.38 A guard’s path in Aragami.

4.39 Screen captures comparing five examples of select path generation results between 4-way and 8-way models. White Lines represent known paths, blue lines are generated paths, between start (green square) and end (blue square) of gap segment.

A.1 Example of a game where player must avoid patrolling enemies that are non-player characters (NPC).

A.2 Figure explaining the different elements of the animated survey questions and format.

A.3 First 2 questions of the survey.
A.4 Questions 3 and 4, following the introduction of waiting. . . . . . . . 109
A.5 Question 5. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 110
A.6 Question 6. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 111
A.7 Question 7. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 112
A.8 Question 8. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 113
A.9 Question 9. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 114
A.10 Question 10. . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . . 115
Chapter 1
Introduction

The difficulty of a stealth or combat game is strongly affected by player knowledge of enemy movements. Previous playthroughs, outside knowledge, as well as general gaming experience allow players to make increasingly strong assumptions about non-player character (NPC) behaviours, and thus better predict or identify safer or more strategic movement options in game levels. Modelling player assumptions and expectations of NPC movements provides an insightful mechanism for understanding how players may attempt to solve a level, particularly over repeated play. It also offers potential for adapting game difficulty by dynamic modifications that either conform to or violate expectation.

Modelling player knowledge is important when creating content for modern digital games. It allows game developers to create and modify game levels and content based on the player’s progress, knowledge, preconceptions and play style. Most modern digital games use some form of artificial intelligence in-game, whether it is content generation, enemies or non-player characters, to create interesting game play and interactions. Various data and information is gathered after each save point or in real-time, and often the content in the game is slightly adapted to each play-through situation in such a way that the story line is cohesive and interesting for the player.

In the context of video games, as for many other situations, a human player can be of any experience and skill level, which is difficult to measure, as it is a combination of previous knowledge of games the player has played in the past, as well as what the player knows about the current game that they are playing. Player styles vary greatly too: there
are a multitude of approaches when it comes to playing any game, whether it is a puzzle, a stealth, strategy or combat game. This is because there are often different approaches to reaching one common goal, and depending on a player’s play style and experience, this may vary greatly. For example, in a game where the end goal is to reach the finish line inside a maze with patrolling guards, a player may decide to use a brute force method and speed past guards towards the exit, or they may decide to observe the guards’ exact movements to plan their escape strategically. This makes it difficult for game developers to create games that are appealing to all kinds of players, and to generate well-balanced, fun and interesting game situations. For game developers, one way of appealing to multiple kinds and levels of players is to create games that are adaptive and interesting in the way that would intrigue and captivate each player in different and customized ways.

Ideally, for instance, players that enjoy solving puzzles would find more challenging riddles in the game, and players that enjoy suspense and surprise elements would stumble upon more guards and unexpected behaviours from non-player characters. Adaptive gameplay is being integrated in many modern computer games, such as the stealth game ECHO [52] in which the gameplay revolves around NPCs learning from the player’s behaviour and mirroring its actions. More commonly, other games such as Left 4 Dead [53] will have an AI element (e.g. AI Director [8]) that generates enemies in varying locations and numbers based on the player’s situation, state, skill level, etc., creating different experiences for each play-through, and for different players. Although these games adapt to the player based on certain in-game numbers and statistics, player knowledge is infrequently modelled in a comprehensive way. Understanding and concretely modelling what a player knows about a scene, or a game situation, is an interesting way to collect data about the player’s knowledge and style, and is one step closer to determining what elements of a game can be modified to suit a player’s play style.

In this work we describe an algorithmic approach to estimating and representing player knowledge of NPC movements. We base our design on a uniform representation of partial observations, applying algorithmic and heuristic techniques to build different estimations of position and expected movement. We do not aim at open, unbounded prediction; rather, our focus is on how players may fill the gap between two observations, reflecting the use of prior, partial knowledge that provides consistent, but not identical information, as may
be acquired from repeated playthroughs.

The full extent of individual player knowledge is of course beyond what can be acquired from simple game logging. Our approach is aimed at building the core techniques that enable an exploration tool to better understand and experiment with what information becomes available to a player, given specific NPC routes, level geometry, and incomplete observation. We thus define a baseline system that allows for exhaustive modelling of possible NPC positions, constrained by the gap-time and filtered by knowledge of level geometry. Players may also attribute movement characteristics or make assumptions about NPC behaviours as well. Our design naturally incorporates different constraint models that reduce pathing possibilities to better represent player expectation.

Algorithmic and representational design is supported by a non-trivial experimentation and visualization tool built into Unity®. This allows for flexible exploration of different, constrained models of player expectation given different segments of NPC observation. The choice of path constraints we offer is further justified by data gathered from a small user study. Additionally, we use NPC movement data from real-game scenarios, as well as simulated ones, to further investigate and validate the paths generated by our tool.

Our work is intended to facilitate design exploration of NPC movements and geometry, and to provide a tool for further understanding of how gameplay may be learned and optimized through repeated player experience. Specific contributions of this work include the following:

- We construct a generic model of player path observations which accommodates a variety of possible observation sources and expectation constraints.

- We provide an algorithm for computing an exhaustive representation of possible movements, as well as extensions that either work through our formal representation or incorporate simple constraints to reduce the set of possible behaviours according to different models of player expectation.

- Our approach is implemented and demonstrated in a non-trivial tool within the open-source Unity® framework. This allows for flexible visualization and exploration of expected movement under different constraints.
• Our design approach is supported by a small-scale user study that shows players generally expect NPC to follow simple, non-looping paths with minimal backtracking.

• Our Unity tool is extended to generate NPC path fragments, which we then compared with paths from actual game NPC movements in order to find the approach that most accurately represent NPC movement in real game scenarios. This effort includes a wide-ranging consideration of different confounding factors that affect algorithmic construction of pathing predictions.

The upcoming chapters are organized as follow. In Chapter 2 we discuss background and related work. In Chapter 3 we describe the methodology used to represent NPC movement and player knowledge, as well as how we created a tool to visualize the problem and to generate path segments. In Chapter 4 we conduct experiments on real games and artificial NPC paths that evaluate paths generated by our model. Chapter 5 concludes our work and discusses potential future directions.

Part of this thesis was published as a paper with the same title in the 2018 Artificial Intelligence and Interactive Digital Entertainment Conference [62]. That publication is based entirely on our thesis work, with the co-author, Clark Verbrugge, providing supervisory and editing assistance.
Chapter 2

Background and Related Work

Video games are increasingly prevalent in today’s culture, especially in contexts other than leisure: many learning tools and applications use video games to achieve a multitude of goals, such as teaching new skills or using games as a different method to explain concepts [13, 45]. Studying how players interact with NPCs in such an environment can be a powerful tool, providing insight into what players might expect and how they choose to progress. Unfortunately there is a scarcity of research in modelling a player’s knowledge of game situations, and of how non-player characters move. In this section, we will give some background information on stealth games, as well as discuss other related works.

2.1 Background

Our work specifically discusses how player knowledge, understanding and expectation of NPC movements within a stealth game environment can be modeled and described. In particular, we use computational geometry algorithms to manipulate the representation of NPC paths. This section will introduce some of the concepts we use.

Stealth is one of many genres of video games that calls for players to develop and employ strategies to avoid dynamic, patrolling or static enemy agents. This style of video game typically requires the player to evade non-player enemies in order to successfully achieve a goal. Some well-known stealth games over the years include Dishonored [4], Metal Gear Solid V: The Phantom Pain [30], Thief: The Dark Project [19], Styx: Master
2.1. Background

*of Shadows* [12] and many more. Figures 2.1 and 2.2 show a screen captures from *Dishonored*, and *Styx: Master of Shadows* respectively, both depicting the player hiding either behind a wall or perched in order to observe enemies without being detected.

![Figure 2.1](image)

**Figure 2.1** Screen capture of *Dishonored* gameplay where the player must hide from the enemy and observe them.

In such games, the vast majority of NPCs tend to behave in a predetermined and predictable way. This presents a puzzle context, where (among other gameplay mechanisms) the player uses current in-game and previous observations to predict future NPC movements in order to help evade enemies or hide from them. This knowledge may come from current in-game observations, observations made in previous playthroughs, as well as from external resources and general player experience in other similar games. Accumulation of this knowledge is essential to the strategic decisions made in solving a stealth level.

For instance, players frequently use knowledge of NPCs from past gameplay or from different similar games, in order to pre-construct possible movement styles and paths that are likely to be taken by an NPC in a game. Depending on the game and the complexity of the implemented AI, gameplay may differ and NPC movement patterns will vary from game to game; nevertheless, with enough previous observation and experience, the majority
of NPCs’ movement patterns can be deduced by players. While accumulating more and more game knowledge, players may then find it easier to solve different levels as the game progresses.

In order to adhere to the aesthetic of a simulated environment, NPCs tend to move in humanoid ways. Unless specifically part of the game mechanics, NPCs generally behave in predictable ways, such as walking around obstacles, not traversing walls, being occluded by other objects, moving at predetermined speeds, etc. The path they follow can be drawn out as continuous or discretized lines or curves. To represent these paths, we use 2D coordinates to represent positions on a plane (game level) alongside a time element; we give a formal definition in Section 3.2. Representing NPC paths may seem trivial but in a game context, determining what a player observes in a particular scenario can be complex when considering obstacles. What is observable in a scenario involves occlusion and field-of-view computations; however, what a player notices, comprehends and retains of a scenario is variable and less certain. This will be further explored in Section 4.3.
2.2 Related Work

Although our novel perspective focusing upon modelling players’ understanding of NPCs and their pathing behaviour is specific and unexplored, there exists many interesting related works on researching player movement and developing adaptive AI in NPCs. Our work also involve simplifying and comparing paths, which has many applications in-game and beyond games. The following section will acknowledge some of the related works.

Pathing in Games. Path finding and planning algorithms have been explored at length [33] due to their many applications in the AI world. Typically, grid-based path finding [59] remains the preferred approach in robotics and in games; however, over the years, original search algorithms such as Dijkstra’s Algorithm [15] have been replaced and improved by more efficient ones, notably A* and its variants (which include, among others, anytime heuristic search [22] or ARA* [34]). Existing research in path finding and path modelling is primarily focused on understanding and predicting player motion from the perspective of the NPC, in order to optimize the performance of its AI. This area of research is useful for designing NPC agents that can: interact with users creatively [43]; apply basic strategy to intercept human opponents [48]; and attempt to appear “human-like” in understanding how and where a player may move [25]. As an extension outside of games, this approach has also proven useful as a means of estimating pedestrian behaviour [60]. Furthermore, while models of human motion are often based on learning AI systems, they may also incorporate other approaches, such as use of particle systems for estimating occluded position [55], and steering systems for physically motivated movement constraints [49]. Further, heuristic constraints can be derived based on looking for similarities in observed movement pattern [3], as well as gameplay-specific properties, such as a quantified notion of the “danger” or “risk” inherent to, and thus affecting different path choices [51].

Player Modelling versus NPC Modelling. Modelling the “opposite” perspective, that is what a player expects of NPC movement, is much less common, although possible with generic planning systems that do not make assumptions about human or AI agency [21]. Particle approaches and other techniques that generate probabilistic occupancy maps can be
2.2. Related Work

used for making position estimates and for modelling both point of views. The stealth game *Third Eye Crime* [26], for example, is well known for incorporating positive and negative knowledge from an AI’s perceptual system [27] into the gameplay, although again in terms of modelling the player, whilst our work focuses on the player’s perception of the NPCs. However, because players may also attribute human-like properties to NPC movement such as interpreting moving shapes as animated beings [24] or human actions [42], modelling players and modelling NPCs from a player perspective likely have some degree of synergy.

The application of more widely scoped models of player behaviour have also been explored. Knowledge of a player’s tactics or strategy, for example, allows for a user-specific AI response, whether the goal is to make a game more competitive or more entertaining. This modelling of player behaviour [6] can further enhance the performance of AI agents [32], optimizing user interactive experiences, such as adapting interactive storytelling based on player personality [14, 50]. Modelling a player’s knowledge also enables a better understanding of how to adapt to their needs, and in some cases, predict their skill level [29] in an educational game setting. This of course extends broadly, motivating a generic taxonomy [44] to help clarify the many approaches. Our approach in this terminology is a universal, descriptive reaction model interpreted through, although also grounded in, user data.

**General Path Manipulation.** Our work also touches on path simplification, using (our own) modified iterative procedure [40] to reduce points on generated paths. We use this in our work to estimate and to simplify paths to either better represent and improve understanding of actual observed movements, to avoid problems due to discretization, or to aggregate and simplify path segments – this is discussed in Section 4.2.5. Although *Ramer-Peucker* [16] is the most popular algorithm for path simplification, alternative methods such as the *Visvalingam-Whyatt* polyline simplification algorithm [54] (which simplifies paths by removing points in order of least importance) can also be applied. Other ways to manipulate paths include path smoothing using Bézier curves [58].

As for determining path similarity, in the area of computational geometry, Fréchet distance [1] is often itself used in a multitude of ways, such as a foundation to redefine curves, (ie. backbone curves [5]), to compute path similarity in a geographic context [11], or to
2.2. Related Work

use it in map-matching algorithms to track vehicle movements [56]. Discussions on the difficulty of finding faster ways to compute Fréchet distance [9] and its ability to approximate [10] remain popular topics. Fréchet is originally meant to compare continuous curves, but the modified discrete Fréchet [20] can be used to compare paths with finite numbers of vertices, and converges to the finite form as more points are added along the path. In our work, we bring on yet another variation and modification to accommodate our in-game scenarios. The Fréchet algorithm requires further modifications to take obstacles into consideration when comparing paths; we discuss this in Section 4.3.1. (Modified) Hausdorff distance [17,23], often used as a metric to compare shape similarity and to measure spatial similarity [28], can be adapted to compare paths as well. However, applied to our scenario, it is not as good as Fréchet distance since Hausdorff shape similarity set distance does not consider order of motion.

**Beyond Games.** Path prediction and reconstruction can also be applied to many real-life situations beyond video games. These diverse contexts include: security monitoring; using dynamic oriented graphs to identify moving objects and help predict abnormal behaviour [18]; as well as in observing individual animal behaviour, using state-space modelling bio-geography and spatial population dynamics to study animal movement [39]. Path modelling is also used in activity-based travel demand models, estimating where a transit user went between checkpoints [37]. Similar to our work, these concrete real-life examples seek to answer the question of where something or someone went, and are also interested in filling in the unknown gaps between (or following) known events.
Knowledge of enemy positions and movements can be acquired from various sources. Wikis and strategy guides may provide a wealth of locations and suggested routes (with widely varying levels of detail), third party game videos constitute partial sources, previous attempts by the player can provide sets of observations, and even single attempt observation can yield multiple data points while observing cyclic behaviour of NPCs.

Representing and combining this information introduces an initial challenge. Our approach is to use a uniform model that reduces knowledge of path behaviour to a series of specific, ordered and time-stamped observations independent of the source. We assume (repeated, deterministic) enemy movement is partially captured by segments of prior observation. This allows us to structure player expectation in relation to their knowledge as a process of filling in the (unknown) gaps between (known) segments, for which we can use different algorithmic solutions. The search for more constrained solutions within this space is then bounded, and the same formalism also lets us easily incorporate a variety of constraints that may heuristically limit the scope of expectation.

In this chapter we define an abstract way of modelling and representing non-player character paths, their movement, as well as player observations by means of gaps and path segments. We construct a generic model of player observations that can be accommodated to a variety of possible observation sources and expectation constraints. Then we explain how we use different approaches to fill in missing information (gaps) in character paths, as well as how these methods allow us to represent movement knowledge as accurately as
3.1 Representing Movement Knowledge

Possible. Finally, in this section we also describe our path segment generation tool and how we use it to visualize potential paths.

3.1 Representing Movement Knowledge

In this section we formalize our basic representation, building a model that incorporates both knowledge and its absence. The section following then shows how player expectation can be computed, and limited by adding assumed observations and imposing search constraints.

3.2 Definitions

First we will define the state space of the problem. We will then define paths and segments, and explain how the problem can be reduced to extracting and manipulating these paths.

3.2.1 State Space

Our game domain $\Sigma$ is a discrete two-dimensional grid, extended into a continuous, positive time dimension: $\Sigma \subseteq \mathbb{N}^2 \times \mathbb{R}^+$. NPCs follow polygonal, obstacle-avoiding routes through the grid-space, and at each point in time they have a defined orientation (field of view). We form observations of an NPC movement as a series of points that record position, time, and orientation. We will refer to elements of point $p$ using the notation $p(x)$, $p(y)$, $p(t)$, and $p(\theta)$.

Definition 1 A point $p$ is 4-tuple $\langle x, y, \theta, t \rangle$ where $x, y \in \mathbb{N}$ is the position in our 2-dimensional grid, $t \in \mathbb{R}_{\geq 0}$ indicates the time, and the angle $\theta \in (0^\circ, 90^\circ, 180^\circ, 270^\circ)$ represents the direction of the field of view, $0^\circ$ indicating the designated North.

Generally, an observation is continuous over a span of time, implying an infinite set of data points. We are, however, mainly interested in points where the observed target changes their behaviour. We thus structure path observation into segments, assuming constant motion or rotation between two points, and creating new segments when behaviour changes.
3.2. Definitions

Our model uses discrete positioning and a 4-way movement model, but it can be expanded to more complex, continuous models. Additionally, observation is not assumed to be complete, and thus a series of observed segments will have gaps between them, starting when the player loses track of their target and ending when they regain sight of the same target.

**Definition 2** A segment $s$ is a pair of points $\langle p_a, p_b \rangle$ where $p_a(t) \leq p_b(t)$.

Segments are meant to describe ordered pieces of a path, and the condition ensures the start of the segment precedes the end of the segment. We generally refer to $p_a$ as $\text{start}(s)$ and $p_b$ as $\text{end}(s)$, i.e. the beginning and end of the segment $s$ respectively.

Segments may occur in two forms, indicating either straight-line fragments of a path (path segment), or the gaps between such fragments (gap segment). The former constitute subsets of continuous observation, and for simplicity we assume that either $(p_a(x), p_a(y)) = (p_b(x), p_b(y))$, or $p_a(\theta) = p_b(\theta)$—i.e., either position changes or orientation changes, but not both. (Neither changes for a waiting agent, but time must increment.) Gap segments represent a period of missing observation that may include multiple, arbitrary motions, and are thus only constrained by advancing time.

**Definition 3** A complete path is an ordered list of $n$ consecutive path segments $\sum s_i \subseteq S = \{s_0, s_1, \cdots, s_n\}$, where $i, n \subseteq \mathbb{N}$.

The list is ordered and consecutive if and only if the end point of each path segment $i$ denoted by $\text{end}(s_i)$ is the start point of segment $i + 1$ denoted by $\text{start}(s_{i+1})$. This path may be a cycle, where $\text{end}(s_n) = \text{start}(s_0)$.

![Figure 3.1 Example of a complete path](image)

The path shown in figure 3.1 is a complete path, as it does not contain any gap segments, only consecutive path segments.
3.2. Definitions

**Definition 4** A partial path is an ordered list of path segments consisting of both path segments and gap segments.

A partial path is a valid path including at least one gap segment, starting with a path segment, and without consecutive gap segments. The final segment may only be a gap segment in the case of cyclic paths.

We require gap segments to be bracketed by path segments in order to bound the scope of non-observation. Note as well that the condition on consecutive gaps is a simplification, not a constraint, since adjacent gap segments can be trivially merged and reduced to a single gap segment.

![Figure 3.2 Example of a partial path with different ways to fill in unknown gap segments](image)

Figure 3.2 shows two (cyclic) partial paths, each with two gap segments $t_1, t_2$. The dotted lines show possible behaviours that may have occurred during the gaps. The idea is that, although the path segments of the two partial paths are identical, each gap segment can be filled in with different potential path segment(s).

In a path segment $s_i$, $p_a = \text{start}(s_i)$ denotes the starting point of $s_i$ and $p_b = \text{end}(s_i)$ denotes the end point of $s_i$.

In a gap segment $t_i$, $p_a = \text{start}(t_i) = \text{end}(s_i)$ denotes the starting point of $t_i$ and $p_b = \text{end}(t_i) = \text{start}(s_{i+1})$ denotes the end point of $t_i$.

Note that consecutive gap segments $t_j$ and $t_{j+1}$ where $\text{end}(t_j) = \text{start}(t_{j+1})$ can be reduced to a combined gap segment $t'$ where $\text{start}(t') = \text{start}(t_j)$ and $\text{end}(t') = \text{end}(t_{j+1})$. This will be our assumption from now on.

Segments can be composed into valid paths by forming ordered sequences.

**Definition 5** A valid path is an ordered list of $n \geq 1$ consecutive path segments $S = \{s_0, s_1, \ldots, s_{n-1}\}$, such that $\text{end}(s_i) = \text{start}(s_{i+1})$, $\forall i < n - 1$. A valid path may be cyclic if $\text{end}(s_{n-1}) = \text{start}(s_0)$, assuming modulo time.
3.2. Definitions

A given valid path may form a complete path, in which case all segments are path segments. We are of course more interested in partial paths, which contain gaps.

We can verify the validity of a path by insuring that its segments follow a set of constraints:

1. A partial path will contain path segments (written as p) and gap segments (i.e. g) in the form of \((pg?)^+\). This ensures that there are no consecutive gap segments in a path, and that paths always start with a path segment.

2. Complete paths will contain one or more path segments only.

3. For the \(i^{th}\) segment \(s_i\), \(end(s_i) < start(s_{i+1}) \implies end(s_i) < end(s_{i+1})\) since \(start(s_i) < end(s_i)\) by definition 2.

4. For any path segment \(s\), \(start(s) = p_a\) and \(end(s) = p_b\) can only differ in either position or in rotation, but not both. This can be represented by the conditions \(P : (p_a(x) \neq p_b(x)) \lor (p_a(y) \neq p_b(y))\) and \(Q : p_a(\theta) \neq p_b(\theta)\). The time elapsed during a segment \(\Delta t = p_b(t) - p_a(t)\) must be positive, following condition \(R : \Delta t > 0\).

To summarize, a path segment is only valid if it satisfies the condition \(\neg (P \land Q) \land R\).

Now that valid paths are defined, we will discuss how observations of NPC movements can be translated to valid complete paths, and how partial observation result in partial paths.

### 3.2.2 Player and Guards

Different games use different approaches to implementing guard movement. Unlike human players, non-player characters (NPCs) such as guards or enemies tend to behave in a cyclic way, whether it is patrolling in a repeated pattern indefinitely, or randomly selecting a move from its list of possible moves at every decision making point (i.e. choosing to turn left or turn right at a T-intersection). In our model we assume that guards can only either move forward, rotate or wait.

Guards and players are both limited by the geography of the level and cannot occupy an obstacle tile, as we will explain in section 3.3.1.
3.2. Definitions

Because a character can either move forward, rotate, or stop and wait, when moving forward, their position in $x$ or $y$ will change, and so will time $t$, but nothing else. When stopped, only time $t$ increases, and position does not change. Rotations happen instantly, either between two path segments or during a stop. Note that change in rotation only does not count as a segment. These movements observed can be translated into paths, and any missing information during observation will result in partial paths.

3.2.3 NPC Movements

Our design rests on the idea of knowledge being acquired incrementally, creating action-able expectation based upon past observation. This supposition particularly applies to stealth and combat games, where solving a puzzle made from enemy NPCs location and movement requires understanding its pattern and anticipating its future direction from observation. To achieve this end, different movement models can be used. We assume a simple model: NPCs can only either move forward at a fixed speed, rotate while standing at a single spot, or wait. Actual movement follows a simple Manhattan movement model, making discrete steps to adjacent tiles in the grid space. As previously stated, this means $(p_a(x), p_a(y)) = (p_b(x), p_b(y))$, or $p_a(\theta) = p_b(\theta)$, implying that either position changes or orientation changes, but not both. Thus, time must increment as one or none of these two positional variables change (none being when the NPC is engaging in waiting behaviour).

More complex movement models with variable speed, combining rotation and moving, and so forth are left for future work.

3.2.4 Observations

When building a path from observations made by the player, our model considers changes in rotation and position as separate path segments. Although both cannot vary simultaneously, a segment where only $s(t)$ increases is valid, as the observation made is one of the guard standing still.

In order to predict the path of a guard or to fill in the gap segment with possible path segments, we need to first reduce observations to partial paths. Then, using the knowledge
of the behaviour of the guards seen in path segments we can make predictions and attempt to reconstruct missing path segments.

Generally, an observation is continuous. We are interested in points where the observed target changes their behaviour or does something new, in our case, change in direction of movement, rotation, stopping, etc. These are the delimiters of our path segments, the start and end points. Gap segments start when players lose track of their target and ends when they regain sight of the same target.

3.3 Filling in the Gaps

Once we have a valid partial path, we can build models that estimate NPC position, filling in individual gap segments with possible path segment(s). Below we discuss different approaches, as well as describe the tool we built to visualize the results. We first explain the way we set up a level using a labelling system, then we show the conservative approach as well as variations on our algorithm to conform to expectations of NPC movement patterns. Some of these beliefs may stem from observing past NPC behaviour, obstacles, level geometry (including interesting sites), etc.

3.3.1 Level Setup

Figure 3.3 Example of a level setup where tiles \{0,1\}, \{0,2\} and \{2,3\} are obstacles.
3.3. Filling in the Gaps

The basic idea of tiling a level is to overlay the geometry of the level onto a grid with a width and length. Each tile will have a unique label based on its position in relation to the origin (top left corner). Then, from there, all reachable tiles are labelled as reachable, and all tiles that contain walls or obstacles are marked as obstacles, and are therefore unreachable. Figure 3.3 shows an example of a simple level setup where the dark tiles represent unreachable tiles, e.g. obstacles.

Each tile has a set of neighbouring tiles. These neighbouring tiles indicate the tiles where a character may travel next. This requires all of our algorithms to first make a labelling pass, thereby removing neighbouring tiles that are obstacles: since they are not reachable, they should not be included when searching for potential valid paths. We can easily compute the distance of two tiles \(a\) and \(b\) based on their coordinates by the formula \(|a(x) - b(x)| + |a(y) - b(y)|\). The distance is used as part of our calculations of shortest and most likely paths that we will describe next, and it is the basis of most of our algorithmic approaches.

3.3.2 Conservative Approach

Our baseline approach gives a full representation of every position the NPC could have occupied. We iterate through each gap segment \(g_i\) in the partial path. Given that the amount of time elapsed \(\Delta t(g_i)\), and naïvely assuming that the NPC did not vary in displacement speed, we can calculate all potential positions the guard may have occupied between, and for how long. In general, if \(\Delta t\) is greater than the minimum time to go from \(p_a\) to \(p_b\) then, we can only delimit where the guard may have been within specific time periods.

The general idea of the algorithm is that given a starting point \(start(g_i)\) and an end point \(end(g_i)\), which describe the beginning and end of a gap segment, we look at the time elapsed \(\Delta t\) to calculate a general area where the target could have been within this time. Since the target was only seen at the start and end points, and there were no indications of where they could have gone in-between these two points, then based solely on the geography of the level, we can eliminate areas of the level where it would have taken too long to reach, causing it to be impossible to find the character in those areas between the two points when the character was seen. We can also eliminate areas blocked off by obstacles.
3.3. Filling in the Gaps

Our method associates each position with labels, one representing the earliest time an NPC may arrive at that location having departed from the start of the gap segment, and one for the latest time they must leave it in order to reach the end point of the gap segment.

The labelling algorithm is essentially a BFS performed on both the beginning and end points separately. We flood out from the starting point $start(g_i)$, computing for each tile a value $l_a$ as the minimum time the tile could be entered. Then we perform the same procedure from $end(g_i)$, computing $l_b$ as the minimum time required to reach the end. For a given tile $\tau$, we can define $w_\tau = \Delta t(g_i) - (l_a + l_b)$, which gives the maximum time an NPC can possibly wait at that position before continuing if they must reach $end(g_i)$ within $\Delta t(g_i)$. By sorting the tiles by their $w_\tau$ value and colour-coding them, we can see a heat-map effect. Figure 3.4 shows an example.

In this figure we have two endpoints of a gap segment, shown as black dots with a red circle: a guard was observed moving upward on the left side of the map from tile $[2,5]$ to tile $[0,3]$, and then 7 time units later, the same guard was observed moving upward from the obstacle on the right. Our goal is to solve where the guard went during the gap of 7 time units ($\Delta t = 7$). In the image, $p_1$ is the first observed segment, and $p_2$ is the following known segment. The figure also shows the obstacle tiles in grey.

Our initial labelling system results in a topographical representation of the game space
3.3. Filling in the Gaps

where each calculation allows us to narrow down the possible positions of a given character for one specific gap segment. We can integrate this labelling representation into our approach to reconstruct NPC paths, thus using player observed limitations to constrain possible paths for each separate gap segment.

This topographic depiction has the advantage of compactly representing reachable locations, while still enabling us to recover the exponential [31] set of feasible paths by conducting a search from start to end of the gap segment, entering a neighbouring tile $\tau$ only if it satisfies $\Delta t - t_e - l_b > 0$, where $t_e$ is the time taken on the path so far. This is an easy way to calculate a node on a feasible path because any tile that requires less time to reach end point ($l_b$). Note that this computes minimal paths when $\Delta t = \Delta t_{min}$. Additionally, $\Delta t_{min} = \tau_{start} \cdot l_b = \tau_{end} \cdot l_a$, which can also be interpreted as the distance between the beginning and end of the gap segment.

However, this leaves us with increasing uncertainty, as the solution set gets exponentially bigger as $\Delta t$ grows larger. This is because using the labelling algorithm, we can only estimate and calculate all the possible locations the target could have travelled, and at what time, but we cannot narrow it down to specific paths. As the elapsed time grows increasingly longer, the target could virtually have gone anywhere.

![Figure 3.5 Resulting search space based on increasing the amount of time elapsed without changing the position of start or end of a gap segment.](image)

Figure 3.5 shows an example of the search space for different $\Delta t$, using the 3rd dimension as time. We used blue blocks to show the additional reachable tiles when incrementing
3.3. Filling in the Gaps

$\Delta t$. Grey boxes represent unreachable spaces (obstacles) and red boxes represent the reachable tiles at $\Delta t - 1$ (i.e. what was available before increasing $\Delta t$) to better show what has been added.

To illustrate this idea, consider Figure 3.6, an example of a gap segment that has only one possible path between two path segments $p_1$ and $p_2$. Since we know the amount of time that has elapsed ($\Delta t$) between end($p_1$) and start($p_2$), we can visualize the problem in 3 dimensions where the height is the element of time. If the target can travel at most one space per time unit, we can attempt to fill the gap by using blocks to represent possible paths.

Figure 3.6 Top and isometric view of filling in a gap segment

In this particular example, the start and end of the gap segment are set to 3 tiles away, as $\Delta t = 3$. By keeping the same initial and final positions constant while increasing time, e.g. $p_2(t)$ to 4 and 5, it can be demonstrated that with each added layer, exponentially more blocks appear to show different areas the observed target may have visited at a particular instance in time. If we took one level of the graph at a time, such as at $t = 1$ then we can see that on that plane, all the possible highlighted areas the target could have been located at that time. However, this algorithm does not allow us to view the possible distinct paths.

An example of different time slices is shown in Figure 3.7. The grey tiles are obstacles, e.g. tiles that can never be reached. The clear blue tiles indicate where the character may
3.3. Filling in the Gaps

be at that time slice. (For example, at \( t = 0 \), the character can only be at one possible tile because it was last seen there.) Then, on the next time slice, arrows indicate where the character could have gone next. Hashed tiles are possible places where the character may have stopped. From the different states of the map of each time slice, notice that some tiles are technically reachable but not coloured in because of constraints such as not being able to reach the end tile of the gap segment in time.

Finally, Algorithm 1 below shows the labelling method used in our conservative algorithm in pseudocode.

Algorithm 1 Labelling tiles in Conservative Approach

Require: All unmarked tile.label \( \leftarrow -1 \)

1: \textbf{procedure} LABEL(tilelist, l)
2: \hspace{1em} nlist \( \leftarrow \) null \hspace{2em} \Comment{Create empty queue}
3: \hspace{1em} \textbf{while} tilelist.COUNT \( > 0 \) \textbf{do}
4: \hspace{2em} \hspace{1em} t \( \leftarrow \) tilelist.DEQUEUE \hspace{2em} \Comment{Get first tile in the queue}
5: \hspace{2em} \hspace{1em} \textbf{for all} n \textbf{in} t.neighbours \textbf{do} \hspace{2em} \Comment{Label all the neighbouring tiles}
6: \hspace{2em} \hspace{2em} \hspace{1em} \textbf{if} n.label < 0 \& n \neq \text{obstacle} \textbf{then} \hspace{2em} \Comment{Neighbour has yet to be labelled}
7: \hspace{2em} \hspace{2em} \hspace{2em} \hspace{1em} n.label \( \leftarrow \) l + 1
8: \hspace{2em} \hspace{2em} \hspace{1em} nlist.ENQUEUE(n)
9: \hspace{1em} \hspace{1em} l \( \leftarrow \) l + 1.
10: \hspace{1em} \hspace{1em} \textbf{if} nlist.COUNT \( > 0 \) \textbf{then} LABEL(nlist, l)
11: \textbf{return}

3.3.3 Obstacles and Choke-points

If we think of the different combination of paths possible in a grid, where the only valid steps taken are ones in the direction of the goal, we can think of it as a combinatorial problem where each variation of the path is a different ordering of the necessary steps required to get to the goal. This will naturally eliminate paths that are unfavourable because they take unnecessary “detours” or steps away from the goal.

In a situation where there are no obstacles, it is easy to see that it does not matter the order in which the character performs these UP or RIGHT moves, as long as they perform the right number of each. In a simple grid with no obstacle, seen on the left in Figure 3.8,
we can see three different sample paths that fill out the gap between the start and end points given a time budget of 5 steps. Since the end point is three tiles to the right and two tiles above the starting point, there are only two possible moves a character may perform according to our algorithm: UP or RIGHT. More precisely, the character must perform 5 steps, and exactly 3 of the steps are towards the right and two steps are upwards, in order to completely reach the end point of the gap without backtracking or stopping.

For shorthand, \( R \) will indicate one step towards the right, which increments the character’s current \( x \) position by 1, and \( U \) will indicate one step upwards, incrementing \( y \) position by 1. In the illustrated example without obstacles, we have \( \text{UURRR, RURRU} \) and \( \text{RRRUU} \), three of many possibilities.

Now to derive the generalized formula to calculate how many possible good path segments that can be found, we can use a simple combination formula of \( n \) choose \( k \) where
3.3. Filling in the Gaps

Figure 3.8 Permutation zones shown on maps with and without obstacles, and how they affect paths found within the level.

$k$ is the number of $R$ or $U$ required to reach the goal and $n$ is the total number of moves available.

$$\binom{n}{k} = \frac{n!}{k!(n-k)!} \text{ where } n \geq k$$

The formula is obtained as following. You have $n$ moves, and suppose we split $n$ into two components: $n_R$ $R$ moves as required, and $n_U$ $U$ moves as required, and without loss of generality assume $n_R \geq n_U$. A path is constructed by choosing $n_U$ of our $n$ moves to be $U$ moves and the rest as $R$ moves.

In our example in Figure 3.8 without obstacles, we would obtain $\binom{5}{2} = 10$ different paths a character can take. Now, observing the same scenario with obstacles, we notice that some of the paths options are no longer possibilities. We can approach this by eliminating the paths that were using the tiles that are in fact obstacles. This will require us to traverse all 10 paths once more and only keep a subset of paths. Alternatively, we can approach the problem from a sub-problem perspective. In the example with obstacles in the same Figure 3.8, we notice that all paths must first arrive to the tile that is located one above and two to the right of the start tile, before reaching the end tile. There, we can place a pseudo-end tile, and similarly calculate that there are $\binom{3}{1} = 3$ different possible permutations of the steps to reach the pseudo-end, and since there are no other variations, we can safely say that there are only 3 possible goal-oriented paths.

To illustrate a more complex scenario, we can generate permutation zones defined between any two points, as sometimes there may be more than one such as in Figure 3.9. In
3.4. Conforming to Expectations

In most games, stealth games in particular, NPCs move in constrained, structured ways. Guards, for example, are expected to patrol an area, moving efficiently between locations, perhaps with pauses to look around, but with limited backtracking. Figure 3.10 shows an example of a guard waiting at an interest point taken from actual gameplay footage of the game Dishonored [4], where the NPC pauses in front of a painting, admiring it.
3.4. Conforming to Expectations

**Figure 3.10** Circled in red is a guard seen paused (waiting) in front of an interest point (painting).

The set of paths and positions a player may expect an NPC to traverse is thus less than the set of reachable positions computed in the baseline conservative approach. We want to narrow down the possibilities to paths that are more likely, based on how a player may expect an NPC to actually move. For example, although it is technically possible that an NPC doubles back over a set of tiles, it is unlikely that they would do so, because NPCs are not generally implemented to move randomly: instead, they tend to be implemented following scripted movement paths, or as goal-directed behaviour.

There are a number of different factors that can influence a player’s expectations of possible NPC movement. Optimality is an obvious criterion—given two points, a rational expectation is that the NPC followed a minimal path. When the $\Delta t$ exceeds $\Delta t_{\min}$, however, the extra time must accounted for either waiting, in detours or loops, or both. Other factors, such as number of turns, proximity to obstacles or other narratively interesting content, may also be considerations.

Figure 3.11 shows a small example. In this scenario, the two blue paths follow minimal routes and must incorporate some amount of waiting, with the dark blue path also including more rotations. The red path follows a more circuitous route, trading an assumption of waiting for sub-optimal routing. Given all the possible paths between the start and end of a gap segment $g$, we tend to favour solution paths $q$ where $\text{length}(q) = \Delta t_{\min}(g)$. In this scenario, the two highlighted (blue) paths are equivalent in length (duration), however
3.4. Conforming to Expectations

**Figure 3.11** Example of a gap with $\Delta t = 8$. The two blue paths are equally minimal in length (4), but imply the NPC paused at some point to make up the 4 extra time units. The dotted red path is arguably less realistic in being indirect, even if it implies continuous movement. Turning here is assumed instantaneous.

...they have different number of turns within the gap segment (dark blue being four, light blue being two — one within the segment and one at the beginning of the gap segment). Still, the blue routes are considered more favourable than the red dotted one, because they are more direct (e.g. they both qualify as shortest path). Arguably, the light blue path (2 changes in direction) is preferable to the dark blue one (4 changes in direction).

To better predict the path taken by a NPC, we want to look at a selection of possible paths. As seen in the conservative approach, the larger the difference between $\Delta t$ and $\Delta t_{\text{min}}$, the more paths there are to choose from, since for every unit of time added, more possibilities emerge. Ideally, we can search the solution space for paths that are simple first, with a baseline assumption that players expect NPC movements are unlikely to be sporadic, irregular or random.

The following sections will present different factors that can affect a player’s expectations of NPC movements, such as assumptions after repeated observations, patrolling styles (e.g. simple paths, waiting), interesting sites, and level geometry.

### 3.4.1 Repeated Observations of an NPC

Players often repeat the same part of a level, whether it is caused by reloaded after failing a game mission, or revisiting an area in a later time. The player may be aware and remember
3.4. Conforming to Expectations

the movement pattern of a specific NPC in game, and they may expect NPCs to retain the same patrolling path throughout a single level to meet their consistency expectations.

When a player repeats a section of a game, new observations can be translated into path segments that can be added to an already known partial path and therefore some path segments may overlap. Multiple observed instances can be reduced to segments that are part of the same partial path, which can be updated to include newly acquired knowledge. In Figure 3.12, we show two separate observations of the same NPC. The first observation may lead the player to believe that the NPC took a short and direct path during the gap segment. However, a second observation either later during the gameplay or upon repeating the level, the player observes different segments of the NPC’s path. This provides new information on the original path, and will update the player’s knowledge on that NPC’s movement pattern. Instead of treating the two observations as separate paths and separate patterns, we combine them to produce a single partial paths with three segments (as two path segments are overlapping). The challenge is to determine if (the player recognizes that) observations are part of one or separate complete paths, but will will discuss the difficulties further in Section 4.2.

![Figure 3.12 Example of two separate observations contributing to the knowledge of the same NPC’s movement pattern.](image)

When we consider the goal or purpose of a NPC such as a guard, we realize that it is more likely that the guard would patrol in a pattern that maximizes efficiency and minimizes repetitive and unnecessary movements. Although there are a multitude of ways to categorize NPC movement patterns, in our work we identify guard paths using the follow-
3.4. Conforming to Expectations

...ing characteristics: simple path or looping, with or without waiting, and whether it includes interest points. We expand on these characteristics below and explain how they conform to player expectations, as well as how they affect the way we generate expected paths.

3.4.2 Simple Paths

In the conservative approach, we considered all possible positions where a guard may have travelled in the span of time between when they were last seen and when they were seen again. Although it is possible that the subject in question has doubled back over a set of tiles it is unlikely that they do. When we consider the goal or purpose of a guard, we realize that it is more likely that the guard would patrol in a pattern that maximizes efficiency and minimizes repetitive and unnecessary movements. This is why not only do we favor shortest paths, but we also prefer ideal paths.

**Definition 6** An ideal simple path is a valid, complete path connecting the start and end tiles involving no backtracking, and is constructed using the least number of path segments possible.

The ideal simple path does not involve doubling back or revisiting tiles multiple times. We augment minimal distance with a requirement for a minimal number of turns as well—“straight” paths are intuitively simpler. The definition of straight of course depends on the movement model. In our Manhattan-based model, we have many equivalent paths in terms of distance, that may nevertheless be seen as more or less simple depending on how jagged, or turn-intensive they are.

This gives us two properties we can use to reduce the set of expected paths (and thus positions), either minimizing distance, or number of segments of constant movement. Our interpretation of (ideal) simple paths combines these, as the subset of minimal distance paths that are also constructed from as few path segments as possible.

As previously mentioned, the set of minimal distance paths is easy to compute in our representation by assuming $\Delta t = \Delta t_{\text{min}}$. For reasonable gap sizes and obstacle density, reducing that set to minimize segments is also straightforward.
3.4. Conforming to Expectations

3.4.3 Waiting

Other constraints on expected pathing behaviour can be based on similar properties. Pausing or waiting is a potential movement behaviour, common in many stealth contexts. Player expectation may thus also be informed by the existence or distribution of excess travel time. If NPCs do not generally pause (or vary speed), then a gap segment must be assumed to include detours of some form, and overly quick paths will be excluded from the expectation set. Path search in this context cannot terminate unless the time used matches the gap time: \( t_e = \Delta t \). In other words, the last time frame \( t_e \) should correspond to the amount of time elapsed \( \Delta t \), so that the time elapsed equates the length of the gap segment. In our model, when waiting is not allowed, time can be used as measure for distance travelled, and time and length can be treated as the same units, but it is not always the case, i.e. when waiting is allowed.

**Definition 7** A path includes waiting when the total time elapsed \( \Delta t \) exceeds length\((q_i)\) for all possible valid solution paths \(q_i\). A path segment \( p \) denotes waiting when \( p_a(\theta) = p_b(\theta) \) and \((p_a(x), p_a(y)) = (p_b(x), p_b(y))\).

When waiting is a possibility, the distribution of pauses becomes a factor—a character that repeatedly stops and starts is less likely than one that waits less frequently, and for a longer period of time. Tiles that can be part of paths with a single waiting event of total duration \( w \) are trivially found as ones for which \( w = \Delta t - (l_a + l_b) \), and since (maximum) waiting time is part of our representation we can also easily incorporate different allocations of waiting into a path search.

When \( \Delta t \) is relatively close to \( t_{\text{min}} \), and waiting is part of the NPC’s behaviour, then it is possible that an NPC waits at an interest point along a short or simple path. However, if \( \Delta t \) is significantly larger, then longer periods of wait time are unlikely, even distributed along its path. How much waiting is reasonable of course may depend on the specific game context.
3.4. Conforming to Expectations

3.4.4 Interesting Sites

A further source of expectation exists in geometric, or content-driven properties of the game space. An NPC guard in a museum may be expected to pass by specific artwork, or frequently return to a chair or guard station. Generally, in a game, interesting sites are key areas that involve pivotal game play, or player decisions. Depending on the narratives, these are spaces designated by the game developer, and we use them to further predict how a player may expect an NPC to move.

**Definition 8** An interesting site is a manually assigned or automatically generated tile $\tau_i$ in $I$, where an NPC is most likely to wait, or choose to include in its path.

Interest points may be incorporated by introducing an artificial path segment, locating the NPC at an interesting site at some time within the gap segment. The time values (and orientation) are less determined, but can be bounded, since for a given tile in the interesting area arrival and departure must be between $l_a$ and $\Delta t - l_b$ respectively.

Interesting sites are clearly game-dependent, and are best manually identified. In our implementation, when no interest point is defined the algorithm assigns corners and tiles around obstacles to be points of interest, as heuristically important occlusion and observation points in a stealth context.

To calculate a path that favours interest points, we first calculate all reachable interest point tiles $\tau_i$ with $\Delta t - (l_a + l_b) > 0$, then we simply consider paths that include interest points where this value is largest, therefore using up time most efficiently.

In the example shown in figure 3.13, we try to fill in a plausible path for the gap between observed path segments $p_1$ and $p_2$, with a duration of $\Delta t = 9$ in this case. Because $t_{\text{min}} = 3$ and is significantly smaller than $\Delta t$, if waiting has not been previously observed then a shortest path may not be plausible, and we would expect a player to consider longer paths that do not require wait time. The paths shown in cases A and B both fulfill this requirement. The latter, however, both explores more interesting area (the lone obstacle), and has fewer turns, arguably making it more probable. This goes hand-in-hand with the simple path characteristic we mentioned above.
3.4. Conforming to Expectations

3.4.5 Level Geometry

A player’s awareness of obstacles also affects their expectation of pathing possibilities. This is easily incorporated into our approach by relaxing the base level representation—gaps in geometric knowledge reduce path constraints, expanding the underlying conservative set of (expected possible) reachable positions.

Some unreachable areas are more subtle than others. An obvious example is when a player observes the NPC enter an alcove. Based on the previous knowledge of it being a dead-end, the player may assume that the NPC cannot reach other spaces in a reasonable amount of time, until they’ve exited the alcove. The example in Figure 3.14, shows tiles that are physically unreachable because of time constraints under the chosen movement models. The light-grey tiles are ones that cannot be reached from the start tile if the end tile requires to be reached within \( \Delta t = 4 \), and the crossed-out tiles are a subset of those tiles that cannot be reached from the start tile within those time steps. Tiles that are unreachable from the starting tile are naturally included in the set of the tiles that are eliminated based on the ending position of the gap segment. These limitations, although obvious to compute, are difficult to pick up by a human player, and are less useful at eliminating possible path segments when determining where the NPC went.

In our case, our models are based on players knowing the entirety of the level’s geom-
3.4. Conforming to Expectations

![Diagram of a game map with inaccessible tiles and constraints](image)

**Figure 3.14** A screenshot of an example with unreachable tiles, based on the starting and ending positions of the gap segment.

etry. More advanced approaches that would model the partial knowledge of the level and expected geometry could also be integrated, at the cost of significant additional complexity, and could be integrated as future work.

### 3.4.6 Negative Knowledge

*Negative knowledge* can be modelled similarly to path segment knowledge. For instance, a player that does not have an NPC in sight is making observations of where the NPC is not. These observations can be translated into negative segments.

**Definition 9** A negative segment $\gamma \in \Gamma$ is a list of points $\langle p_a, p_b, ..., p_n \rangle$ where $p_a(t) \leq p_n(t)$.

In other words, $\Gamma$ is a set of observation points taken from after the end of path segment until the start of the next path segment. These observation points happen during gap segments, e.g. when and where the NPC in question was *not* observed.

This negative segment also represents the negative knowledge and allows us to further eliminate paths that coincide with these segments by only accepting paths that do not include any $s_i$ where $p_i \in \gamma$ and $p_i \in s_i$ for each $s_i \in S$ and for each $\gamma \in \Gamma$.

Evidently, this requires taking into consideration field of view and estimating which tiles the player may have seen during the gap segments, and can prove to be challenging. The location of obstacles and how a level is designed greatly affects the number of tiles
3.4. Conforming to Expectations

that can be seen at once, in any direction, and the complexity can be too high to calculate accurately using individual points.

However, we can use our conservative approach and treat each $γ$ as a path segment where two points $p_1, p_2$ (selected using $p_1 = \min[p_i(t)]$ and $p_2 = \max[p_i(t)]$) can be treated as the start and end points of the segment. Then we simply run our labelling algorithm and create a negative cloud of the places where the NPC could not have been based on the observation and eliminate this cloud from the initial computation before searching for potential paths.

In our research, we assumed that the player has full knowledge of the space, but in a game where the first-person view provides only partial knowledge of the space at any given moment, more uncertainties will arise. A player that has not yet explored a space will find it more difficult to predict the path of an NPC once it disappears from sight. Similarly, if the player does not recall most of the level or remembers the geometry incorrectly, they can only make fuzzy predictions of the NPC’s future movements. When collecting player data, modeling the environment in real-time, and adding obstacles as the player encounters them, may provide a more accurate idea of what the player may know of the space, and hence of where the player may expect an NPC to go.

A player’s perception of the space tend to be skewed and inaccurate, but measurement of player’s perception and mental representation of the space is left for future work, because in the end, negative knowledge remains unbounded and therefore non-trivial to integrate into our current work.

3.4.7 Summary

Guards in games may in theory move in different, arbitrary ways within the level. However we found that there are many factors that can narrow down the possibilities based on characteristics of typical guard behaviour and movement limitations that conforms to general player expectations.

First, by assuming that guards always take shortest paths, we can eliminate paths that are (although possible) highly unlikely. The assumption of guards always taking the shortest path works well when the player has previously observed the guard engage in waiting.
3.4. Conforming to Expectations

behaviour. However, when waiting is absent, a more sensible solution is to find paths that use up elapsed time efficiently: paths that include maximum wait times $w_{\text{max}}$ are less probably paths. In these cases, we are looking for simple paths and ones with the least wait times. This different approach will allow paths that are not necessarily the shortest, but are justifiably valid as patrolling paths. The longer (and not necessarily direct) path is often one that include interest points, which justifies why shortest path is not used. Nevertheless, interest points can still be used in short paths that include waiting as it is realistic to assume guards tend to wait at interest points.

Ideally, we aim to combine all the approaches to form a model that will take in any partial path and return a set of most likely complete paths. In order to achieve this goal, we need to define what properties are in play and how influential these properties are. For example, some of the key properties include: simplicity of path, redundancy and cyclic behaviour, minimizing wait time, shortest path. These properties are all different approaches in selecting a subset of paths that can fill in any gap segment.

By analyzing previously seen path segments, we can determine whether a player may have observed that an NPC has tendency to take simple paths, interesting paths, or wait, etc. This allows us to focus on specific properties when generating segments used to fill in gap segments.

Based on previous observations of NPC movement being generally repetitive and cyclic, a movement pattern that was observed can be used to assume future movements and decisions made by the NPC. This knowledge is gained either in the same level, in a previous playthrough or from other games, then based on those observations, they may notice that guards tend to take shortest paths, pinpoint interesting sites, or learn about the level’s geometry, which in turn help them predict where the guard will go next.

In the following section, we describe the tool that is used to generate potential paths of the different factors we discussed, namely shortest paths, simple paths, ones with waiting, and ones that include interest points. Although our setup naturally takes into consideration level geometry, the (partial) knowledge of it, repeated observations, and negative knowledge are left for future work.
3.5 Path Segment Generation Tool

The geometry, set of guard motions, and expected player entry and observation points affect how a player will approach a game stealth puzzle. In order to understand the effects of different observations we created a tool in Unity®, which allows us to construct simple game levels and visualize the conservative results. This also allows us to filter and generate example paths segments that respect our various constraints.

3.5.1 Tool Usage and Features

Usage of our tool requires basic knowledge of Unity®, and allows the user to build levels and generate paths inside them. The tool expects a discretized level as input, indicating which tiles represent pathable space and which represent obstacles; for easy experimentation tile designations can also be interactively modified. Within this space one can specify a gap segment within a fully customized level. The level is dynamically filled with coloured columns indicating reachable tiles based on our conservative algorithm, forming a heatmap of possible positions according to maximum waiting time. Using selectable options, the user can generate likely paths based on different path characteristics.

Our tool allows the user to explore the generated paths using two different views: a top-down perspective and a rotating 3D view, from which the user can visualize the length and wait time possible at every tile as a column in the time dimension rendered above the level space. In this section, we describe the tool’s main features and how to use them, and the following section will focus on how to interpret the generated data.

Getting Started. Upon launching the tool, an empty map is generated if no imputed map is detected. The user can click on individual tiles to toggle its state (obstacle or not), and once satisfied with the representation of the level on the map, the user can choose to save the map by navigating to the editor and copying the group of tiles and pasting them once the tool is in edit mode. This enables the user to store multiple maps and set the group of tiles to be “active” in the editor to load that previously created map.

Not only does the interface of the tool allows the user to set obstacles in the map, the
3.5. Path Segment Generation Tool

user must set the start and end points of a gap segment. The $\Delta t$ for the gap can also be adjusted using the top left slider. After clicking the “Regenerate (R)” button, the tool will provide three possible paths that adopt the following algorithms: shortest path (green), exact steps (blue) and most interesting path (red). The path generators were implemented based on our algorithms mentioned in Chapter 3. Each one of the generated paths (as well as wait time) can be individually toggled on/off using checkboxes.

The user can then choose to display the heatmap via an overlay generated by our labelling algorithm, which shows where the NPC can possibly go given the chosen parameters. This can be better viewed in the 3D rotating view where the user can visualize the length and wait time possible at every tile.

**Generating paths.** Within the map, to generate potential paths, the user must first designate start and end points on non-obstacle tiles by pressing “1” and “2” respectively on the keyboard over the two selected tiles. The start tile will now be shown with a green square, and end tile with a blue square. If they wish, the user can also adjust the $\Delta t$ using the slider labelled “Steps/Time” in the top left corner. The user can now generate paths using these settings by clicking on the “Regenerate (R)” button, or by pressing “R” on the keyboard.

As seen in Figure 3.15, the user can select which style of paths they want to generate using the checkboxes. These paths with different properties are generated using variations of the conservative approach described in Section 3.3.2. The three sets of paths that can be generated and displayed include:

1. Shortest Paths, with or without the property of:
   - Include waiting;
   - Simple paths only, as per Section 3.4.2;

2. Paths with exact steps without waiting, as discussed in Section 3.4.3;

3. Interesting paths, with or without interest points, explained in Section 3.4.4.

Individual constraints can then be selected using checkboxes, and a path respecting each resulting property can then be generated as example routes. By default, a shortest
3.5. Path Segment Generation Tool

![Path Segment Generation Tool Interface](image)

**Figure 3.15** Buttons, checkboxes, slider and Δt\textsubscript{min} labelled in a partial screen capture of the tool interface for clarity.

A path will be displayed in green, a no-wait path in blue, and an interesting path in red. In the case of multiple valid paths a representative path will be displayed. Other constraints are straightforward to integrate.

The user may choose to generate simple shortest paths only (an ideal simple path can be generated by selecting the “simple” checkbox), and select whether they want to add waiting to those paths. In this case, the user must regenerate again upon changing properties.

An additional functionality is the ability to manually create paths. The user may choose to input a “manual” path inside the tool within the gap, i.e. this manually defined path is the “actual” known path taken. This allows the user to compare generated paths to real-game NPC paths. The tool will then use the 2D distance and a modified version of the Fréchet distance \cite{1} to compare the accuracy of the generated path sets. This is further detailed in chapter 4. Users may hold “M” while selecting tiles that are part of the manual path, use “N” to restart, or use “B” to run the modified Fréchet Algorithm, which will then display the average distance, maximum distance for each set of generated paths in the Unity\textsuperscript{®} console.
3.5. Path Segment Generation Tool

3.5.2 Viewing Paths and Interpreting Results

Essentially, the tool allows for exploration of possible path estimations in different ways. One is through the conservative view, presented as a heatmap and 3D view. The other is through integration of the different constraints (shortest etc) to give a sampled “most likely” path. Figure 3.16 shows a complete screenshot of the interface. The figures in this chapter show paths of different colors that may have coloration gradients: this is to show the path’s direction which is created via the way lighting renders certain 3D lines in 2D settings.

![Figure 3.16](image)

**Figure 3.16** A screenshot of the tool built to analyze gap segments. The red, green and blue lines represent generated paths of different styles. Green square is the beginning of gap segment whereas blue square is the end of the gap segment. White dot is used to animate the simulation of the enemy or guard taking the paths.

Figure 3.17 shows examples of three different generated paths for the same gap segment (green for shortest, blue for no-waiting, and red for interesting). These paths can be viewed in 2D (left) or in 3D (right). The colouring of the vertical columns in the 3D view indicates when the character can enter the tile beneath it, while the height of each of the columns is calculated by \( \Delta t - (\tau \cdot l_a + \tau \cdot l_b) \).

Here’s an overview of the buttons available for path exploration, and their functionality:

1. The user can use the “Toggle View” button to switch between the top-down view and 3D view.
3.5. Path Segment Generation Tool

![Figure 3.17](image)

**Figure 3.17** Two separate views: top-down and 3D-rotating side view of paths generated using our tool for the gap segment indicated by the green and blue squares.

2. Additionally, they can choose to “Hide Heatmap”, which allows for a clearer view of the level and paths.

3. “Hide Paths” allows the user to conceal the generated paths, which enables them to simulate NPC movement without giving away the path it is on. We use this in Chapter 4 when creating scenarios for our user survey.

4. “Start Walk/Cancel Walk” button shows an NPC (represented by an animated white dot) follow the selected path (using “Short”, “NoWait”, “Int” buttons) at the selected speed (adjustable using the “Walk Speed” slider). This can be used whether the paths are on display or not. Although we can simulate the “walk” at different speeds, we do not consider speed as a variant in our algorithms.

Other useful information displayed include $\Delta t_{\text{min}}$, i.e. minimum time step required for an NPC to reach from start tile to end tile. Additionally, the user can hover their cursor on any tile and the tile’s information will be reflected on the left such as the tile position and label values.
3.5. Path Segment Generation Tool

Even more exploration can be done using the “Scene” view in Unity\textsuperscript{®}. For example, the colored time columns forming the heat map are grouped by short, medium and long. The user can dynamically hide one or many of the columns by setting the group of GameObject as inactive.

**Tool Usage Examples.** Figure 3.18 shows a slightly larger example with three generated paths, following different constraints. Within the visualization we can animate characters following any of the paths generated, allowing us to see relative differences in movement. We used this visualization as a basis for the different examples shown to users in our human study, in chapter 4. The tool’s goal is to allow its users to build levels and explore potential possibly predictable NPC movements. This may be especially useful to game designers that want to generate NPCs that move and follow a typical pathing style that respect distinct constraints.

![Figure 3.18 Example of shortest simple path, exact steps and interesting paths generated by our tool on a larger map, with $\Delta t = 26$, manually defined segments and a hidden heatmap.](image)

**3.5.3 Future Directions and Tool Potential Expansion**

The tool is best used to illustrate a single gap segment: the paths that are generated for that gap segment can be clearly visualized. When incorporating multiple gaps into a single
3.5. Path Segment Generation Tool

map, we would require multiple layers of labelling and potentially overlapping heat maps, and visualization can become confusing and messy, which can be seen in Figure 3.19. If desired, a solution to this would be to iterate through all gaps and run the algorithm and visualize the generated paths and heat maps one gap at a time.

Figure 3.19 Representation of two gap segments on the same map. Overlaying heat maps makes it difficult to distinguish the computed path information between the two gaps.

Another limitation of the tool is that it does not allow speed variance in guard movement, other than waiting. Although the tool can simulate the movement of NPCs at various speed, the tool does “not allow speed” as a factor in its actual calculations. This is generally not a problem since NPCs tend to move at a constant speed. Also note that maps are not always easily represented on a Manhattan-movement grid and thus accurately translating movement from a more complex map is difficult. This is further discussed in Section 4.4.
Chapter 4
Experiments

Thus far, we have detailed how our method allows us for thorough exploration of players’ understanding of NPC movements through an abstract representation visualized in 2D or 3D by our tool. To support our approach to modelling player knowledge, we conducted related experiments to further understand what players expect of non-player characters’ movement, and also tested our methods in a recreation of guard movements from real game levels.

In this chapter, we discuss how we created a small user study with the intention of investigating whether players hold a preference between different NPC movement models and/or behaviour. We explain the threats to validity during path generation and observation representation that we encountered while emulating what a player may know of an NPC. The following is but a sample of variables that may affect the outcome and accuracy of generating paths: gap complexity, negative space, and fuzziness. We also touch on difficulties in translating certain movements and paths using our model, and how we counter those complications. Furthermore, we used artificial scenarios to construct different paths and dropped out segments to emulate partial observations a player may experience, and ran series of tests to conclude that simple shortest paths best represent actual paths. Finally, we use levels from the games Dishonored [4] and Aragami [35], and applied our path generation algorithm on observations of enemies in both respective games. Other games such as Pillars of Eternity [38] and Thief [19] are used to make observations on guard movement patterns in stealth games.
4.1 User Study

First, we validated our choice of constraints on path expectation through a simple user study. Using the tool that we built, we generated different paths with distinct features in various level setups and created video recordings to simulate an NPC taking these generated paths. We removed the NPC from view during gap segments and asked survey participants to choose which path the NPC had likely taken during those segments.

![Figure 4.1](image)

Figure 4.1 A screen capture of a question in the survey.

4.1.1 Survey Format

Our survey consisted of 10 questions. Each question presented an option between two generated paths that could have filled the gap segment presented, as seen in Figure 4.1. The two options presented in each question differed according to the following characteristics:
4.1. User Study

short/long, simple/complex, interesting, with cycles, with backtracking, with or without obstacles, etc.

In order to simulate NPC movement, we animated each scenario by using our tool to trace out the generated paths. The NPC is hidden during the gap segment in the scenario. Naturally, we also omitted the heat map and actual generated paths in order to not bias the user.

We created different levels with varying complexity involving paths with a combination of the different above-mentioned features. We also compared paths that include waiting (contrasting short paths with longer waiting against longer paths with little or no waiting), as well as loops or interesting points that give the NPC reason to cover more tiles.

We collected a total of 104 responses and conducted the study under the hypothesis that players have a preference between different pathing behaviours. The survey is available online\(^1\) for inspection. Screen captures of each question in the survey, as well as full results can also be found in the Appendix, Section A.1. In the 10 questions, the survey asked the respondent to:

1. Choose between a simple path and a more complex path (that cuts through the level space diagonally, no obstacle).

2. Choose between a simple path and a more complex path (but this time the paths must bypass an obstacle).

   Then waiting is introduced before the next section. Now the respondent has to:

3. Observe no waiting, then choose between a long and “interesting” path or a short path with waiting.

4. Observe no waiting, then choose between a long path with loop or a short path without loop with waiting.

5. Observe no waiting, then choose between a path where NPC is patrolling with doubling back or where the NPC is on a short path, with waiting.

\(^1\)The full survey [63] can be found online, at https://goo.gl/forms/9KbNednVIZvxyHGs2.
4.1. User Study

6. Observe a long path without waiting then choose between a path that continues to be short with waiting or a long one with no waiting.

7. Observe a short path with NPC waiting then choose between a path that continues to be short with waiting or a long one with no waiting.

(Note that 6. and 7. have the same gap segment choices, but with different observations).

8. Choose between a shortest path with back and forth patrolling or a longer/interesting path (both have patrolling behaviour).

9. Choose between an NPC path involving looping/circling around obstacles behaviour or patrolling behaviour back and forth between obstacles.

10. Observe patrolling back and forth movement then choose between a path with another back and forth movement or a longest/interesting path without any more back and forth movements.

Our goal was to create a survey that could be understood and answered by respondents that are recruited from various online game communities from the website Reddit [2]. It is non-trivial to collect data on what a player may think or know during an actual playthrough without conducting a human experiment and recording the respondents’ thought process and comments throughout the experiment. Additionally, gathering playthrough data requires a complex model for data collection, and there would be a large amount of noise.

Another possibility would have been to create an interface that would allow survey respondents to draw their own paths between known path segments. Although achievable, it is difficult to impose time and spacial constraints between segments that would haven been easily represented and understood by respondents. Since our tool favours ideal paths, by proposing two different ideal paths with distinct characteristics, we decided that the survey format of asking respondents to choose between them was the best solution for our investigation purposes.
4.1. User Study

4.1.2 Study Results

Table 4.1 summarizes data from the survey results. As expected, players anticipate NPC movement to be more simple than complex. 75.0% and 68.3% of those surveyed viewed the simple paths as more likely when presented in levels with and without obstacles respectively, given a simple path and a complex path of equal length \((p < .005)\). After introducing waiting, the majority still expected the NPC to take the simpler and shorter path, preferring waiting behaviour to more elaborate pathing: 67.7% over long or interesting paths, and 61.8% over paths with loops \((p < .025)\).

Interestingly, if we show the player a longer path segment before the gap segment where the NPC engages in waiting behaviour, 58.7% picked the path that includes waiting to fill in the gap. When shown a NPC engaging on a longer, more interesting path before the gap, 67.3% chose the longer path to fill in the gap instead \((p < .005)\). We also found that 72.1% of respondents were consistent with their preference when picking the most probable path taken, possibly based on their preconceptions of how NPCs should move or behave, whether it is simple or complex. As a whole, this suggests that players take into consideration previous observations and what they know does influence their expectations. Given that our results were significant, we can thus reject the null hypothesis of players not having a preference in NPC movement.

<table>
<thead>
<tr>
<th>Preference</th>
<th>Obstacle</th>
<th>No Obstacle</th>
<th>With Waiting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple/Short</td>
<td>75.00%</td>
<td>68.27%</td>
<td>67.31%</td>
</tr>
<tr>
<td>Complex</td>
<td>25.00%</td>
<td>31.73%</td>
<td>-</td>
</tr>
<tr>
<td>Interesting</td>
<td>-</td>
<td>-</td>
<td>32.69%</td>
</tr>
<tr>
<td>Looping</td>
<td>-</td>
<td>-</td>
<td>-</td>
</tr>
</tbody>
</table>

However, when participants are given relatively short path segments to observe before the gap segment, those who chose a path with a loop over one with backtracking consists of 55.8% of those surveyed, which does not indicate an obvious preference \((p > .05)\). There was also no significant preference between long interesting paths and patrolling back-and-forth behaviour \((51.92% \text{ to } 48.08\%)\). In these cases, there is not enough evidence to reject
4.1. User Study

the null hypothesis.

Due to the nature of our study, previous video game knowledge likely influenced subject responses. 70 respondents (67.3%) claimed to play video games “often”, 23 (22.1%) said “occasionally” and 11 (10.6%) said “rarely”.

By filtering out the responses “occasionally” and “rarely” responses, we find that those who play video games often had more consistent expectations of NPC movement. For instance, 77.8% answered the same for questions 1 and 2 (compared to 72.1% of all responses). 74.6% answered the same for questions 3 and 4 (compared to 69.2%). Interestingly, 71.43% did not change their expectations of the NPC movements based on what they have observed previously (in questions 6 and 7). This may be an indication that players that do play various video games will have different preconceptions of how NPC move. Asking the respondents what games they play would give a better idea of the reasoning behind the selected paths.

In figure 4.2 we can see survey results when comparing surveyed players’ preferences for simple paths and different complex paths.

![Figure 4.2](image)

**Figure 4.2** Survey results when comparing surveyed players’ preferences for simple paths and different complex paths out of 104 respondents.
4.2. Path Generation Accuracy and Threats to Validity

Other confounding factors in our study may include the non-randomized ordering of the questions, as well as priming the participants by introducing the idea of waiting within the survey, although this was necessary to avoid confusion. For example, we explained that NPCs can sometimes engage in waiting behaviour, or may choose to take longer paths instead, and that waiting is represented by the “dot” staying on one tile for longer than a single time frame.

Of course, more data in more contexts would help clarify these issues. However, our overall conclusion is that players generally expect shorter and simpler (“ideal”) paths, and otherwise adapt their expectations to reflect what they have previously observed. The full set of responses we received can be found in the Appendix, Section A.1 in Tables A.1, A.2 and A.3.

4.2 Path Generation Accuracy and Threats to Validity

In order to determine whether the paths generated by our tool are feasible and valid, we conducted a series of experiments that allowed us incrementally test how the different features of the gap affected the generated path’s accuracy. We also used a modified version of Fréchet distance [1] to determine which set of paths generated by the tool is closest to the actual path inputted manually based on our observation of NPCs from real games, which is detailed in Section 4.4.

In the process of determining whether a path is accurate, we found that path accuracy highly depends on the complexity of the gaps, on negative space or negative knowledge, and on general player’s abilities to recall what they have observed, all of which the following sections will describe in further detail.

The biggest threat to validity stems from issues that prevent us from accurately translating observation into actual precise data recording, namely the “fuzziness” of player recollection as a result of the player’s incapability to recall the exact timing and location of non-player characters. Other threats involve difficulties in aligning different partial paths to simulate repeated observations, and limitations due to computational complexity. We discuss these threats in the following subsections.
4.2. Path Generation Accuracy and Threats to Validity

4.2.1 Complex Gaps

Similar to simple path segments, as seen in Section 3.4.2, a simple gap would involve a minimum number of possible turn combinations within the gap. A gap can be simple for various reasons, such as limitations caused by the level’s geometry, or due to its length.

When taking path examples and removing sections of it to simulate “gaps”, we found that gaps that are simple are easier to fill in – in this case, meaning the generated path segments are more accurate. Gap segments that do not include waiting also allow more accurate path generation, as do those that have a limited number of potential optimal paths. The contrapositive being that complex gaps lead to less accurate guesses on NPC’s whereabouts.

In figure 4.3, it is made clear why gap simplicity plays a role in the accuracy of generated segments. Although both middle and right examples have the same $t = 3$, the rightmost example’s gap segment has a unique solution for how it can be filled in, whereas, the middle example has multiple possible solutions, decreasing the chances at filling the gap in correctly. This is a small example to illustrate how gap complexity plays a role in path generation. In this case, it is possible to show the exhaustive list of possible paths, but when the gap becomes longer, it is not reasonable to show all the possibilities in addition to the accuracy being further reduced.

We can determine the complexity of a gap based on the number of path segments of the actual path that remains unknown, which consequently translates to the increasing number of drastically different paths possible.

4.2.2 Negative Space

As discussed earlier in Chapter 3, negative space can play an important role in narrowing down the search space of potential paths. However, when observing gameplay videos of players solving a level, while players tend to observe guard movements in segments, they do not typically explore wider areas of the map between observations. We noticed this especially in playthrough videos (from both first time players [46], and expert players [61]). This results in negative spaces focused predominantly on areas seen shortly before and after an observed path segment; these areas tend to be spaces near the observed segments as well.
4.2. Path Generation Accuracy and Threats to Validity

Figure 4.3 Two examples of gap segments, one complex and one simple, both from the same complete path (on the left). A correct segment has been generated for the simple gap, and an incorrect one for the complex gap.

As a result, although in theory negative space can be added as an interesting aspect and method to generate more accurate paths, it is an extra step that does not consistently add a great amount of information in practice. Since tracking negative space requires knowing the player’s field of view, which is difficult to translate into actual data, we did not take it into consideration when computing potential paths in our experiments, and is left to future work.

4.2.3 Difficulties of Reliably Representing Observations

The ability to accurately represent what a player observes is ultimately the biggest challenge. The precision of translating correctly what the player experiences in a game highly depends on the player’s interpretation as well as the tool used to model the observation. In our case, we noticed that player “fuzziness” as well as our grid model both may cause inaccuracies. It is a given that players are not as accurate as programs in terms of estimating positions of guards or enemies, however, in the event that diagonal or more complex movements are observed in a scenario, our Manhattan-movement grid solution to model paths may cause issues. On top of spatial inaccuracies, a player’s repeated observations
4.2. Path Generation Accuracy and Threats to Validity

may not align. We will discuss these issues below.

**Fuzziness.** The concept of “fuzziness” stems from the fact that player recollection is unlikely to be perfect, and thus when making observations, players will naturally introduce uncertainty in the model.

Uncertainty can originate from timing or location. A player may have a good idea of where an enemy was when they first see them, but as time elapses and as the $\Delta t$ becomes larger, the player will lose sense of how much time has passed. Unfortunately, introducing fuzziness into our model will simply increase the number of paths that can be generated caused by the greater gap, and potential difference in time between the perceived $\Delta t$ and the actual time taken by the NPC.

In our experiments, we did not integrate fuzziness into our model, as its severity depends highly on the observer and on the game.

**Diagonal Traversal in a Grid.** When modelling real-game scenarios, most limitations we encountered occur while trying to accurately depict and translate the player’s observations into segments. Uncertainties and inaccuracies are inevitably introduced. For example, when an NPC follows a continuous path, timing may not match up to our discrete model. In Figure 4.4, the actual NPC path (solid arrow) is much shorter than the approximated version of the path (dotted arrows). This evidently causes problems when filling in gaps, since areas that were deemed unreachable are potentially reachable within the game due to more time that is available caused by characters moving freely using “shortcuts” along more direct paths.

We can improve our approximations by implementing other movement models, and allowing diagonal neighbours to be added as valid reachable tiles. Choosing the appropriate map size (by choosing the number of tiles required to model a game level) is also a factor to consider, since larger maps allow us to model movement more accurately, but makes paths increasingly longer to compute.

Another way to counteract this problem in smaller maps is to use the *Ramer-Douglas-Peucker* algorithm, which we also use to simplify manually defined paths, detailed further in the chapter.
4.2. Path Generation Accuracy and Threats to Validity

Aligning Repeated Observations. Depicting what a player knows of an NPC cycle becomes challenging when attempting to line up multiple observations from separate observations, whether it is later in the same playthrough or upon reloading a level. When unaware of the length of an NPC’s movement cycle and when presented with very short known path segments, it is a complex task to guess where each individual segment fits in a complete path. Again, what may seem trivial is in reality a very hard problem, when including a more realistic depiction of what a player may know or observe.

4.2.4 Computational Complexity and Unbounded Possibilities

To better understand the tool’s limitations, we created maps of varying levels of different complexity to test our algorithms. Unfortunately, due to the limitations of the nature of our tool, we could not run extremely complex cases.

While selective path computations from larger maps were demonstrated to be achievable, it became evident that it was unrealistic to assess all possible paths for each gap segment, especially since wait time increases the possibilities of each path variation exponentially and numbers easily become too large to compute in a reasonable amount of time. However, by limiting the search to only what players may expect of NPC movement, we can reduce the number of possible paths generated by our tool to only a few that are realistically expected by the player.

Although our algorithms are not computationally complex, the limiting factor is in the
4.2. Path Generation Accuracy and Threats to Validity

Visualization of the problem. Because each tile and time representation for the heat map is an individually instantiated GameObject, each additional row of tiles adds to the number of objects that need to be rendered, and will slow down the program considerably. Since our tool was built as an exploration tool rather than a benchmarking tool, we did not perform rigorous performance testing. The other reason being that path-generation is highly affected by the map and the level set-up—it is very scenario-specific, and depending on the obstacles and the placement of the gaps the number of potential paths will vary greatly. Performance is thus fundamentally context-sensitive, and dominated by the size of the output (number of paths).

4.2.5 Path Simplification with RDP

The most demanding and unforgiving element of representing player knowledge is achieving consistency and accuracy. Unfortunately, error can be introduced through faulty or inadequate player observation and understanding of NPC movement, which compounds any error occurring from translating their abstract understanding of the movements into actual data.

Although we could not counter fuzziness, timing issues, misinterpretations, or the complexity, we were able to reduce the inaccuracies stemming from the jaggedness of diagonal traversals by simplifying paths using the Ramer-Douglas-Peucker [16] algorithm (RDP), which we modified to take into consideration obstacles.

Path simplification is not only useful to fix jaggedness, it is essential for automatic recognition of sets of manually defined continuous tiles better defined as one direct path segment. For example, although a path is defined by a user by successively selecting tiles that belong to the path, RDP can be run to determine whether a series of tiles are part of the same direct path segment by checking if they are part of a straight line. This is trivial to do, since we can easily pattern match to find successive tiles that are all positioned on the same column $\pm 1$ or row $\pm 1$, but not both.

However this becomes tricky when paths that are manually drawn follow more complex paths that are simply approximations of continuous movement. Figure 4.5 shows different straight paths in the Euclidean domain that require approximated paths in our grid-based
4.2. Path Generation Accuracy and Threats to Validity

representation. These approximated paths, shown in red dotted lines, may appear to be a jagged path composed of multiple path segments, when they are meant to be represent one path segment where the NPC travels diagonally across tiles.

Figure 4.5 Examples of different paths that require approximations.

Simplifying manual paths, and fixing jagged observations are essentially two sides of the same coin, and can both be remedied using RDP. Ramer-Douglas-Peucker is a recursive algorithm that, given a path or a curve composed of multiple line segments, allows its user to reduce the path to a similar one with less segments by removing points. Given a path fragment with starting and ending positions, the algorithm determines if a straight line approximation is sufficient based on whether or not the distance of all intermediate points from the proposed straight line approximation are less than a given threshold $\epsilon$.

Simply put, RDP recursively goes through the list of points (tiles), finds the one that is furthest in perpendicular distance to the line formed by the first and last points of the segment, and creates two new segments using the furthest point as new head/tail for the new segments. This is repeated until the maximum distance between the furthest point and the line is smaller than the selected $\epsilon$. Figure 4.6 shows an example of a curve that is simplified through 5 iterations of RDP, before the algorithm terminates when the difference between the original and simplified path is acceptable. The resulting simplified path consists of
4.2. Path Generation Accuracy and Threats to Validity

points that are a subset of points from the original path.

![Figure 4.6 Example of a curve through multiple iterations of RDP.](image)

The original algorithm mainly considers continuous paths in a non-game setting, and therefore does not take into account obstacles. Below, Algorithm 2 shows the pseudocode for the modified Ramer-Douglas-Peucker algorithm we used. The modification we made is to take into consideration obstacles, which the original algorithm does not do. For every iteration, we ensure that the newly formed simplified path does not clip through obstacles. We use a similar modification in Algorithm 3 calculating path accuracy in our experiments.

The modification to the original RDP algorithm consists of adding an extra check to ensure that the new path does not cut through obstacles in the level.

Alternatively, this algorithm can also be used to approximate continuous paths and to map key tiles, indicating places where the NPC turns or makes changes to its path trajectory from an inputted list of tiles. Even for non-continuous paths that we manually define within the tool, the RDP algorithm can be used to simplify the path by only keeping critical tiles, meaning the start and end tiles of each path segment. This is achieved by choosing a small enough $\varepsilon$, generally smaller than 1.

In Figure 4.7, we show the RDP algorithm used in different ways. From the original path, if RDP is performed with $\varepsilon = 0$, then the path is simplified to less segments without any path difference: this results in a simple path. In our example, the simple path has 5
### Algorithm 2 Modified Ramer-Douglas-Peucker Algorithm

**Require:** \( path \leftarrow tilelist \)

1. **procedure** \( RDP(tilelist, \epsilon) \)
2. \( nlist \leftarrow null \)  \hspace{1cm} \triangleright \text{Create empty list} 
3. \( maxDistance \leftarrow 0 \)
4. \( furthestTile \leftarrow tilelist[0] \)
5. \( h \leftarrow 0 \)  \hspace{1cm} \triangleright \text{Get first and last tiles} 
6. \( t \leftarrow tilelist.size - 1 \)
7. **for all** \( n \) **in** \( path \) **do**  \hspace{1cm} \triangleright \text{Check for furthest tile} 
8. \( d \leftarrow \text{perpendicularDistance}(n, \text{Line}(h,t)) \)
9. **if** \( d > maxDistance \) **then**
10. \( maxDistance \leftarrow d \)
11. \( furthestTile \leftarrow n \)
12. \( i \leftarrow furthestTile.index \)
13. **if** \( maxDistance > \epsilon \) **then**  \hspace{1cm} \triangleright \text{Recursive Call} 
14. \( \text{firstHalf} \leftarrow RDP(path[h...i], \epsilon) \)
15. \( \text{secondHalf} \leftarrow RDP(path[i...t], \epsilon) \)
16. \( nList \leftarrow \text{firstHalf} + \text{secondHalf} \)
17. **else**
18. \( \text{obstacleHit} \leftarrow \text{HASOBSTACLE}(tileA, tileB) \)  \hspace{1cm} \triangleright \text{Check for any hit obstacles} 
19. **if** \( \text{obstacleHit} \) **then**  \hspace{1cm} \triangleright \text{New Path Goes through obstacles, keep iterating} 
20. \( \text{firstHalf} \leftarrow RDP(path[h...i], \epsilon) \)
21. \( \text{secondHalf} \leftarrow RDP(path[i...t], \epsilon) \)
22. **else**
23. \( nList \leftarrow newPath[h,t] \)
24. **return** \( nList \)

---

**Figure 4.7** Example of simplifying a multi-segment path using RDP and a modified RDP.
4.2. Path Generation Accuracy and Threats to Validity

path segments.

However when $\varepsilon = 1$ or greater, then we see that the effect of RDP on the original path is much more drastic in this case. The path is further simplified but may become apparently different, such as the examples in Figure 4.7, even allowing the path to traverse obstacles, although still respecting $\varepsilon$ (total of 2 path segments). However, by applying a simple modification to RDP algorithm to be conscientious of obstacles, the algorithm will result in a better simplification that does not allow the path segment to traverse tiles that are marked as obstacles (in this case, path of 3 segments). Although interesting, larger $\varepsilon$ are not useful because it will introduce inaccuracy in the path representation, but it is useful to simplify multiple short segments that are part of a longer segment in a manually defined path.

Essentially, the RDP algorithm proves itself useful in cases where the manually defined path (by clicking on individual tiles) can be simplified automatically into simpler paths to the desired approximation accuracy. However, it may introduce error when simplifying jagged segments caused by an observation of a diagonal traversal of a map that uses Manhattan movement, especially in terms of timing.

4.2.6 Summary

Representing what a player sees or knows in any given situation is easier said than done. In the process of accurately narrowing down a large number of possibilities, we encountered a series of complications, from the complexity of gap segments that we are trying to fill to difficulties representing observations. Although we countered some of the problems (diagonal traversal, manual path simplification) using RDP, others difficulties such as aligning repeated observations requires further effort. Nevertheless, representing the most crucial player knowledge is feasible and worthwhile, exhaustively integrating every aspect of what a player may know or observe requires us to consider an insurmountable and infinitely many threats to the validity of the representation.
4.3 Testing Path Similarity

Despite the difficulties in perfectly representing knowledge, we strive to test and compare sets of generated paths to actual NPC paths, both artificial ones and real ones taken from real game examples. In order to gather more data, we created examples of made-up levels with predetermined paths based on how an NPC would typically move, based on our observations of real games. Then we removed segments of the path, increasing in size, and used our tool to fill in the gaps back using generated segments. We then performed similar tests for real game examples.

We used the tool that we developed to represent maps and scenarios and to run experiments. Our method involves replicating a game and modelling inside the tool in order to determine how accurate the generated paths are at reproducing missing path segments. As a result, we created complete paths, and then drop out sections to create gap segments that we then proceed to fill in with our path generation tool. We used two separate sets of paths, one with waiting, and the other without.

The subsets of generated paths are produced by the tool based on a random selection from paths that fit the given criteria. This introduces some noise, but testing all paths that can be discovered from the tool is infeasible and not overly meaningful, given the conservative solution includes all possible paths, some very similar to each other. In order to compare manually defined paths and reduce the number of points along a path segment, we used RDP to simplify manual paths. Then we tested similarity between the generated path and the actual path segment using a modified discrete Fréchet [20] distance, as described below.

4.3.1 Testing Path Similarity Using Fréchet Distance

In order to decide which group of paths (i.e. shortest, exact steps or interesting) generates the “closest” path (or most similar path) to the actual path, we use the Fréchet distance to determine path similarity. Fréchet uses distance to calculate similarity between two curves [1] using points along the curves.

The algorithm for Fréchet distance determines the similarity of two continuous curves
4.3. Testing Path Similarity

by using a bicontinuous function between the curves and minimizing the maximum pair-wise distance. The popular analogy is that a man walks along one curve while his dog, on a leash, travels on the other curve. Both may move at different speeds, but cannot backtrack. The Fréchet algorithm computes the shortest leash possible that would allow for the man and dog to walk such paths. The algorithm can be simplified and discretized [20] and Figure 4.8 shows an example of the dog, man and leash metaphor.

![Diagram of Fréchet algorithm analogy](image_url)

**Figure 4.8** Illustration of the man and dog analogy. The dog travels along the red line, while the man on the blue. The dotted lines represent leash length throughout the walk.

In our implementation we used the points of the path generated by the algorithm compared to the tiles from the actual path. We use every point along the path, and not the simplified path using RDP, as we want to calculate similarity as precisely as possible. This is because the discretized version of Fréchet may cause loss of precision if points along two identical paths are not the same or aligned, as shown in Figure 4.9; two equivalent paths may yield a non-zero Fréchet Distance, and as we add more points along each segment, the result will eventually converge to continuous Fréchet. By keeping as many points as possible on our grid map, we can somewhat mitigate the issue.

However, neither the original Fréchet, nor the discrete Fréchet can fully accommodate paths in the game level context. Pathing in games is critically affected by obstacles, and Fréchet distance by itself does not consider the impact of obstacles on the length of the leash as the dog and man move. We thus require a modified discrete Fréchet that will help
4.3. Testing Path Similarity

Figure 4.9 Example showing that despite two paths 1 (red) and 2 (blue) being identical, the Fréchet Distance between them is non-zero because path 2 has one more point.

We distinguish between two paths that are exactly the same, except for the presence of an obstacle. This enables us to consider paths of similar distance on the same “side” as the obstacle to have greater similarity.

Figure 4.10 Modified Fréchet allows us to penalize paths that are not on the same side of an obstacle.

In Figure 4.10, we have an example where the actual NPC path is compared to two generated paths 1 in blue and 2 in green. Based on the original algorithm, these two paths would appear to be equally similar to the actual path, since they both yield a maximum distance of 2 at any point. However, when taking into consideration the obstacles, our
4.3. Testing Path Similarity

modified algorithm reflects the fact that path 2 is more similar to the actual NPC path, since the leash does not intersect as many (or any) obstacles at any given time frame. In this case, path 1 will result in a larger distance, and be less similar.

One solution to this is to detect the presence of obstacles between the paths and account for it: we implemented this using Raycast in the tool to take into consideration obstacles. This is a function native to Unity®, and allows us to detect game objects along a line. If an obstacle is found when drawing a line between two points, we calculate the shortest path around the obstacle and assign that as the length instead.

![Diagram showing path similarity with and without obstacles](Image)

**Figure 4.11** Example of the same man and dog analogy as Figure 4.8, but using the modified Fréchet algorithm that considers obstacles.

Figure 4.11 shows the same example as the previous example depicted in Figure 4.8, but in a game environment. This means that there are potentially obstacles that separate different paths, and we want to ensure that paths that are on different sides of an obstacle are considered as drastically different paths. Below, Algorithm 3 shows the pseudo-code for the modified version of calculating Fréchet Distance.

The basis of the algorithm essentially uses the discrete Fréchet method to calculate the largest difference between two paths. The algorithm iterates through all the points on the two paths that are being compared and increments the pointer on the path that would yield the shorter “leash”. The modification ensures that when calculating the Euclidean distance between any two points, if the straight line encounters an obstacle (e.g.
4.3. Testing Path Similarity

Algorithm 3 Modified Fréchet

Require: pathA ← tilelist
Require: pathB ← tilelist

1: procedure MODIFIEDFRÉCHET(pathA, pathB)
2: maxDist ← 0
3: i ← 0
4: j ← 0
5: distGrid[i][j] ← new float[pathA.LENGTH][pathB.LENGTH]
6: for (i = 0; i < pathA.LENGTH; i++) do ▷ Fills a grid of distances of size ixj
7:     for (j = 0; j < pathB.LENGTH; j++) do
8:         distGrid[i][j] ← GETDISTANCE(pathA.GET_TILE(i), pathB.GET_TILE(j))
9:   while i < pathA.LENGTH-1 || j < pathB.LENGTH-1 do
10:      d1 ← distGrid[i+1][j+1]
11:      d2 ← distGrid[i+1][j]
12:      d3 ← distGrid[i][j+1] ▷ Pick shortest leash by advancing on path A and/or B
13:         if d1 < d2 & d3 then
14:             i ← i + 1
15:             j ← j + 1
16:             if d1 > maxDist then
17:                 maxDist ← d1
18:         if d2 < d1 & d3 then
19:             i ← i + 1
20:             if d2 > maxDist then
21:                 maxDist ← d2
22:         if d3 < d1 & d2 then
23:             j ← j + 1
24:             if d3 > maxDist then
25:                 maxDist ← d3
26:      return maxDist
27: procedure GETDISTANCE(tileA, tileB)
28:     if HASOBSTACLE(tileA, tileB) = true then ▷ Considering Obstacles
29:         return SHORTESTPATH(tileA, tileB) ▷ Use shortest path as distance
30:     else
31:         return EUCLIDEANDIST(tileA, tileB)
4.3. Testing Path Similarity

hasObstacle(tileA, tileB) = true) then we would use the shortest path between the two tiles as the minimum distance instead. Then, the algorithm chooses the shortest of the three possibilities: advance to the next point on either one or both leashes that would yield the shortest “leash” which is the greedy approach used in the discrete Fréchet [20] method.

Because our modification relies on distance measure, the calculated result is no longer a purely Euclidean distance, thus not a Fréchet distance, but rather a heuristic measure of similarity. Our algorithm allows us to easily determine the path similarity of a generated path to the actual path, and the modification will heavily penalize paths that circle obstacles requiring significant deviation from a straight line or from the actual path.

Our modified Fréchet algorithm is one of the heuristic measures used in our experiments to determined whether a generated path is accurate. It should be noted that the modification causes the original algorithm to lose its translational symmetry property and variants when taken outside the game scenario, and it does not take into consideration wait time when comparing paths. For these reasons, we opted to use it in addition to the original Fréchet distance calculations, and to area and volume metrics. We will discuss this further in the upcoming sections.

4.3.2 Testing Artificial Paths with Dropout

As discussed in Chapter 3, we suspect that players will predict NPCs to follow certain regular movement patterns and according to their preconceptions of NPC movement. In this section, we will give examples of complete paths, and we proceed on removing sections of it of different sizes in order to simulate gap segments. Then, we will use our tool to recover the lost path segments to see which category generates the dropped path segments more accurately. We measure accuracy using the modified Fréchet described prior in Section 4.3. Not only do we look at the Fréchet value, but we also a calculated the average size of the “leash” in the walk as an extra point of comparison that will serve as an extra heuristic measure.

Hypothesis. We suspect that smaller gaps will result in more accurately generated paths, as well as that relatively simple gaps will cause more accurately generated paths compared
4.3. Testing Path Similarity

to more complex gaps of the same size. We also suspect that shortest path always generates
the most similar paths to the actual paths.

4.3.2.1 Testing paths without waiting behaviour

To begin, we use the chosen complete path (shown in figure 4.12) of size $t = 20$ as our first
small example. We opted to do tests on gaps of different sizes: 4 (small, or less than 20% of
total path size), 6 (medium, or between 20% and 50% of total path size), and 10 (large,
meaning 50% or more of the complete path is unknown).

![Image](image.png)

**Figure 4.12** The full path drawn on the map as a reference.

**Small size gaps.** Figure 4.13 are images of the partial paths that were taken into consider-
ation for the chosen gap size of $t = 4$. Although there are many more, these partial paths were
chosen for the various placements of the gap segment. Non-chosen ones were omitted because they were symmetric, similar, or uninteresting when compared to the selected ones.

Figure 4.14 are the results of filling in the gap size of $t = 4$ at various places of the full
path. Because the gap size is considered “small”, all generations of paths with the pro-
erties of “simple” and “shortest” produced the correct path segment. The other properties,
4.3. Testing Path Similarity

Figure 4.13 The example variations of dropping various small-size segments of the path (i.e. gaps of size \( t = 4 \)).

however generated other possible shortest paths, since there was no waiting involved. In the figure we overlaid multiple iterations of generated paths for each separate gap segment to show what was generated and how frequently (thicker lines indicate multiple overlaid paths).

We notice that when the gap includes a change in direction or a turn, there tends to be more than one predicted solution, when multiple paths are possible. When no waiting is a property (i.e. \( \Delta t = \Delta t_{\text{min}} \)), all examples in Figure 4.14, all the categories yield very similar results, particularly for simple gaps.

Table 4.2 summarizes our measure of path generation accuracy, over all generated paths. For instance, generating “simple paths” always yield the correct path (for all 5 variations), while others, when presented with a more complex gap (e.g. 1a, 1d), had varying results. Simple gaps (e.g. 1b, 1c, 1e) always resulted in the correct path segment, evidently because in this case there are no other possible solutions. The results were as we expected, as seen in Figure 4.14.

<table>
<thead>
<tr>
<th>( \Delta t = 4 )</th>
<th>1a, 1d</th>
<th>1b, 1c, 1e</th>
</tr>
</thead>
<tbody>
<tr>
<td>Property</td>
<td>Fréchet Dist.</td>
<td>Average</td>
</tr>
<tr>
<td>Simple</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Shortest</td>
<td>1.41</td>
<td>0.68</td>
</tr>
<tr>
<td>Exact Steps</td>
<td>1.41</td>
<td>0.68</td>
</tr>
<tr>
<td>Interesting</td>
<td>1.41</td>
<td>0.68</td>
</tr>
</tbody>
</table>
4.3. Testing Path Similarity

Medium size gaps. When increasing the gap size to \( t = 6 \), we notice that more path possibilities emerge in cases where corners are involved in the gaps, whether there is waiting involved or not. This is in agreement with our previous discussion on gap complexity. Figure 4.15 shows the sampled gap segments we used, and Table 4.3 summarizes the results.

It may be worth noting, that out of all the tested gaps (2a – 2e) in set 2, only 2a had the Modified Fréchet detect an obstacle, which indicates that the set of generated paths for that gap contained paths that were on a different side of an obstacle than the actual path. This was detected in “Shortest”, “Exact Steps” and “Interesting”, but not in “Simple”.

Once again, we notice that Fréchet distance is 0 for 2c and 2e as expected, because of the simplicity of the gap: there is only one possible path, and therefore all categories

![Image](image_url)
4.3. Testing Path Similarity

Figure 4.15 The example variations of dropping various medium-size segments of the path (i.e. gaps of size $t = 6$).

Figure 4.16 Result of filing in the gaps of size $t = 6$. 
4.3. Testing Path Similarity

Table 4.3 Results for all Generated Gaps of Size $\Delta t = 6$

<table>
<thead>
<tr>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
<th></th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Shortest</td>
<td>2.83</td>
<td>1.86</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>1.41</td>
<td>0.24</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Exact Steps</td>
<td>2.83</td>
<td>1.92</td>
<td>1.41</td>
<td>0.40</td>
<td>0</td>
<td>0</td>
<td>1.41</td>
<td>0.69</td>
<td>0</td>
<td>0</td>
</tr>
<tr>
<td>Interesting</td>
<td>2.83</td>
<td>1.92</td>
<td>1.41</td>
<td>0.57</td>
<td>0</td>
<td>0</td>
<td>1.41</td>
<td>0.69</td>
<td>0</td>
<td>0</td>
</tr>
</tbody>
</table>

generated it correctly.

Large size gaps. As expected, When increasing the gap size to $t = 10$, the computed average and maximum Fréchet values are even larger, seen in Table 4.4. Figure 4.17 shows the corresponding partial paths.

![Figure 4.17 Partial paths selected for gaps of size $t = 10$.]

Table 4.4 Results for all Generated Gaps of Size $\Delta t = 10$

<table>
<thead>
<tr>
<th>Property</th>
<th>3a F. D.</th>
<th>Avg. F. D.</th>
<th>3b F. D.</th>
<th>Avg. F. D.</th>
<th>3c F. D.</th>
<th>Avg. F. D.</th>
<th>3d F. D.</th>
<th>Avg. F. D.</th>
<th>3e F. D.</th>
<th>Avg. F. D.</th>
</tr>
</thead>
<tbody>
<tr>
<td>Simple</td>
<td>0</td>
<td>0</td>
<td>2.83</td>
<td>1.16</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>0</td>
<td>2.83</td>
<td>1.70</td>
</tr>
<tr>
<td>Shortest</td>
<td>1.41</td>
<td>1.14</td>
<td>2.83</td>
<td>1.16</td>
<td>1.41</td>
<td>0.31</td>
<td>2.83</td>
<td>1.07</td>
<td>2.83</td>
<td>1.70</td>
</tr>
<tr>
<td>Exact Steps</td>
<td>5.66</td>
<td>2.26</td>
<td>2.83</td>
<td>2.19</td>
<td>5.66</td>
<td>2.78</td>
<td>2.83</td>
<td>0.96</td>
<td>2.83</td>
<td>1.56</td>
</tr>
<tr>
<td>Interesting</td>
<td>5.66</td>
<td>2.86</td>
<td>5.66</td>
<td>3.19</td>
<td>5.66</td>
<td>2.69</td>
<td>5.66</td>
<td>2.86</td>
<td>6.32</td>
<td>3.84</td>
</tr>
</tbody>
</table>

When we compare the various sizes of gaps, it is evident that, as the gaps become a significant portion of the actual path, the generated paths are generally less accurate, and this is as we expected.
4.3. Testing Path Similarity

For each gap segment we generated 10–15 paths per group (simple, shortest, exact steps, interesting) and recorded the maximum Fréchet measured, and the average of all the leashes of all generated paths. This is only a random sample from the large number of possible paths, but they give us sufficient insight.

Measuring player uncertainty using area. We also use area as a measure the evaluate the amount of uncertainty the player has in a level. The greater the size of possible locations and positions, the more difficult it is for the player to be certain of the observed NPC’s whereabouts. Using our tool, calculating area is simple, as it is exactly the number of tiles that may be travelled to by the NPC during the gap segment. This includes the start and end tiles.

Table 4.5 Areas of potential NPC locations depending on the size of gaps based on the same Examples 1, 2 and 3 above.

<table>
<thead>
<tr>
<th>Area for</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Small Gap</td>
<td>7</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>$\Delta t = 4$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>2. Medium Gap</td>
<td>14</td>
<td>9</td>
<td>7</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>$\Delta t = 6$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>3. Large Gap</td>
<td>25</td>
<td>32</td>
<td>29</td>
<td>25</td>
<td>34</td>
</tr>
<tr>
<td>$\Delta t = 10$</td>
<td></td>
<td></td>
<td></td>
<td></td>
<td></td>
</tr>
</tbody>
</table>

Results. Based on the first set of tests, we can conclude that, in a small size map, gap complexity plays a large role on the accuracy of generating potential paths. We also noticed that generally, “simple” and “shortest” paths are the most accurate at predicting the actual path. This is expected in such a simple example.

It may be worth noting that using the largest Fréchet Distance in a set of paths is a plausible measure for this case, but calculating the average is less meaningful, because completely different diverging paths may average out very similarly for various sets of paths.
4.3. Testing Path Similarity

Figure 4.18 Visualization of the resulting areas of where the NPC may be during the gap of all the variations of examples 1, 2 and 3.

4.3.2.2 Testing Paths with Waiting Behaviour

The examples in the previous section do not consider paths that include waiting. This is because taking into account waiting can be tricky and adds in complexity. As previously discussed in Section 3.4.3, the time spent waiting can be spread out and randomly distributed anywhere along the generated path. When two paths seem identical but involve NPCs waiting at different locations, what the player observes and knows of the NPC will be different. Although Fréchet takes into consideration the ordering of points, it does not take into account wait times, or when an NPC stays stationary. To mitigate this, we can simply use positions of points in a 3D space. This means that two paths that have more points with similar wait time and location, the Fréchet distance will be much smaller than if the
4.3. Testing Path Similarity

generated wait location was drastically different than actuality, but is otherwise identical to the actual path.

**3D Fréchet.** In our 3D Fréchet calculations, our goal is to take into consideration waiting time into our calculations, but to only allocate a small non-null weight to it. By including the vertical position of each point along the path by $0.5 \times t_e$ in height, the general 2D path remains the main contributor to the final 3D Fréchet Distance, and we found that a factor of 0.5 lessen the impact waiting has on evaluating path accuracy, yet still contributes in comparing different paths. Figure 4.19 shows an example, where Fréchet is calculated in a 3D space. In these 3D representations, the distances may appear distorted because of the 2D nature of the image. Additionally, the height shown in the picture is 1 for each increment of $t$ to better show the path, but in our calculations $0.5 \times t_e$ is used to calculate 3D Fréchet Distance.

![Figure 4.19](image)

**Figure 4.19** Top and 3D view of an example of Fréchet distance calculated in 3D space.

Figure 4.20 shows one of the 19 generated “Shortest with Waiting” paths by running an actual calculation for a small map with 3D Fréchet distances. The maximum generated Fréchet Distance was 1.80, and average distance of all leashes for all paths is 0.90.

In turn, we can compare this result to a no wait “Exact Steps” set of generated paths
4.3. Testing Path Similarity

Figure 4.20 One of the “Shortest With Waiting” paths generated with $\Delta t = 15$. Results for the set of paths (not indicated in this figure) are: Fréchet Distance 1.80, and average distance 0.90.

using the same map and settings (same gap start and end positions, $\Delta t = 15, \Delta t_{\text{min}} = 7$) and find that max Fréchet Distance was 1.41 and the average distance 0.76. One of the 19 paths generated can be found in Figure 4.21

Results using the Fréchet Distance and average leash distance show that these are two different heuristic measures, both valid ways to calculate path similarity. For the same scenarios, one may yield a bigger Fréchet distance while the other a larger average distance. These tests can also be used in combination with the original Discrete Fréchet to obtain a better general idea. Determining which method is better to evaluate path similarity in different level situations can be interesting future extension of our work.

Measuring player uncertainty using volume. Similarly, instead of using area to measure
4.3. Testing Path Similarity

![Diagram showing two paths with Fréchet Leash Distances](image)

**Figure 4.21** One of the “Exact Steps” paths generated with $\Delta t = 15$ and $\Delta t_{\text{min}} = 7$. Results for the set of paths (not indicated in this figure) are: Fréchet Distance $1.41$, and average distance $0.76$. Distances may seem distorted, caused by the 3D figure and nature of the representation.

Player difficulty, we can use volume to take into consideration waiting time. The volume is the sum of all the possible tiles to which the NPC may travel, at any given time within the gap segment. The greater the wait time within the gap, the more possibilities there will be, hence the greater the volume of possible positions. In our tool, we simply calculate the volume as $\sum((\Delta t - t_n.l_b) - t_n.l_a + 1)$, $t_n$ being each reachable tile given the specific start and end points of the gap segment. As a reminder, in Chapter 3, we defined $l_a$ and $l_b$ as the labels that indicate the earliest arrival time and latest leaving time for each tile, respectively.

Most times when the $\Delta t$ increases, presumably the area will increase as well, but it is not always the case. In the example shown in Figure 4.22, the top two images represent the top-down and 3D views of a gap with a $\Delta t$ of 7, and the bottom two images represent a gap.
4.4. Real Game Examples

that has a $\Delta t$ of 8. Although in both cases the area is identical (9), their volume is different (top example of $\Delta t = 7$ has volume of 10, bottom example has a volume of 20). This is because the uncertainty of where the player may have waited when the $\Delta t > \Delta t_{\text{min}}$ increases the difficulty, as the difference between $\Delta t$ and $\Delta t_{\text{min}}$ increases. In some cases like the one shown in the example, the extra wait does not allow the character to travel to additional tiles, yet the wait still adds additional uncertainty as to where they may have waited.

With this in mind, Table 4.6 shows a summary of the volumes of various synthetic levels depending on the gap size and $\Delta t_{\text{min}}$ of each level.

Table 4.6 Areas of potential NPC locations depending on the size of gaps based on the same Examples 1, 2 and 3 above.

<table>
<thead>
<tr>
<th>Volume for</th>
<th>a</th>
<th>b</th>
<th>c</th>
<th>d</th>
<th>e</th>
</tr>
</thead>
<tbody>
<tr>
<td>1. Small Gap $\Delta t = 4$</td>
<td>7</td>
<td>5</td>
<td>5</td>
<td>7</td>
<td>5</td>
</tr>
<tr>
<td>2. Medium Gap $\Delta t = 6$</td>
<td>14</td>
<td>9</td>
<td>7</td>
<td>9</td>
<td>7</td>
</tr>
<tr>
<td>3. Large Gap $\Delta t = 10$</td>
<td>25</td>
<td>32</td>
<td>29</td>
<td>25</td>
<td>34</td>
</tr>
</tbody>
</table>

4.3.3 Summary

Using artificial examples, we have demonstrated that path similarity tests can be performed in order to measure the accuracy of the generated paths using our tool. Likewise, a player’s expectations can be compared to the actual taken paths of NPCs with non-waiting behaviour (in 2D) and with waiting behaviour (in 3D) using the same testing methods. In the following section, we extend our testing to real game examples.

4.4 Real Game Examples

In the stealth world, as previously discussed, a player observing non-player enemies and moving according to their expectations is part of the core of the gameplay. In this section,
4.4. Real Game Examples

Figure 4.22 An example where the top two images represent the top-down view and 3D view of a gap with a $\Delta t$ of 7, and the bottom two images are views of a gap that has a $\Delta t$ of 8.
we analyze stealth levels from *Dishonored*, and from *Aragami* to better assess whether our methods can be applied to a real game situation.

Ideally, we take existing in-game scenarios and, using our tool, translate them into path segments and let the algorithm fill in the gap segments with potential paths. As we observed in-game non-player-characters in *Dishonored*, and other games, we quickly realized that translating in-game movement into path segments isn’t a trivial task. Although imperfect, we used our 4-way (then expanded to 8-way) movement model to approximate the continuous movements of NPCs in the games we observed, and to evaluate the accuracy of the generated paths. We also use area and volume to further assess the difficulty of the real game scenarios. To obtain a better idea of how NPCs generally move in stealth games, we also visited games such as *Pillars of Eternity* [38] and *Thief* [19] to collect more information on how enemies move, as well as how our gap segment representation applies to other stealth games.

In *Pillars of Eternity* [38], the player controls multiple characters in orthogonal view to complete tasks while avoiding being detected by patrolling guards and enemies. Although the player sees a third-person view of the level, further areas are often hidden or engulfed in darkness. As an example, Figure 4.23 shows that although the player can see the movements of the NPC enemy in a bird’s-eye view, when the enemy moves away, or become too far from the player characters, they will simply disappear from the map, as shown in a series of screenshots taken from actual gameplay [47] in Figure 4.24. Essentially, in this game, the player will encounter gaps caused by distance (as opposed to by obstacles), and will lose track of the enemy’s whereabouts during their observation if they (or the enemy) wanders too far.

We also looked at another popular stealth game *Thief* [19], where the player plays in first person, similar to the play style in *Dishonored*. Many instances of observing NPC guards have demonstrated that guards indeed move in predictable and simplistic ways. Their paths are short and repeated, such as the one shown in Figure 4.25.

In the following subsections, we will use concrete examples from *Dishonored* and *Aragami* in order to better understand how our methods apply to real game scenarios.
4.4. Real Game Examples

Figure 4.23 Orthogonal view in *Pillars of Eternity*, player sees a portion of the map (bottom image shows the region highlighted in first image), and can observe the movements and paths taken by the NPC enemy.

4.4.1 Dishonored

We use levels from *Dishonored* [4] to analyze the guards movements and how a player may expect the guards to move. In this specific game, guards are stationed in their own designated areas and are patrolling it. They do adapt their behaviour to their environment, and will check on suspicious sounds. Using distractions, the player can create an “interesting” site for the guards to investigate, if the diversion successfully catches their attention.

Because the game is played in a first-person perspective, the player is initially presented with only partial views of the level’s layout and space. In any given level, a player is aware of a mission they must accomplish, and they may choose a stealth approach and avoid the
4.4 Real Game Examples

Figure 4.24 A sequence of actual gameplay from *Pillars of Eternity* where a gap is created when the guard and characters get too far from each other. Magenta arrows were added to show the observed guard movements. The screen captures were enhanced (in brightness and contrast) for clarity.

guards. Should they get caught, the game is over and a previous save point is reloaded. This is the perfect opportunity for the player to learn from its previous attempts and to piece together the NPC’s guard movements in order to bypass them. Interestingly, some maps found in-game can be consulted to gain immediate knowledge of the area, although players must still observe guards to learn their movement patterns.
4.4. Real Game Examples

Figure 4.25 A sequence of actual gameplay from *Thief* where the player watches the guard for a full cycle before proceeding. The screen captures were enhanced (in brightness and contrast) for clarity.

4.4.1.1 Modelling levels in *Dishonored*

In *Dishonored*, many level goals involve being stealthy and avoiding detection by enemy guards. In many cases in the game, the player may hide from the guards to observe their movement, but due to hiding behind obstacles, or looking away, they may only see part of the guards paths. They often disappear from view to reappear at a later time. We are interested in where the player may think the guard went between observations that they have made. Figures 4.26 and 4.27 shows an example of the player observing guards patrolling an area while staying hidden from them.

As an example, from the player’s stationary point of view near *Holger Square* (as seen
4.4. Real Game Examples

**Figure 4.26** A screen capture of the game Dishonored: example of the player observing a guard (circled in red) and his movements from a distance without being detected.

**Figure 4.27** A screen capture of the game Dishonored: example of the player observing multiple guards patrolling an area while staying hidden.
in Figure 4.27), we can observe, and thus model partial movement of the guards stationed nearby. When translating observed guard movement to observed path segments, since there is no tracking in-game, observations are subject to the player’s impressions and are not exact nor precise. Besides the issue of fuzziness when collecting data, we found that it is difficult to observe complete paths without triggering the guards’ awareness of the player, and thus changing their behaviour to engage in combat with the player.

However, in order to collect evidence for our gap-filling methods, we mapped the guard movements to the best of our abilities without being detected to reproduce complete observable paths, and reproduced the level using our Unity® tool, and then retroactively removed segments of the complete paths to assess the accuracy of our algorithms.

During the process of gathering data for a concrete in-game scenario, we found that selecting a level to model was tricky. The difficulty in the data collection is a direct result of game developers enforcing scenarios in order to control how the game unfolds, and how they want the story plays out. For example, ideally we want to observe the NPC’s full path cycle, in order to run test cases and to have a reference point for comparison. Instead, it is often impossible to obtain such full cycles without triggering the next segment of the storyline, or without simply being spotted by a nearby guard, and thus triggering a reaction from all guards, breaking the natural patrol route of the NPC that was initially being observed.

As a result, we will analyze the partial observations of in-game levels, taken from actual gameplay found online, to collect data from scenarios closer to what a player may experience.

**Test Cases.** We tested the accuracy of the paths that were generated by varying different factors: the length of gap segment, and complexity of the segment (including interest points). In the case of *Dishonored*, guards have complete paths that are relatively short (most complete paths involve less than 15 time steps), but they include a lot of waiting. Ideally, guard paths are long, and include less waiting, as waiting introduces a lot of uncertainty. Nevertheless, we still modelled an in-game scenario, and used our tool to map out the guard movements.

We quickly found that real gameplay result in short gaps. As an example, Figure 4.28 shows the depiction of actual partial path observations taken from online gameplay [41].
4.4. Real Game Examples

in *Dishonored*. To fill this gap, using simple shortest path, we can generally recover the NPC’s movements easily and accurately.

![Figure 4.28](image)

**Figure 4.28** Observed actual gameplay result in very short and obvious gaps. The area of gap segment is 4, and its volume of possible positions is 4.

When a player observes guards, they rarely lose sight of the enemy for long, such as the screen captures of the gameplay shown in Figure 4.29, taken from an online gameplay video [7].

A second example can be observed in Figures 4.30 and 4.31. The enemy in the room can be observed through the doorway, or by peeping through the keyhole. And although the gap is more complex, because the guard path is short, as the size of the gap increases, the area and volume increase as well. The results are shown in Figure 4.32. The stepped effect of the increase in area is caused by the fact that only an increase of 2 in $\Delta t$ would allow for additional reachable tiles.

Figure 4.33 illustrates the effect as well; it depicts the increase in area and volume of where the player may expect to find the NPC during the gap segment if the player makes no assumption about the NPC’s movements. If the player expects the NPC to take the shortest path, the area and volume will then only include the tiles that are available when $\Delta t = \Delta t_{\text{min}}$. This means the area will remain 10 unit$^2$ in this example and volume is much less as well. Volume will still increase with $\Delta t$. This is interesting considering using Fréchet distance, values remained mostly small or unchanged compared to how much volume and area increases.
Figure 4.29 Screen captures of observed actual gameplay result in very short and obvious gaps.
4.4. Real Game Examples

Another instance of partial knowledge in *Dishonored* is when the player uses the unlockable game mechanic *dark vision* to see through obstacles. Although this gives the player a great advantage in observing the enemy guards without being in danger of being spotted, if the player distances themselves too far from the guard, they will lose track of the guard and create a gap in the observation. An example of this from actual gameplay [57] can be seen in Figure 4.34, and the representation of it in our tool is shown in Figure 4.35. Once again, the Fréchet Distance between the generated paths and actual path is less than 1. As seen in the results in Figure 4.36 this particular example has a solution area of $17 \text{ unit}^2$ and a volume of $28 \text{ unit}^3$.
4.4. Real Game Examples

**Figure 4.31** Path segment representations of what was observed (left), the path generated for the gap (middle), and the 3D representation of it (right), map from the level depicted in Figure 4.30.

<table>
<thead>
<tr>
<th>$\Delta t$</th>
<th>Area (unit$^2$)</th>
<th>Volume (unit$^3$)</th>
</tr>
</thead>
<tbody>
<tr>
<td>7</td>
<td>10</td>
<td>10</td>
</tr>
<tr>
<td>8</td>
<td>10</td>
<td>20</td>
</tr>
<tr>
<td>9</td>
<td>24</td>
<td>44</td>
</tr>
<tr>
<td>10</td>
<td>24</td>
<td>68</td>
</tr>
<tr>
<td>11</td>
<td>44</td>
<td>112</td>
</tr>
<tr>
<td>12</td>
<td>44</td>
<td>156</td>
</tr>
<tr>
<td>13</td>
<td>66</td>
<td>222</td>
</tr>
<tr>
<td>14</td>
<td>66</td>
<td>288</td>
</tr>
</tbody>
</table>

**Figure 4.32** Comparison of area and volume as $\Delta t$ increases for the level from Figure 4.30.
4.4. Real Game Examples

We found that the NPCs in *Dishonored* engage in paths that include a lot of waiting. When removing path segments that do not include waiting (i.e. the guard did not stop patrolling between the last time it was observed and the first time it was seen again), the path segments generated by our tool is almost always accurate. Results depend on the geometry of each level, but in general, as observed in many gameplay videos [36,41,46,57] characters in *Dishonored* cover rather small areas of the map, which, if we limit during the modelling process, would yield very accurate guesses for the missing path segment.

The results are very similar to the ones obtained from our artificial paths experiments in non-waiting scenarios (i.e. when gaps do not include wait). However, for gaps that do include extensive waiting, the algorithm could not accurately determine where the guard waited, unless interest points were defined, and our long/interesting path generation algorithm generated paths that were more dissimilar to actual path.

We notice that thus far in all the cases we observed based on actual gameplay of *Dishonored*, generating the shortest simple paths that include waiting at interest points yield the most accurate predictions for the majority of instances. Additionally, we can use area and volume to assess the difficulty of the level, as the greater the area and volume, the more possibilities there are as to the whereabouts of the NPC.

4.4.2 Aragami

To gather additional data on NPC guard movements, we examined another stealth game *Aragami* [35], where the player plays in third person as a character that needs to infiltrate and defeat the army of Light.

This game is of interest to us because of the seemingly complex levels filled with guards
Figure 4.34 Series of screen captures showing a gap segment created by player distancing themselves from the guard to go around an obstacle before seeing the guard from the other side of it.
4.4. Real Game Examples

Figure 4.35 Gap from Figure 4.34 filled in using the tool. Left picture represents what the user observes and the right shows one of the generated paths.

Figure 4.36 Results of the level in terms of area (17 unit$^2$) and volume (28 unit$^3$) in the level depicted in Figures 4.34 and 4.35.

at every corner. The main character, required to stay within shadows, has a limited field of view on the guards, creating multiple gap segments as it is not possible to observe all of them at once at all times, as seen in Figure 4.37.

Upon observing multiple guards we notice that, although a greater number of guards patrolling adds complexity for the player, individual guard movement cycles are short and repeated. There is a great amount of waiting involved, and waiting points do not change. This provides additional insight on the way enemies move in stealth games: they should be predictable enough for the player to solve the level.

We selected one random guard as an example to introduce gaps in its path. Figure 4.38 shows in parallel both a screen capture of the guard in the level (a) and a representation of
4.4. Real Game Examples

Figure 4.37 Screen capture of gameplay in Aragami, where the player needs to avoid multiple guards (on different paths) at once, and must stay in the shadows to prevent being detected. Two guards are visible, while 3 others are in vicinity, but out of sight.

its full movement cycle (b), where black numbers indicate the time elapsed during that path segment, and red numbers coincides with elapsed wait time observed at that wait point.

Using this complete observable path, where black solid dots represent wait points, we attempted to fill in gaps that were introduced on this path. Since the actual path taken by the guard is continuous and does not follow Manhattan movement, the approximations using 4-way movement is highly inaccurate, as the calculations for shortest path yields a much longer path than reality.

To mitigate this issue, we modified our model to allow diagonal traversal between tiles, into an 8-way model. Although still non-continuous, it provides more freedom of movement that renders our estimations and representation of observations more accurate.

Although 8-way movement model introduces other issues such as clipping corners of obstacles, or cutting between obstacles that touch at the corners, the generated paths seen in Figure 4.39 of the 8-way model are more realistic, and the amounts of time taken to travel the generated paths in the 8-way are closer to reality than those generated by the 4-way model.

Using Volume to Measure Difficulty. For this level, we also used volume to compare different gaps that were dropped from the complete path data gathered from Aragami. We
4.4. Real Game Examples

(a) Screen capture of a guard, and the approximation of the path observed by the player in red.

(b) The path translated to a grid. The solid line indicates the path taken and the dots show locations of where the NPC waited.

Figure 4.38 A guard’s path in Aragami.
4.4. Real Game Examples

Figure 4.39 Screen captures comparing five examples of select path generation results between 4-way and 8-way models. White Lines represent known paths, blue lines are generated paths, between start (green square) and end (blue square) of gap segment.
chose random possible positions for gap segments along the path to determine level difficulty based on the different segments dropped from the complete path. Although this is not an exhaustive comparison, and the numbers are example specific, it still provides insight on the growing size and areas as gap size increases. The results are compiled in Table 4.7, and they show data from a random possible very short, short, medium and large gaps dropped from the 8-way model in the Aragami example. Note that there is no possible gap that is large yet does not include waiting, and thus the value is not computed and is missing from the table.

### Table 4.7

<table>
<thead>
<tr>
<th>Gap Size</th>
<th>Gap includes waiting</th>
<th>Gap does not include waiting</th>
</tr>
</thead>
<tbody>
<tr>
<td>Very Short Gap ($\Delta t = 4$)</td>
<td>14</td>
<td>5</td>
</tr>
<tr>
<td>Short Gap ($\Delta t = 6$)</td>
<td>46</td>
<td>11</td>
</tr>
<tr>
<td>Medium Gap ($\Delta t = 9$)</td>
<td>276</td>
<td>38</td>
</tr>
<tr>
<td>Large Gap ($\Delta t = 14$)</td>
<td>487</td>
<td>-</td>
</tr>
</tbody>
</table>

### 4.4.3 Conclusions

NPCs move in predictable ways, and that is shown across the multiple games that we examined. Whether they are played in first person, third person or orthogonal view, surprisingly, we noticed that most NPCs in all observed games have very short and simple cycles as their patrolling sequence. Although most players will stand and observe an enemy for at least a short amount of time before proceeding, it is rare that a player would follow an enemy for long periods of time. When a player does follow an NPC for a long period of time, or waits to observe the full movement cycle, we notice that these NPCs tend to cover surprisingly small areas, and tend to lengthen the cycle with long wait periods at either ends of their patrol path, and will patrol back and forth. We saw examples of this behaviour in *Dishonored*, *Aragami*, *Pillars of Eternity*, and *Thief*.

Our exploration of stealth games revealed that, although we may think that NPCs may take interesting paths, or elaborate looping paths, in reality, NPCs rather take simple paths
4.4. Real Game Examples

that include backtracking and generous wait times. Players expectations may or may not reflect that, because players only observe small segments of NPC behaviour, and may imagine the NPCs taking more elaborate paths than they actually are.
Chapter 5
Conclusions and Future Work

Previous playthroughs have an obvious impact on how a player approaches and understands a game: incomplete and partial knowledge is acquired, aggregated, and then applied to improve performance or further progress—games are frequently attempted more than once, and understanding the learning curve is an important direction in improving design. Modelling such knowledge in full generality has obvious difficulty. Our work defines an approach to modelling observation-based knowledge of NPC movements that offers ample flexibility to represent different kinds of player expectation, and can be used within a tool for exploring how level design and prior knowledge may interact. The user study confirms players have constrained expectation of NPC movements, that is also affected by prior experience.

5.1 Future Directions

A number of interesting future directions are possible from our work. Actual game integration with player tracking data would of course be useful, and would let us validate that our computed expectations correlate with player behaviours. Additionally, the tool can benefit from supplementary implementations, as well as more game examples and scenarios.

Tool Improvement. The algorithms of the tool can be improved by including computation of negative space knowledge (i.e. remove path segments that include spaces where the
5.1. Future Directions

player observed the NPC was not present at certain times). Optimizing its computations to speed up the visualization can be done by replacing the game objects with other visual rendering methods.

Movement models other than Manhattan or 8-way can be implemented and would be helpful in representing observations more accurately, especially continuous movement.

**User Study.** Our study can be further reinforced by conducting a more extensive user study. By tracking what the player does or observes when playing different game scenarios or levels will provide better actual gameplay data, that will give further insight on what players retain from observing NPC movement.

**Other Real Game Examples.** It would also be interesting to explore other games that require stealth or NPC path prediction, particularly games that involve guards with longer patrolling cycles, or with interesting level geometry. Based on the games that we analyzed, we noticed that in theory the way we apply path prediction on existing NPC do work to certain extents but it is game-specific and we will need more examples to show that NPCs indeed tend to take simple and shortest paths.

Other games such as *Pillars of Eternity* [38] and *Metal Gear Solid V* [30] involve guard avoidance behaviour, and can offer further interesting insight. In *Pillars of Eternity*, similar to *Dishonored*, guards patrol dungeon areas as the player attempts to explore and complete quests. Unlike *Dishonored*, where the player has limited first-person perspective and view of the map, *Pillars of Eternity* allows players to have a limited third-person view of the places surrounding the player in-game character (with further spaces darkened) which may provide better data for partial observation.

**Ambiguity in Player Observations.** Modelling knowledge is a difficult task, as there are infinite factors and circumstances that influence the way a human player makes their decisions or learn a new skill. Player knowledge is far from uniform and impossible to perfectly categorize. We are also interested in incorporating ambiguity into the model in future work, reflecting imprecision and lossiness in player observation. Breaking down the path prediction problem by modelling player path observations, and building a tool that
allows developers to simulate possible logical expectations enables game designers to have a better understanding of how a player may think and learn during a playthrough of a stealth game.

NPCs are becoming increasingly “intelligent” and utilize adaptive methods to interact with players. We can thus expand pathing flavours by including more path features beyond waiting, simple, or interesting. Although it is difficult to enumerate all potential path properties, which in any case would be game-specific, an automatic approach to capturing different path features by analyzing known paths would add a lot to our existing model.

Use of Player Knowledge. There are also possibilities of using this knowledge model of what players know of NPCs to improve on existing NPCs to make gameplay more interesting or difficult. By predicting what a player may expect of the NPCs movement, in many cases, the NPC can challenge the player’s expectations by changing its path and moving in unpredictable ways. Arguably, games are fun because the player hopes to win by outsmarting the NPCs, and if the enemies were to become “too smart” or unbeatable, games may be frustrating to play. The key is to use the data on the players’ knowledge wisely.

5.2 Conclusion

Our work explored ways to model a player’s knowledge on stealth levels based on the player’s partial observations, and we can conclude that it is not trivial to do so. However, we did find that players do have preferences and preconceived expectations of NPC movement. Through our small scale survey, we found that players expect NPCs to follow simple routes, and assume they fill in gaps using a shortest path approach. We also concluded that there is no preference between different longer, more complex paths, such as ones with loops and backtracking. Future work may include gathering background data of games those surveyed may have played, as well as investigating whether players expect different behaviour when observing in a first person view. Although the amount of information that can be gathered by tracking a player movement can be extensive, it can ultimately be reduced to a pathing and visibility problem that remains broad and complex with many opportunities for potential further investigations.


[9] K. Bringmann. Why walking the dog takes time: Frechet distance has no strongly
subquadratic algorithms unless SETH fails. In Proceedings of the 2014 IEEE 55th
Annual Symposium on Foundations of Computer Science, FOCS ’14, pages 661–670,
Washington, DC, USA, 2014. Institute of Electrical and Electronics Engineers.


In N. Xiao, M.-P. Kwan, M. F. Goodchild, and S. Shekhar, editors, Geographic Infor-


[13] M. de Aguilera and A. Mendiz. Video games and education: (education in the face of
a “parallel school”). Computers in Entertainment (CIE) - Theoretical and Practical

[14] E. S. de Lima, B. Feijó, and A. L. Furtado. Player behavior and personality modeling


points required to represent a digitized line or its caricature. Cartographica: the
international journal for geographic information and geovisualization, 10(2):112–

In Proceedings of 12th International Conference on Pattern Recognition, volume 1,

[18] D. Duque, H. Santos, and P. Cortez. Prediction of abnormal behaviors for intelligent
video surveillance systems. In 2007 IEEE Symposium on Computational Intelligence
and Data Mining, 2007.


A.1 Survey Questions and Responses

Survey respondents were prefaced with the following, and must agree to the terms before continuing on with the survey questions.

Purpose of the study. In this study, we are interested in observing assumptions players make on non-player characters’ (NPC) movements. In each question, you will be given a series of situations, each showing an animation of a NPC’s behaviour. These observations will be partial, as there will be gaps of time where you will not know the whereabouts of the character. Once you have completely watched the animation in each question, select the path you think the player most likely took. There are no wrong answers.

Time. This study should take less than 10 minutes. Dissemination of Results We expect to disseminate the results of this project through master’s thesis, conference presentations and publications in conference proceedings and journals, including online appendices. We may publish the results online as well. Confidentiality The stored data from this survey is completely anonymous and no personal information is asked. Data such as IP address or web browser type will not be stored. No other identifiable data is collected either. The data is stored privately. Any individual who wishes to view the responses will require granted
access by a principal investigator. Responses will not be published, only the statistics and aggregate results will be available to the public.

**Participation.** Your participation in this study is entirely voluntary. You may refuse to participate or withdraw from the study at anytime, until you submit your data. If you shall feel discomfort of any sort you may cease participation in the online study procedures and your data will be withdrawn from the study. You can take as many breaks as you wish and return at any time, as there is no time limit. The data is only collected at the end of the survey, which implies that once sent, your response cannot be withdrawn.

## A.1.1 Survey Questions

**Observing Non-Player Characters (NPC).** In video games, often times, non-player characters (NPCs) play an important role in the player’s success at achieving their main goal. In our research we look at how a player may predict NPC movements based on previous observations.

![Figure A.1 Example of a game where player must avoid patrolling enemies that are non-player characters (NPC).](image)

**How to answer the questions in this survey.** First, in the question, we will show you
A.1. Survey Questions and Responses

a level from the top-down view, where you, the player, will make a partial observation. There is a TIME in the upper left corner and a progress bar at the bottom showing the time progression. The animation replays indefinitely. Observe the NPC (white circle), and during the time between the green (last seen) and blue (first seen again) square, think of which path the NPC could have likely taken, and then choose one that is most likely based on your observations.

Figure A.2 Figure explaining the different elements of the animated survey questions and format.

The following Figure A.3 show the first two questions from the survey.
Introducing Waiting. NPCs can sometimes engage in waiting behaviour, or may choose to take longer paths instead. Waiting is represented by the "dot" staying on one tile for longer than a single time frame. You can imagine guards stopping to investigate, look around, or for other reasons. Example of waiting behaviour: NPC pauses on certain tiles.
A.1. Survey Questions and Responses

3. Based on the following observation, which path did the character most likely take? *

(a) Question 3

4. Which path did the character likely take? *

(b) Question 4

Figure A.4 Questions 3 and 4, following the introduction of waiting.
5. Which path did the character likely take? *

**Figure A.5** Question 5.

**Longer Observations.** The following Figures A.6, A.7, A.8, A.9, A.10 are longer paths that involve more observation.
6. Which complete path did the character most likely take? *

Figure A.6 Question 6.
7. Which complete path did the character most likely take? *

![Figure A.7 Question 7.](image)
8. Which path did the character most likely take? *

- Option 1
- Option 2

**Figure A.8** Question 8.
9. Which path did the character most likely take? *

![Figure A.9 Question 9.](image-url)
A.1. Survey Questions and Responses

Finally, the last question asked “Do you play video games?” and given the choices: “Rarely”, “Occasionally”, “Often” and “Prefer not to say.”

**A.1.2 Survey Results**

Below in Tables A.1, A.2 and A.3 are the anonymous responses submitted by users through our survey. Time stamp is generated automatically, and columns 1-10 were questions asking which path the user expected the NPC to take, based on an animation, and the last column asked the user how often they played video games.
### Table A.1 Survey Responses

<table>
<thead>
<tr>
<th>Timestamp</th>
<th>1</th>
<th>2</th>
<th>3</th>
<th>4</th>
<th>5</th>
<th>6</th>
<th>7</th>
<th>8</th>
<th>9</th>
<th>10</th>
<th>11</th>
</tr>
</thead>
<tbody>
<tr>
<td>6/11/18 19:41</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6/11/18 20:11</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6/11/18 20:34</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6/11/18 20:40</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6/11/18 21:01</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6/11/18 22:07</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6/12/18 0:53</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6/12/18 7:35</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6/12/18 9:18</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6/12/18 13:52</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6/12/18 14:18</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6/12/18 14:20</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
</tr>
<tr>
<td>6/12/18 14:29</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6/12/18 14:35</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6/12/18 14:47</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6/12/18 14:49</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6/12/18 14:51</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6/12/18 14:57</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6/12/18 15:03</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6/12/18 15:05</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6/12/18 15:13</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6/12/18 15:15</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6/12/18 15:15</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6/12/18 15:19</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6/12/18 15:28</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6/12/18 15:38</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6/12/18 15:51</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6/12/18 15:52</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6/12/18 16:09</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6/12/18 16:10</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td></td>
</tr>
<tr>
<td>6/12/18 16:15</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6/12/18 16:27</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td></td>
</tr>
<tr>
<td>6/12/18 16:46</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td></td>
</tr>
</tbody>
</table>

116
### Table A.2 Survey Responses (continued)

<table>
<thead>
<tr>
<th>Date/Time</th>
<th>A</th>
<th>B</th>
<th>B</th>
<th>B</th>
<th>B</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>2</th>
<th>1</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>6/12/18 17:00</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>Often</td>
</tr>
<tr>
<td>6/12/18 17:29</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>Occasionally</td>
</tr>
<tr>
<td>6/12/18 17:59</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>Often</td>
</tr>
<tr>
<td>6/12/18 18:03</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>Often</td>
</tr>
<tr>
<td>6/12/18 18:35</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>Often</td>
</tr>
<tr>
<td>6/12/18 18:36</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>Often</td>
</tr>
<tr>
<td>6/12/18 18:42</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>Often</td>
</tr>
<tr>
<td>6/12/18 19:05</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>Often</td>
</tr>
<tr>
<td>6/12/18 19:07</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>Often</td>
</tr>
<tr>
<td>6/12/18 19:08</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>Occasionally</td>
</tr>
<tr>
<td>6/12/18 19:11</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>Often</td>
</tr>
<tr>
<td>6/12/18 19:14</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>Often</td>
</tr>
<tr>
<td>6/12/18 19:17</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>Often</td>
</tr>
<tr>
<td>6/12/18 19:38</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>Often</td>
</tr>
<tr>
<td>6/12/18 19:38</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>Often</td>
</tr>
<tr>
<td>6/12/18 19:58</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>Occasionally</td>
</tr>
<tr>
<td>6/12/18 20:03</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Occasionally</td>
</tr>
<tr>
<td>6/12/18 20:15</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>Often</td>
</tr>
<tr>
<td>6/12/18 20:15</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Occasionally</td>
</tr>
<tr>
<td>6/12/18 20:18</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Often</td>
</tr>
<tr>
<td>6/12/18 20:18</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>Often</td>
</tr>
<tr>
<td>6/12/18 20:30</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>Often</td>
</tr>
<tr>
<td>6/12/18 20:33</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>Rarely</td>
</tr>
<tr>
<td>6/12/18 20:44</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>Often</td>
</tr>
<tr>
<td>6/12/18 20:46</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>Often</td>
</tr>
<tr>
<td>6/12/18 21:02</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>Often</td>
</tr>
<tr>
<td>6/12/18 22:32</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>Often</td>
</tr>
<tr>
<td>6/13/18 1:32</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>Often</td>
</tr>
<tr>
<td>6/13/18 1:45</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>Often</td>
</tr>
<tr>
<td>6/13/18 2:11</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>Occasionally</td>
</tr>
<tr>
<td>6/13/18 4:00</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>Often</td>
</tr>
<tr>
<td>6/13/18 4:24</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>Often</td>
</tr>
<tr>
<td>6/13/18 4:30</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>Often</td>
</tr>
<tr>
<td>6/13/18 4:41</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>Often</td>
</tr>
</tbody>
</table>
## Table A.3 Survey Responses (continued)

<table>
<thead>
<tr>
<th>Date/Time</th>
<th>A1</th>
<th>A2</th>
<th>B1</th>
<th>B2</th>
<th>B3</th>
<th>B4</th>
<th>B5</th>
<th>B6</th>
<th>B7</th>
<th>B8</th>
<th>B9</th>
<th>B10</th>
<th>Response</th>
</tr>
</thead>
<tbody>
<tr>
<td>6/13/18 4:48</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>Occasionally</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/13/18 6:31</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>Often</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/13/18 6:34</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>Often</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/13/18 9:11</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>Often</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/13/18 9:43</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>Often</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/13/18 10:56</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>Often</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/13/18 12:56</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>Often</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/13/18 13:24</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>Often</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/13/18 13:45</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>Often</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/13/18 17:56</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>Often</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/13/18 19:19</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>Often</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/13/18 19:22</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>Often</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/13/18 19:26</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>Often</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/13/18 20:20</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>Occasionally</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/13/18 20:51</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>Rarely</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/13/18 20:51</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>Occasionally</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/13/18 21:11</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>Rarely</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/13/18 21:12</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>Often</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/13/18 21:16</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>Occasionally</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/13/18 21:23</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>Rarely</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/13/18 21:24</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>Often</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/13/18 21:31</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>Often</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/13/18 21:52</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>Occasionally</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/13/18 22:11</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>Often</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/13/18 22:14</td>
<td>B</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>Rarely</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/13/18 23:05</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>Often</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/14/18 7:44</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>Often</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/14/18 8:49</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>Often</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/14/18 9:26</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>Often</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/14/18 11:16</td>
<td>B</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>Occasionally</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/14/18 17:48</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>Often</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/14/18 20:19</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>Occasionally</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/15/18 3:56</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>Often</td>
<td></td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/15/18 16:32</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>1</td>
<td>1</td>
<td>1</td>
<td>2</td>
<td>2</td>
<td>Occasionally</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/16/18 13:03</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>2</td>
<td>2</td>
<td>2</td>
<td>1</td>
<td>1</td>
<td>Rarely</td>
<td></td>
<td></td>
</tr>
<tr>
<td>6/18/18 14:30</td>
<td>A</td>
<td>A</td>
<td>B</td>
<td>B</td>
<td>A</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>1</td>
<td>2</td>
<td>Often</td>
<td></td>
<td></td>
</tr>
</tbody>
</table>