From A to B: An algorithmic approach to circulation inside buildings

Arash Naderpour

A Thesis

submitted in partial fulfillment of the

requirements for the degree of

Master of Science in Architecture

University of Washington

2018

Thesis Committee:

Brian R. Johnson, Chair

Alex T. Anderson

Program Authorized to Offer Degree:

Architecture

University of Washington

Abstract

From A to B: An algorithmic approach to circulation inside buildings

Arash Naderpour

Chair of Thesis Committee:
Brian R. Johnson, Associate Professor
Architecture

Circulation design is a challenging part of the architectural design process. Architects traditionally address these challenges based largely on rules of thumb, personal experience, and simple tools. The modern environment, with greater computational power available, invites a more careful analysis and consideration. The purpose of this thesis project is to illustrate how algorithmic tools can be used to simulate human behaviors during the architectural design process or after construction of the building to improve building's performance. To achieve this objective, analysis approaches related to congestion, emergency egress, and wayfinding inside architectural space were surveyed. Then an algorithmic toolkit was developed based on an artificial intelligence pathfinding algorithm called Theta*. Finally, several application prototypes were

created to illustrate application of the toolkit to (a) evaluating positive and negative implications of congestion in the design of the buildings, (b) assessing congestion during emergency egress situations, and (c) assist individuals navigating complex buildings.

Table of Contents

List of Figures

ACKNOWLEDGEMENTS

I would like to express my sincere gratitude to my advisors Prof. Brian Johnson and Prof. Alex Anderson for the continues support of my thesis study, for their patience, motivation, and immense knowledge. Their guidance helped me in all the time of research and writing of this thesis. I could not have imagined having better advisors and teachers for my Master study.

My sincere thank also goes to my family for supporting me spiritually throughout my whole education, especially during writing of this thesis and my life in general.

DEDICATION

This thesis is dedicated to my beloved parents, Farideh and Reza Naderpour who have always been a source of inspiration, encourage, and stamina to undertake my difficulties and to face the eventualities of life with strong will and hard work.

1 – Introduction

This chapter presents an introduction to this Master of Science research. In chapter 1.1, the motivation of this research is explained by presenting several challenges that indoor wayfinding and indoor navigation present for designing and maintaining architectural spaces and introducing some possible solutions for the challenges.

1.1- Motivation

One of the problems of urban growth, with its proliferation of roads and rapid expansion of complex buildings, is increasingly difficult wayfinding. Be it for work, shopping, healthcare or recreation, it can be frustrating and time consuming for people to find their way in unfamiliar urban environments or in complex buildings. Some individuals in such circumstances are overwhelmed by a feeling of ineptitude which they attribute to a missing "sense of orientation;" others blame the architects for designing confusing settings and the graphic designers for providing vague signage. Whoever the target of complaint, the irritating and unpleasant experience of getting lost influences people's general attitudes towards the environment. Architectural settings spoiled with way finding difficulties are not appreciated by anyone (Passini 1996). At the urban and regional scale this problem prompted creation of navigation tools like Google Maps or Apple Maps, which provide guidance for people to reach their destinations more easily. Over the past few years these tools have become established for route planning, but they are mainly designed for outdoor environments, since they employ satellite-positioning technologies (GPS).

Although much research has been done on exterior pedestrian navigation systems in the past few years, way finding inside complex buildings like hospitals, airports, malls, office buildings, universities and commercial centers with complex floor plans has received far less attention by researchers and remains a challenging topic. Individuals must depend on their general knowledge about the structure of buildings, or their previous experience, and the visual inputs, like posted

maps, which they encounter to find their way while moving inside most buildings. They must do this without the support of external navigation assistance, so they often fail to find their way quickly in complex buildings. Some corporations have mentioned that the irritation caused by wayfinding difficulties not only arouses negative feelings for the physical setting, but that it also effects people's impressions of the corporation itself and the services provided in that setting (Carpman and Grant 2016). Therefore, a well-established tool offering interior navigational assistance would be beneficial both for users of a space (Makri, Zlatanova and Verbree 2015) and the organizations that own them.

Aggregate way finding, which we refer to as *circulation*, has important implications in the design of buildings as well. Uncomplicated circulation inside a building reduces the time spent consulting confusing information displays and even liberates individuals from time consuming interpretation of vague direction information given by staff. It solves some building efficiency issues and has financial impacts that are not easy to calculate. However, there is more to good circulation design than efficient movement; one occupant-level spatial quality that interior architecture and space planners are interested in is the concept of *congestion*. This can be both a positive and a negative quality in architectural space, as it can encourage productivity (*productive congestion*) in some environments and evoke displeasure in others. Despite extensive research into the role of both types of congestion, the lack of a specific tool that evaluates a designed floor plan based on this quality is perceptible.

Another primary problem that difficult wayfinding creates is accessibility. Individuals often avoid environments in which they know they will get lost. Contemporary planners avoid building complex shopping centers with lots of twists and turns, so that people would spend more time wandering around and theoretically shop more due to countless objections and negative evaluation of commercial productivity. People with physical or sensory impairments have even harder times in buildings with complicated circulation, and inappropriate wayfinding tends to be especially challenging for them. These difficulties can create architectural barriers of a psychological nature which can be just as disruptive as physical barriers (Carpman and Grant 2016).

Building safety is also associated with wayfinding. Emergency evacuation is much more difficult in confusing settings. In emergency situations wayfinding decisions must be made quickly and problem-solving behavior may be confounded by anxiety-induced stress. Lack of consideration for easy and fast access to emergency facilities and exits during the initial stages of architectural design by architects can increase the total evacuation time for users of the building during

emergency situations like fire, earthquakes, terrorist threats, and so on. Moreover, without predictions of the best routes that first responders must take to access various parts of complex buildings, the performance of the first responders and emergency teams is reduced because of increased time required to provide services for individuals inside buildings.

Wayfinding influences not only navigation in existing buildings, but also key design parameters during architectural design process, especially for large public buildings. Many standards have to be met related to the circulation and the paths that users should take in order to access different parts of the building. For example, there are fire codes related to maximum distance of each space to the closest fire escape, or the maximum distance of each room to the entrance of the emergency section of a hospital. In addition, analyzing the accessibility and circulation of a floor plan on the basis of design efficiency and convenience for users is beneficial to the architecture design process.

A toolkit that provides circulation analysis would be extremely helpful, since it could provide insights for architects about, for example, the best place to establish amenities inside an office based on optimized routes, or about locations that make it convenient for people to collaborate with each other, or how long will it take to travel from one place to another inside a building. The toolkit could also be beneficial for occupant health by calculating how many calories will be burned during travel from one part of a building to another or by demonstrating the desirability of using stairs in a building. These benefits would also contribute the efforts of designers and building owners seeking Living Building Certification.

1.2- Problem Statement

The spread and popularity of algorithmic tools has changed the way architects design (Kalay, 2004). Architectural design processes have been increasingly developed to foster more thorough and precise considerations of numeric criteria like stiffness of a building's structure, or the amount of daylight that a space receives. This is visible within contemporary computational platforms, as well as organizational frameworks, which try to implant the variations and invariants related to a building's materiality, its internal spatiality, and its external relations with a solid process structure (Ahlquist, Sean and Menges 2011). Simulation techniques are a critical

part of the current digital design framework and can be applied to building design, although they are mostly in the domain of engineering.

To analyze human-level concerns, such as how the space feels, how the space will be utilized, or how the layout of the space matches the needs of the program, architects often rely on agent-based crowd simulation models that predict the movement of people in a space. The results of these analyses are extremely accurate; however, since they must be calculated dynamically (over a series of time steps), they are also computationally expensive and time consuming, often taking hours or even several days to return the results for a single design (Nagy, et al. 2017). Although these analyses may be suitable to validate some design projects based on their functionality and needs, they may not become part of the architectural design framework due to their expensive required resources (time and computing power).

On the other hand, there are three main problems that make developing an indoor navigation tool challenging in contrast to outdoor *PNS* (Pedestrian Navigation Systems):

1. Lack of reliable position information inside a building (Li, et al. 2012).

2. Lack of proper data (maps) designating the locations of the obstacles, routes, walls, and destination inside a building.

3. Complexity of indoor space configuration and freedom of users' movement inside the building.

Availability of satellite-positioning technologies (GPS) in outdoor environments provides the possibility of locating the user with accuracies to a few meters radius in outdoor environments, depending on the strength of the GPS signal. However, these technologies are not available for indoor spaces. This is due to the fact that signal to noise ratio of satellite signals are usually close to the threshold for detection of GPS receivers, so GPS signals are not detectable indoors, specially behind concrete walls. Apart from that, high frequency signals like GPS (1500 MHz) do not propagate inside buildings, since they will get absorbed much more by objects.

Unlike outdoor navigation systems where location information provided by GPS is paired with rich databases of road networks and obstacles provided by satellite-derived map data to produce navigation, indoor spaces lack such information and database, hence the path networks in indoor environments must be computed. Such path finding algorithms often rely on *graph theory* and graph representations, which is an area in mathematics that studies mathematical processes and data structures used to model pairwise relations between objects. Therefore, generating a decent

network and data model that simplifies a building structure and floor plan is crucial for deriving the overall connectivity of spaces in a building and for representing positions of objects, obstacles and walls within the environment. Producing such a model for various indoor environments automatically is one of the challenges of indoor PNS.

While individuals' movements are restricted by massive obstacles like buildings or roads in exterior environments, the only objects that limit people's movements inside a building are walls and furniture, hence they have unlimited paths to take in order to go from one place to another inside the space between the interior borders of buildings. The freedom of human movements and complicated paths they take inside a building presents another challenge to developing indoor navigation systems, but it is also one of the reasons that various search algorithms have been developed for wayfinding purposes. The paths generated by *Artificial Intelligence* search algorithms have to maximize the usability and successful navigation while minimizing the chance of a user getting lost. They also have to mimic human movements. Developing and using more sophisticated and smart pathfinding AI algorithms that simulate human movements are essential to developing a practical indoor navigation system.

There are additional challenges in developing an indoor PNS. The path network construction process may not be easy and straightforward due to complex indoor space configurations. Considering the rich semantic information invested in building components like indoor furniture, or elevators or stairs that provide a choice for users who want to move from one floor to a space on another floor, and the lack of rich semantic models with room adjacency information in them are some of the other challenges of indoor navigation systems.

2 - Background

In the past decades, many researchers from multiple fields have studied PNS. This review divides that work into four main categories. First, many studies have been done in computer science and robotics on generic search algorithms. Second, studies about extracting interior information about a building from a CAD or BIM model, or research on building the 3D model of the physical environment in real time, using a camera or laser scanner. Third, research done on locating a user

inside a building in real time. Fourth, studies related to representation of indoor location and navigation information to users of indoor space. This review introduces some of the most important studies in each of the four categories and explores their pros and cons in order to support development of a better indoor navigation tool for specific tasks.

2.1 – Pathfinding Algorithms

Any algorithm that solves a search problem is called a "search algorithm" in computer science, which means the algorithm extracts information stored within some data structure or calculated in the search space of a problem domain. It is possible to categorize search algorithms based on their mechanisms of search. Recent developments in artificial intelligence provide the possibility to solve problems that were unsolvable with exhaustive enumeration search algorithms. In addition, current AI algorithms are many times faster and more reliable than classical search algorithms. Finding the optimal route between two points (called "routing") is one of the categories of search problems that is now solvable cheaply and reliably thanks to advancements of artificial intelligence.

The standard routing AI algorithm is the *Dijkstra algorithm* developed by E. W. Dijkstra (Dijkstra 1959). This algorithm finds the *shortest path* between nodes in a graph. The algorithm labels each position with its distance from the start node along the best-known path. Initially, no path is known, so all the nodes are labeled infinity. As the algorithm proceeds and the paths are found, the labels may change, reflecting the better paths. Simply put, the algorithm calculates the fastest path between two nodes as a function associated with the cost of travelling.

The main drawback of the original Dijkstra algorithm is that it explores an unnecessarily large search area. This led to the development of heuristic search algorithms that instead of searching all nodes, look for shortest path in the direction of the destination node. These have names like *A*, D*, D*-Lite, Theta*, AP Theta*,* etc. Hart, Nilsson and Raphael 1968 developed the A* algorithm that avoids considering directions with non-favorable results and decreases computational time by finding the nodes on the shortest path based on their distance from the start node and distance from destination node.

Stentz (Stentz 1994) introduced the D* (Dynamic A*) algorithm, which searches an environment that is either completely unknown, or when only partial information about it is available, to find the shortest path between two nodes. The algorithm makes assumptions about the unknown parts of the environment and starts looking for the optimized path based on the initial assumptions. If new map information (such as previously unknown obstacles) are observed during the search, the information will be added to the map and, if necessary, a new shortest path will be computed from the last node found to the destination node.

One of the downsides of all the grid-based pathfinding algorithms like A*, D* and its variants (D*-Lite, Focused D*, etc.) is that paths formed by grid edges can be sub-optimal and unrealistic looking, because the possible headings are artificially constrained. As an improvement, Nash et al. (Nash, et al. 2007) developed the Theta* algorithm that searches grid-based models to find the shortest realistic looking path without constraining it to grid edges. The algorithm considers *line of sight* from node to node in order to find the optimal path. The realistic looking paths that Theta* algorithm finds are extremely close to human movements, so Theta* is an appropriate algorithm for human movement simulation.

There are a variety of routing algorithms available specifically for traversing road networks. For instance, a dynamic programming approach was developed by Bellman (Bellman 1958) to minimize the travel time between two points. In his approach the cost of traveling between a set of locations that are connected by road are time instead of distance. Also, a finite number of iterations is mandatory in his iterative algorithm.

2.2 – Indoor Navigation and Model

Zlatanova (Zlatanova and Baharin 2008) studied the movement of mobile rescue units in emergency situations in order to provide a solution for navigating on road networks. They assumed that the locations of the moving objects (rescue units) are recorded uninterruptedly in a Database Management System (DBMS) and organized according to a predefined spatial-temporal schema. They considered information factors related to generic groups named: *spatial*, *user*, and *event* information. Their emergency response network model of the navigation tool represented

the road network, the characteristics of the emergency event, information about the user, and the optimal route generated by the A* algorithm provided by the pgRouting extension of PosGIS.

Wu et al. (Wu, Marshall and Yu 2007) used A* and Dijkstra's shortest path algorithm to navigate through an "Intelligent Map" which is based on a new data structure called "Cactus Tree" to provide indoor navigation guidance for sightless individuals. Their approach consists of three components: Cell decomposition, Cactus tree-based path planning for indoor space, and A* based path computation. Cactus tree has pre-knowledge of relationships between indoor elements that Intelligent Map provides. Since this non-linear data structure is an ordered/sorted tree, the structural connection is richer than the simple "before" and "after" relationships among objects in sequences. The objects are "left", "right" and "branches (below or equal)" to other objects in this tree. Eventually, the path following algorithm used by them is based on *dead reckoning*. However, they integrated human factors, plus data about the flooring and furnishing structure beside the intentional path that was planned. Although their approach is promising for assisting visually impaired people to navigate indoor environments, unrealistic paths generated by the A* algorithm and visibility graph is one of the main drawbacks of the proposed approach.

Chen and Feng (Chen and Feng 2009) developed two fast flow control algorithms for real-time emergency evacuation of populous places such as shopping malls, subway stations, campus buildings, etc. to minimize the overall evacuation time based on the properties of those buildings. The first algorithm was developed for situations where there is no limit on the allowed number of evacuation paths, while the second algorithm was developed for situations where a limited number of evacuation paths are available. They used a network of "nodes" representing the locations inside the building (rooms, corridors, stairs, halls, etc.) and "arcs" representing the travel path between two nodes. Then, the Dijkstra algorithm was used to find the shortest paths from the given source location to each one of the exit doors. They calculated the traversal time of each path by dividing the length of each path by the average walking speed of people. Eventually, to illustrate real-time application of the developed flow control algorithm, they integrated a preliminary head-mounted augmented reality display (HMD) which tracks the position and orientation of the user in their evacuation system. Omitting all the indoor objects and furniture in order to abstract the navigation model, plus unrealistic paths generated by the Dijkstra algorithm are the main limitations of this approach.

Xu and Van Doren (Xu and Doren 2011) developed a "Museum Visitors Guide (MVG)" in the Greenfoot programming environment using an optimized A* algorithm to provide navigation guidance for museum visitors. Their application allows the user to find the shortest route to

his/her favorite artwork(s) from his/her current location using any computer that is conveniently located in the museum. The application reacts dynamically to the modifications of the floor plan, and/or locations of the exhibits, plus emergency situations that might make part of the currently suggested path invalid and regenerate a new path if it can. The application also allows the user to select multiple artworks and it generates the quickest route to cover them all. Finally, the application provides guidance for a user to avoid fire while he is inside the museum. One of the main disadvantages of this approach is that, navigation assistants (computers located inside the museum) are not capable of tracking user while he moves from one place to another, since they are statically located. Apart from that, their proposed navigation model is too abstract and did not consider indoor furniture and objects.

Xu et al. (Xu, Wei and Zlatanova, AN INDOOR NAVIGATION APPROACH CONSIDERING OBSTACLES AND SPACE SUBDIVISION OF 2D PLAN 2016) proposed an indoor pathfinding approach that takes obstacles of different shapes into account, subdivides the remaining navigable space using the *Delaunay* triangulation algorithm and generates a network, which provides realistic navigation by using a *Visibility Graph*. They processed the available indoor spatial information to generate the navigation network. In order to provide geometrical information about the indoor objects, they described them as points, lines and polygons. For instance, they represented doors by their midpoints, walls were represented as a line connecting the two ends, and indoor objects were represented as lines, triangles and polygons based on their size and location in the room. Next, they constructed the network of paths by subdividing the free space into triangles. The center of gravity of each triangle was chosen as a node in the network of navigation, and by connecting all the nodes together, excluding edges that cross the obstacles, the network of paths was generated. Figure 1 illustrates the process used in this study to generate the network of paths from a simple floor plan. Eventually they traversed the network in order to find the path from each node in the network to another node. One of the disadvantages of the proposed approach is that, the process of extracting the navigation model from the architectural documents and abstracting it is not completely automated. Apart from that, the computational process of generating the routes network by connecting the centroids of triangles is computationally expensive. In addition, the navigation model generated in this approach is extremely abstracted, and the paths generated by the algorithm are neither optimized nor realistic.

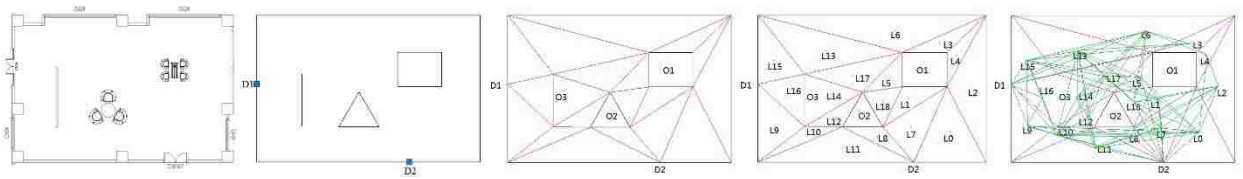

Figure 1: Generation of network of paths from a simple floor plan from left to right.

Xu et al. (Xu, Wei, et al., BIM-BASED INDOOR PATH PLANNING CONSIDERING OBSTACLES 2017) in another study used Industry Foundation Classes (IFC) to extract semantic, topological and geometric information related to indoor space and furniture and used it to develop a 3D indoor navigation system on a multi-floor building. They extracted geometrical information, like the floorplan outline, building components (e.g. tables, chairs, cupboards, beds and sofas), facilities and equipment (e.g. bathroom and kitchen furniture), as well as semantic data, like privacy level of the space, entry door status (if door is locked or not), "association between space and related building elements, and the relationship between an element and the spatial structure (IfcRelContainedInSpatialStructure)", and generated a grid subdivision based on the extracted information. In this way, they created a navigable network for each floor. They then produced the total navigation graph of multi-floors by connecting the graph of every floor together through staircases or elevators. Eventually, the Dijkstra algorithm was used for path planning avoiding the obstacles. The navigation model extraction process proposed in this research is automated and generates a model with acceptable levels of detail. However, the path finding algorithm used in this work is slower than most of the grid-base path finding algorithms. In addition, paths generated by the Dijkstra algorithm are unrealistic and not similar to human movements in most cases.

Diakité and Zlatanova (Diakité and Zlatanova 2016) have proposed a workflow for extracting free navigable space from a fully furnished 3D model, which can then be used to support pedestrian, robot, and drone navigation systems. The authors indicate that, while the existing approaches use simplified representations of the space and ignore the objects inside, their workflow models the indoor navigable space with enough complexity to precisely know where a subject can move inside the building. They presented two approaches for extracting the navigable space from the 3D model without ignoring the inside objects based on the format of the input 3D model of the building. Their first approach could be used to automatically extract geometrical and semantic information from an IFC input mode. The second approach could be used to extract information about the building that just contains a geometrical description in the form of 3D CAD models of their

interior design. One of the drawbacks of the proposed workflow is that the stability and reliability of the result depends on the quality of the input IFC or CAD model. Apart from that, their approach seems extremely promising for indoor navigation systems.

Zhu et al. (Zhu, et al. 2013) developed the Chinese standard of Multidimensional Indoor Location Model, which describes ontology of indoor location. The model corresponds to 3D concepts like CityGML and IndoorGML (Geography Markup Language). The goal of their study was to develop an exchange GML-based format for indoor navigation. They hypothesized that nonrepresentational indoor locations can be formally described using qualitative (semantic) and quantitative (geometrical) information. They argued that current indoor location services may be providing contextual noise and lack user-concerned data about the location. They proposed a richer, multi-dimensional indoor location model that represents absolute and relative geometrical and semantic information about indoor space for the user.

Liu and Zlatanova (Liu and Zlatanova 2011) developed a pathfinding algorithm that represents "natural movement of pedestrians" inside the floor plans of buildings with complex indoor structures. The proposed algorithm which is called "DOOR-TO-DOOR" is interpreted as the direct walking route from one door to the next visible door or the shortest feasible route between two invisible doors. The algorithm is a typical network-based algorithm that treats the doors as nodes and the rooms as edges of the network in contrast to network models that treat doors as edges connecting rooms (nodes). The algorithm is organized with a two-level approach. First, the shortest path between the location of the user and the final exit will be generated using the Dijkstra algorithm (Dijkstra 1959), this route describes which rooms and corridors must be passed in order to arrive to the final exit. Second, when a pedestrian arrives at the door attached to a room a shortest path inside the room will be generated using Dijkstra algorithm between the start door and the target door in the room without considering the interior obstacles (e.g. furniture, pillars, other pedestrians, etc.). Finally, the algorithm provides the door-to-door path obtained at the room level for the pedestrian according to the sequence of rooms obtained in the floor level. The presented work has several major drawbacks. First, no interior obstacles (e.g. furniture, columns, other occupants, etc.) are considered in this method and rooms were assumed to be completely empty. Second, running the path finding algorithm twice reduces the speed of the process and might make the algorithm useless in some design methods like *generative design* process where many design options will be generated algorithmically and evaluated based on a given design goal to discover one or more high-performing solutions for the design problem, plus rich semantic models that hold room adjacency information required for the first step of the algorithm are not currently available for all buildings. Finally, paths that connect doors to each

other are not reliable and efficient, since most of the time initial location or destination of the individuals is somewhere within the boundaries of a room not at a door.

Mortati (Mortari, et al. 2014) proposed a novel algorithm for extracting the so called "Geometric Network" graph, which holds both metric (i.e. Euclidean distance between nodes) and topological information (i.e. connectivity of nodes) for indoor navigation to overcome the weaknesses and constraints of the existing approaches in the literature. They assumed that a semantic geometric model of the building is available and developed their algorithm in a hierarchical two-level approach similar to Liu and Zlatanova's (Liu and Zlatanova 2011) approach. Firstly, a weighted graph that only includes inter-space connectivity was constructed. A shortest path algorithm (Dijkstra) was launched on top of the graph in order to find the sequence of spaces between a source and a destination (Figure 2).

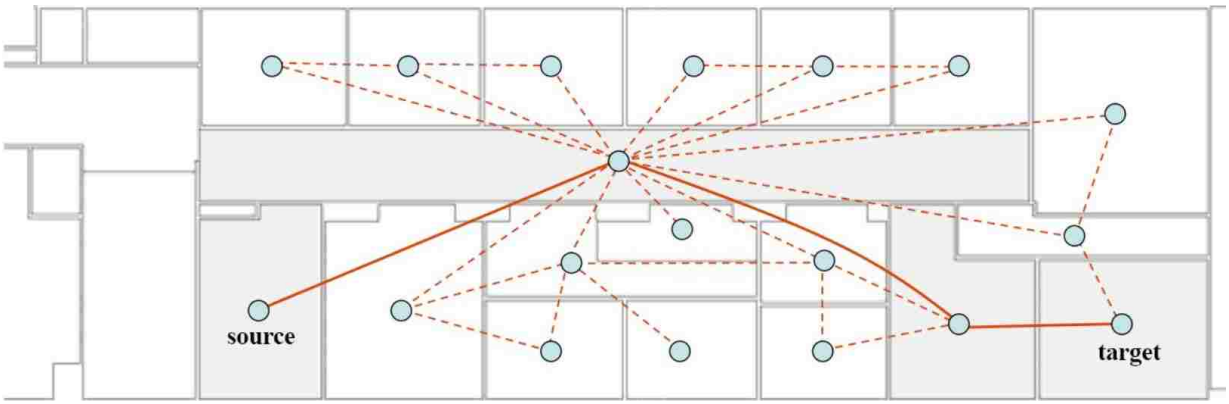

Figure 2: Sequence of spaces between start point and destination.

After determining the sequence of rooms, they generated the physical path incrementally room by room along the defined sequence of rooms. The algorithm that generates the physical path on the 2D floor plan consist of six geometrical operations:

1- Inward offsetting of the footprint of the space.

2- Sampling of the boundary of the originating polygon.

3- Computing a Constrained Delaunay Triangulation on the sample points.

4- Dividing each mesh face that has no constrained edges into three smaller triangles.

5- Assigning the nodes that represent topographic space in dual representation to the space they belong.

6- Linking nodes to each other based on adjacency of the topographic space they represent in original space.

In contrast to the "DOOR-TO-DOOR" approach that was mentioned before, the "Geometric Network" does not ignore indoor obstacles; however, room adjacency information is still required for the first part of the algorithm and paths generated by it are not optimized and are dissimilar to natural human movements.

Goetz and Zipf (Goetz and Zipf 2011) developed a model that represents indoor environments with topologic, semantic and metric information that allows nearly length-optimal routing in complex building structures (2D floor plan). They extracted the graph elements manually from floor plans and gathered additional semantic information for various functions of the graph. The authors also focus on related parts of a complex indoor space which are required for network construction like corridors and stairs. Additionally, they consider obstacles and special semantic areas inside rooms and how to incorporate those areas in the routing graph by adding additional nodes to the graph manually. One of the main advantages of this work compared to other research in this area is that the authors considered indoor obstacles and furniture; however, the process of adding nodes around obstacles manually makes the generation of the navigation model slow and not automated. In addition, the resulting paths are not optimal and appear unrealistic, especially around obstacles.

Nagy (Nagy, et al. 2017) proposed a concrete methodology by which architects can evaluate and quantify productive congestion within their architectural designs. First, they generated a traversal graph with evenly sampled nodes and overlaid it onto the floor plan in a way that it covers the entire bounding rectangle of the floor plan. They removed any edge of the graph that intersects with any of the indoor obstacles. Eventually, they connected the start and destination nodes to the traversal graph and performed a Dijkstra path finding algorithm on it to find the shortest path between pairs of start and end nodes. After finding the shortest paths between start and end nodes, all other nodes were labeled based on their level of participation in routes between start and goal nodes. In order to compute the metric to measure the total amount of congestion and its distribution through the space, they found the centroid of all nodes, weighted by their dissipated traversal values, then they connected this centroid value as a node into the analysis graph and computed the shortest routes from each node with traversal value higher than zero to the centroid

node. The authors suggested that the resulting "buzz" metric is the sum of the length of these routes, weighted by the traversal value of the starting node (Figure 3).

Although their method provides one single number for measuring the productive congestion inside a building, their method for creating the traversal graph is not efficient and effective. Hence, the shortest routes they used for measuring the congestion metric are not optimized and are dissimilar to human movement. Therefore, their analysis results might be different from what will actually happen in the real world.

$$buzz = \sum_{i=0}^{n} l_n * t_n$$

Figure 3: Congestion metric equation.

2.3 - Localization and Positioning

Location information related to the pedestrians and goods inside buildings is essential for indoor PNS. However, gaining spatial data inside a building is a challenging task. A variety of technologies have been used for positioning, localization, and mapping indoors in the absence of GNSS (Global Navigation Satellite System aka GPS). However, constraints in accuracy and coverage, plus expensive infrastructure requirements limit the existing solutions to only a few successful circumstances (Khoshelham and Zlatanova 2016). This section provides an overview of some of the most recent developments in sensor technology, and it offers a methodology for indoor localization and mapping.

There are many different technologies that have been developed recently for indoor positioning, such as ultrasound, infrared, Wireless Local Area Network, Bluetooth, Radio Frequency Identification (RFID), and Ultra-Wideband. The RADAR, LANDMARC, and Place Lab techniques are the most representative of these approaches. However, these schemes involve installation of additional hardware devices and complicated infrastructures. On the other hand, the

development of intelligent mobile terminals (smartphones and tablet computers), provide a handful of advanced technologies, (Wi-Fi, Bluetooth, and inertial sensors) that can be used for providing personalized indoor positioning information. The studies mentioned below focus on solving the indoor localization problem using inertial sensors, accelerometers and gyroscopes, which are available on most smartphones. The localization approximation in these studies is mainly based on the Pedestrian Dead Reckoning (PDR) method, which involves step length calculation using accelerometer data.

Chen (Chen, et al. 2015) developed an indoor positioning system by integrating the WiFi fingerprinting positioning method with PDR (Pedestrian Dead Reckoning) positioning method. They invented a new step counting algorithm called "auto-correlation" that "calculates the number of steps according to the similar acceleration of a pedestrian continuous motion". This algorithm reduces the effect of the phone's position and pedestrian's movement on the step count result. The motion of the user is separated to two scenarios: idle, which includes all static states, like standing up, sitting down, etc., and walking, which happens when body location changes while the phone is being carried. It also reduces the effect of phone's position, which is condition of the phone while being carried (i.e. it is being carried by hand or inside a pocket). In the Wi-Fi Fingerprinting localization approach, they suggested the K-means clustering algorithm to decrease the resource cost of the location algorithm and increase the time performance of the system without changing the localization accuracy. Outcomes of using the algorithm on an experimental setup indicated that it can reduce the error that instability of Wi-Fi signal creates in positioning result.

Diaz (Diaz 2015) claimed to present the first inertial step and heading navigation system that considers vertical displacements without using extra sensors, like barometer, GNSS, Wi-Fi access points, or having extra information, such as maps. The proposed pocket navigation system consists of two subsystems, hardware and software. The hardware subsystem consists of a magnetic and inertial measurement unit (MIMU). The software subsystem has two main parts. First, the orientation estimation algorithm, which consists of an unscented Kalman filter (UKF), whose states are the Euler angles roll, pitch and yaw and the biases of the gyroscope, and it utilizes the accelerometer and magnetometer to correct the orientation estimation. Second, the position estimator which consists of the step detector, the step length estimator, and the vertical displacement estimator. The proposed system utilizes the pitch angle in the three main algorithms of the position estimator. For evaluating the proposed system, it was tested in Deutsches Museum, which is a well-known museum in Munich. The result of the experiment shows that system has less than 1% error in detecting vertical displacement of the pedestrian. Eventually, they concluded

that using the pitch angle is recommended not only for detecting steps and calculating the length of it, but also for approximating the height of the steps of the staircase, which leads to estimating the height of floors. The main disadvantage of this system is lack of an accurate orientation estimation. Since the pitch and yaw angle directly influence the position estimation, it is required to count with an accurate orientation estimation algorithm to avoid errors.

Chen (Chen, et al. 2015) proposed a combination of RSS-based localization methods and PDR localization methods and developed a system that can overcome the drawbacks of each and provide positioning information with a precision of 1 meter by identifying indoor landmarks. They presented a WPL (weighted path loss) algorithm that omits the tedious manual process of collecting a training dataset for the machine learning algorithm used in the Wi-Fi fingerprinting approach. The WPL algorithm is also more suitable for resource-limited smartphones, since it does not rely on heavy load machine learning techniques. Therefore, they used the WPL method to provide accurate information about the initial position of the user. In order to overcome the accuracy problem of the PDR positioning approach on long distance, the authors leverage the landmarks, whose positions are known, as new starting points to restart the algorithm when the pedestrian reaches them. Landmark identification was done using their specific sensor patterns. They used accelerometer, magnetometer, gyroscope, barometer and Wi-Fi to identify landmarks, like turns, elevators, escalators, stairs, and doors.

Qian (Qian, et al. 2015) proposed a reliable real-time indoor localization method that uses inexpensive smartphone inertial sensors by considering the phone in four different conditions (texting, calling, in-hand, in-pocket), plus indoor vector graphs to address the PDR long range inaccuracy problem. Their positioning approach consists of three main components. First, the data preprocessing component, which calibrates, interpolates, and filters the retrieved data from accelerometer, gyroscope and magnetometer. Second, a PDR component, which calculates the position of the user by utilizing the output processed data of the first component. Third, the particle filter component, which is in charge of fusing the ultimate localization results for external usage by providing feedback data to rectify the step length and heading error from PDR with the constraint state of an indoor vector graph.

3– Theoretical Background

3.1- Architecture Circulation and Wayfinding

During the architectural design process, architects must pay close attention to movement and rest, which are concerned with circulation and wayfinding inside a building. Connectivity of spaces and how people flow through them are important criteria in architectural design. The way occupants or users of a building move through and interact with the building directly affects their feeling for the building and what they experience while they move inside it.

Wayfinding relates to the deliberate circulation of individuals and their mental ability to locate themselves in a setting (Passini 1996). And pathways or circulation routes must have two properties to be effective. First, they must be clear and unhindered. Second, they must provide the shortest walk between two points in the space. A good circulation design provides the possibility for people to move inside the building with ease and efficiency, without disturbing others, and without the sensation of getting lost. These characteristics are less important in residential and other small-scale projects that are easily understood. However, in complex projects like office buildings, commercial centers, healthcare projects, etc. circulation is a critical aspect of architecture design. In these types of projects inappropriate circulation design can create problems for occupants and users of the building, and a decent circulation design of the project can improve various aspects of daily life for occupants as well as the building owners' economic goals.

3.1.1- Congestion

Congestion is an occupant-level value that has recently drawn the attention of architects, interior designers and space planers. In most building and urban space layouts, congestion has been considered as a negative quality. Congestion can cause discomfort and frustration, or even can be harmful in some situations. For instance, in urban areas with high rates of population and employment growth, traffic congestion can cause fear and unsafe sensations, which will affect lifestyles of individuals and threaten a city's economic success (Abdelgawad, et al. 2011). In healthcare settings, providing proper health services can be strongly affected by spatial layouts and patient flow inside the building. It is possible to omit or reduce the cost of alleviating strategies by predicting the flow patterns of patients and clients inside the hospital and modifying the circulation layout of the building, if necessary, during the design process of the project (Johnson and Happ 1977).

3.1.1.1- Productive Congestion and *Serendipity*

Although congestion is considered to be a negative quality for most architecture spaces, in some buildings, like *knowledge work* environments, some level of congestion is beneficial for increasing the productivity of individuals.

"Knowledge work" as Peter Drucker describes it is the work that happens mainly by mental activity instead of physical effort (Drucker 1959). This type of activity consists of planning, investigating, interpreting, developing, and generating products and facilities by utilizing information and concepts as the raw material, so it is considered as high-level mental activity (Suchman 2000). Since knowledge work is both cognitive and social, workers require time by themselves to develop their thoughts and process their thinking to aid their creativity (Claxton 2000). However, to make raw ideas and concepts useful to an organization they must be shared with others for further advancement. Therefore, conversations and interactions that convey one person's thoughts to others are extremely important parts of knowledge work. This transmission occurs through social networks where individuals come across one another throughout the normal working day in both official and unofficial settings (Brown and Duguid 2000).

Nowadays, the quality that provides the possibility for conversation and interaction in a knowledge work workplace layout is often called "serendipity"[1], which refers to creativity these days (Lindsay 2013).

In an office, while strongly grouped teams do well at solving problems in their knowledge work, they lack the connections to identify complementary thoughts from other groups or other parts of the company. Hence, unintended face-to-face conversations among people with various sets of skills and expertise might generate new ideas and solutions or at the least, increase solidarity between staff in the workplace.

Large companies have begun to realize this value and have begun looking for new ways to encourage interaction among employees from different departments that do not regularly work together. This interest, in combination with the increasing interest in algorithmic solutions and analysis in design is leading architecture firms that design workplaces to take a scientific approach to analyze office floor plans, compute the likelihood that employees will meet inadvertently, and identify the spots that these congestions occur. These scientific analyses provide the possibility for architects to increase the serendipity value of their design and decide on the best places for deploying amenities inside the building to maximize casual conversations between employees. For instance, when NBBJ architects designed the new Google headquarters building in ??, they did so in a way that facilitates the staff's collaboration and bumping into each other. They analyzed how fast people can flow inside the building by measuring the diameter of the space from various angles. They also made the floor plan narrower than in regular office buildings to reduce the distance between various teams (Silverman 2013).

3.1.1.2- Wayfinding in Existing Buildings

As mentioned before, one of the major issues that inappropriate circulation and wayfinding design will cause is difficult wayfinding. People avoid putting themselves in a setting in which they feel they will get lost. This is even worse for people with physical impairments. The sensation of being

[1] The British noble Horace Walpole invented the term in a 1754 letter in which he used the term referring to inadvertent finding.

lost in an architectural space creates a psychological barrier that stops people from going to buildings with complex circulation. Since most public buildings like shopping malls, commercial centers, hospitals, etc. are complicated buildings, the barrier that difficult wayfinding creates around them can be extremely harmful to the main functionality of these buildings and can threaten the economic goals of their owners.

Although most architectural firms realize the importance of appropriate circulation and wayfinding and try to improve these aspects of their future projects by doing scientific analysis, many public buildings with complicated circulation already exist in cities around the world. Wayfinding can be extremely difficult in them, and people do get lost in these complex public settings regularly. Modifying these buildings' layouts is either extremely expensive, or not practical in most cases. Addressing the problem by adding graphic displays and signs to interior components of the buildings is considered a poor work-around, and often conflicts with the interior architecture of the building and may still prove hard to interpret. These signage solutions are also very little help to people with sensory disabilities. Therefore, providing personal mobile wayfinding assistance for users of these settings by utilizing artificial intelligence might be the answer to difficult wayfinding issues in existing complex public buildings.

3.1.1.3- Circulation and Emergency Evacuation

There is no denying that more sensitive design of buildings is crucial for reducing the casualties during disasters. Although many buildings never actually experience a serious disaster, considerations for emergency evacuation is extremely important part of the building design process, especially circulation design. Although much legislation and many regulations that direct egress design are provided in building codes, evaluating the actual performance of the building during egress is extremely difficult. Apart from that, a great portion of disaster regulations are associated with providing a safe escape route in the case of hazard while in many conditions that is not sufficient to guarantee safe evacuation of the building. For instance, in large occupancy buildings like office blocks, shopping malls, hospitals, department stores, etc., where many people evacuate the building during disaster, some individuals may obstruct the evacuation of others due to high level of congestion. The effect of traffic and congestion in the escape route during

emergency egress can be more dangerous than the disaster itself when large numbers of people are involved (Canter 1980).

Time is the most important parameter for pedestrians when the decision to leave the building has been made due to disaster. All their activities from this point is toward one goal: escape. Experiential data shows that main reason of casualties during an emergency egress is usually angst an impulsive behavior of the crowd under panic instead of disaster itself. On the other hand, there are three specific factors affecting the time required for an individual's egress (Canter 1980):

1- Individual's confusion in choosing the exit paths.

2- Length of the exit path.

3- Individuals traffic during evacuation and extensive queue of people in the bottlenecks.

Since all three factors are directly related to circulation design of the building, it is safe to say that, a toolkit that evaluates emergency evacuation of a floor plan during the design process and provides insights about congested areas and individuals' traffic would be extremely helpful in reducing the casualties experienced during a disaster.

3.2- Problem Space, Grids and Graphs

The main requirement of a pathfinding algorithm is a topological model that defines relations between spaces and discriminates walkable places from obstacles and walls. The problem-space model is the environment in which a search operates (Newell and Simon 1972). It consists of a set of states of the problem, and a set of operators that change the state. Generally, two types of topological models are being used in a pathfinding process: a graph or a grid. Both of these models contain nodes representing locations in the space and edges that defines connections between nodes. Two steps are required for generating a graph: First, a set of nodes must be defined that represent location in space or cells of space. Secondly, connections of the nodes must be defined (Figure 4).

A raster model is a grid that can be considered a geometrical and topological model at the same time (Figure 5). Every node of the grid has between four to eight neighbors. While nodes of the

grid represent the occupiable positions in the space, edges of the grid that connect nodes to their neighbors determine the connectivity of nodes within the model and represent possible paths for going from one node to another.

The main difference between a grid and a graph is the number of connections between nodes in each of them. While each node of a graph can have various number of connections, grid nodes are only connected to their closest neighbors.

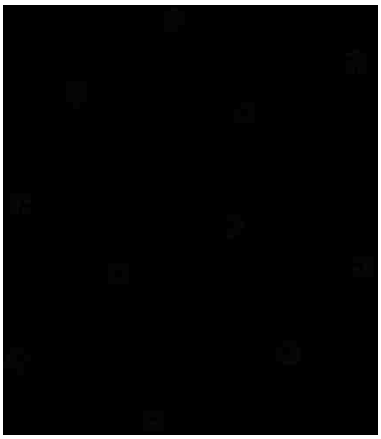

Figure 4: Graph.

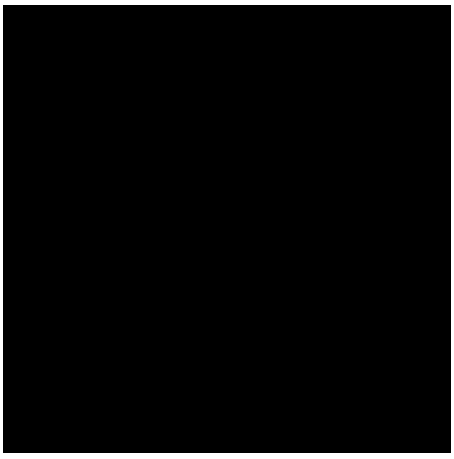

Figure 5: Grid.

3.2.1- Static and Dynamic Model

A model in which a path is computed might be dynamic or static. A static model does not change over time, whereas a dynamic model might alter over time. For computing paths in changing environments, dynamic models are required. One example of pathfinding in changing environments is navigation of discovery robots that search the surface of other planets like the Moon or the Mars, where the landscape is constantly changing. Another example of application of dynamic models is navigation during emergency evacuation, where parts of floor plan might be inaccessible due to fire or debris. Due to the lack of reliable information, and unanticipated changes happening on the landscape of other planets, discovery robots use dynamic models that are constantly being updated based on the robot's received data from the environment for pathfinding purposes.

3.2.2- Single Agent or Multi Agent

Pathfinding can be done for a single or for multi agents (i.e. simulated agents, robots, or people). Regularly, a static model is required for pathfinding for single agent. However, finding a path for multiple agents requires a dynamic model. Each agent's volume has to be included in the pathfinding calculations since agents should not collide with each other. Apart from that, pathfinding methods used for finding routes for an agent are not practical and efficient enough to do for hundreds of actors at the same time. Multi-agent pathfinding requires extreme computational power, so a real-time solution might not exist. Therefore, wayfinding for multiple agents needs a different method, like dynamic agent-based crowd simulation.

3.3- Pathfinding and Artificial Intelligence

Search is a universal solution finding mechanism in artificial intelligence (AI). One of the characteristics that discriminates AI search algorithms from other graph-searching algorithms is the size of the involved graph. For instance, the graph representing the entire chess game is approximated to have more than 10^{40} nodes, so finding a solution in this graph with traditional search method, which checks all the possible solutions would take millions of years. Consequently, the problem-space graphs of AI problems are never represented by listing each state, but rather are represented by identifying an initial state and a set of operators to generate new states from existing ones.

The sequence of steps required for finding the solution is not known in AI problems theoretically, and they must be determined by a trial-and-error inquiry of alternatives. AI search algorithms are used to address three generic classes of problems frequently: single-agent pathfinding problems, game playing, and constraint satisfaction problems (Korf, Artificial intelligence search algorithms 2010). This section provides fundamentals of search algorithms and explains some of the basic algorithms in more detail.

3.3.1- Brute-Force Search

Brute-force searches are the most generic search algorithms, because they do not require any specific domain knowledge. A search environment state description, a set of legal operators, an initial state, and a description of the desired solution are all that a brute-force search requires. The most famous brute-force search methods are *breadth-first*, *uniform-cost*, *depth-first*, *depth-first iterative-deepening*, *bidirectional*, and *frontier* search.

It is worth mentioning here that, in the upcoming text, to generate a node means to create the data structure corresponding to that node, while to expand a node means to generate all the children of that node.

3.3.1.1- Breadth-First Search

Breadth-First search (BFS) is an algorithm for searching decision tree or graph data structures invented by Konrad Zuse in 1945. BFS expands nodes in order of their depth from the root and generates one level of the tree at a time until it finds a solution (Figure 6). The easiest way to implement it is by maintaining a *first-in-first out queue* of nodes, that contains just the root at the beginning, and always removes the node at the head of the queue and expands by adding the roots' children to the tail of the queue.

Since it always generates a node in the tree after all deeper nodes have been generated, the result of the breath-first search is always the shortest path to a goal. The time consumed by the algorithm is proportional to the number of nodes generated, which is a function of the branching factor b and solution depth d, since each node can be generated in constant time. In other words, to find the nodes that are at the distance d from the root or start node (measured in number of edge traversals), BFS takes $O(b^{d+1})$ time and memory.

The huge memory required for BFS is the main disadvantage of the algorithm. The amount of consumed memory is proportional to the number of nodes stored, and the algorithm needs to store each level of the tree in order to generate the next level. As a result, BFS will exhaust the memory available quickly, and does not have many practical applications (Korf, Artificial intelligence search algorithms 2010).

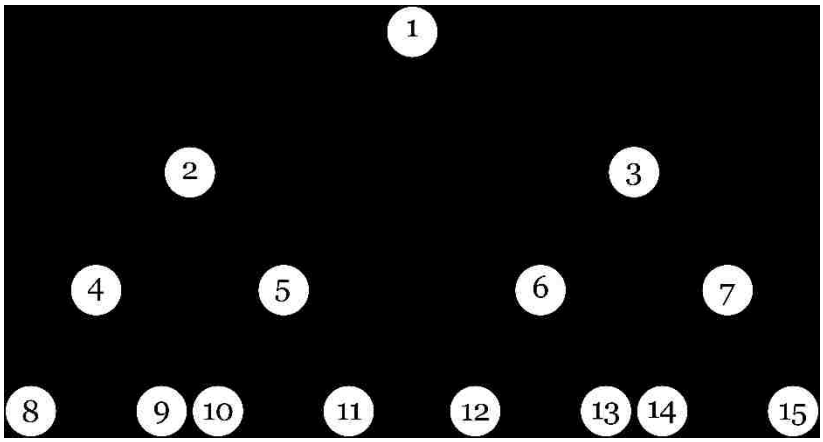

Figure 6: Sequence of generated nodes by breadth-first search algorithm.

3.3.1.2- Uniform-Cost Search

In a graph, if all the edges have same costs, then breadth-first search generalizes to uniform-cost search. Uniform-cost search expands nodes based on their cost from the root or initial state as a replacement for their depth from it. At each step, the node with lowest cost *g(n)* will be expanded, where *g(n)* is the sum of the edge costs from the root to node *n*. The best data structure for keeping the nodes is a *priority queue*. This algorithm is analogous to Dijkstra's shortest path algorithm. The main difference of the two algorithms is that uniform-cost search stops when a goal node is selected for expansion, while Dijkstra only stops the search when every node in a finite graph has been expanded.

Every time a node is chosen to be expanded by uniform-cost search, a lowest-cost path to that node has been found. Time complexity of uniform-cost search is $O\ (b^{c/e})$, where c is the cost of an optimal solution, and *e* is the edge with minimum cost. The main disadvantage of uniform-cost search like breadth-first search is memory limitation. Since the algorithm stores each level of tree in order to generate the next level, the memory consumption is proportional to the number of stored nodes.

3.3.1.3- Depth-First Search

Similar to BFS, depth-first search (DFS) is an algorithm for traversing tree or graph data structures. DFS overcomes the space limitation of BFS by always generating next a child of the deepest unexpanded node (Figure 7). A list of unexpanded nodes is required for DFS, like BFS, however, in contrast to BFS that manages the list as a first-in first-out queue, DFS treats the list as a last-in first-out stack.

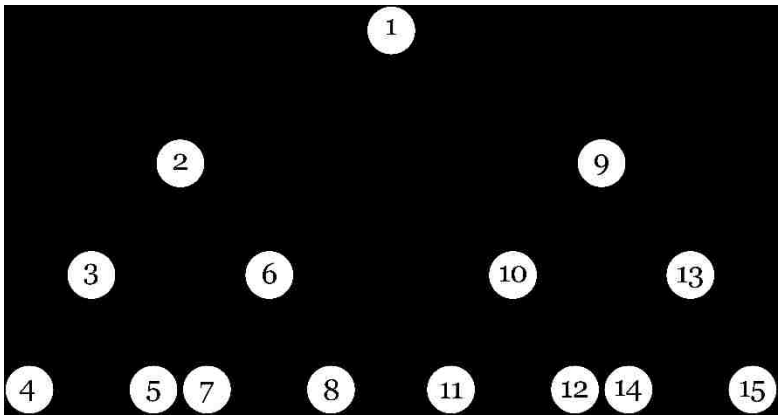

Figure 7: Sequence of generated nodes by depth-first search.

The main advantage of depth-first search is that its memory requirement has linear relationship to the maximum of search depth, as opposed to exponential relationship of breadth-first search. That is because DFS only stores a stack of nodes on the path from the root to the current node. The time complexity of a depth-first search to depth d is $O(b^d)$. Therefore, DFS is time-limited rather than memory-limited.

The main drawback of DFS is that it may not terminate on an infinite tree, but simply go down the left-most path forever. This is due to the usage of last-in-first-out stack, where the left most branch can be extended infinitely by generating an infinite number of duplicate nodes. The common solution for this problem is to impose a cutoff depth on the search. While the perfect cutoff is the depth of the solution *d*, this value is unknown in most of the advanced problems. Hence, if the selected cutoff depth is less than *d*, search will be unsuccessful, and no solution will be found, whereas if the cutoff depth is greater than *d*, execution time will be increased dramatically, and the first solution found may not be an optimal one (Korf, Artificial intelligence search algorithms 2010).

3.3.2- Heuristic Search

Solving more complex problems requires domain-specific knowledge to increase search efficiency. There are two definitions for heuristic search in artificial intelligence. First, in a general

sense, the term heuristic is used for any effective advice that is not guaranteed to work in every case. Second, a more specialized technical meaning, which refers to the specific case of a heuristic evaluation function.

3.3.3- Heuristic Evaluation Functions

A heuristic evaluation function approximates the cost of an optimum route between a pair of states in a single-agent wayfinding problem. For a static destination state, a heuristic evaluation *h(n)* is a function of a node *n* that guesses the distance between node *n* and the destination state. For instance, in order to estimate the distance of a pair of locations on a highway, *Euclidean* or airline distance is used. Or *Manhattan* distance is the heuristic function for solving sliding-tile puzzles, which can be computed by counting the number of moves along the grid that are required to go from one tile to another and adding them together.

The heuristic evaluation function must have two key characteristics. It must approximate the actual cost of each state change, and it has to be efficiently computable in constant time. For example, calculating Manhattan distance between two points is “constant time” similar to Euclidean distance, but calculating Manhattan distance between two states of a tiling puzzle is proportional to the number of tiles.

Most heuristic functions solve a simplified version of the problem, then utilize the cost of an optimal solution of the simplified problem as a heuristic evaluation function for the original problem. For example, a simplified version of the highway problem removes the constraint of driving on roads and navigates instead from one location to another by helicopter; the cost of an optimal solution of this simplified version is the Euclidean distance. In the sliding-tile puzzles example, removing the constraint that a tile can only slide into the blank position will simplify the problem. The Manhattan distance is the optimal number of moves required to solve this simplified version of the problem.

It is safe to say that such heuristics supply the lower bounds on the costs of optimum solutions to the original problem, because they result from simplification of the beginning problem. For instance, in the highway problem, the shortest possible path between two locations is the length of a straight line between them, which is the Euclidean distance between them. Similarly, the

Manhattan distance is a lower bound on the actual number of necessary steps required to solve an instance of a sliding-tile puzzle, because all tiles must move at least as many times as its Manhattan distance to its goal position.

3.3.4- Fundamental Pathfinding Algorithms

Many algorithms utilize heuristic evaluations, like a *pure heuristic search* and the A* algorithm, which are the fundamental to most of the sophisticated heuristic search algorithms. There are also other basic search algorithms that use heuristic evaluation, like *depth-first branch-and-bound*, *the heuristic path algorithm*, *recursive best-first search*. This section provides more detailed explanation of the pure heuristic search algorithm and the A* algorithm.

3.3.4.1- Pure Heuristic Search

The pure heuristic search algorithm is the simplest heuristic algorithm; it expands nodes based on their heuristic value *h(n)* (Doran and Michie 1966). It maintains a closed list, which keeps nodes that have been expanded, and an open list of those nodes that have been generated but not expanded yet. At the beginning, the algorithm keeps the start node in the open list. At each step, the algorithm expands the node with lowest *h(n)* value in the open list, and generates all of its children, and inserts it into the closed list. After applying the heuristic function to the generated children, they will be placed inside the open list based on their heuristic value. This process will be continued until the destination node is chosen for expansion.

The algorithm generates multiple paths to the same node in a graph with cycles and the first path found might not be the optimal one. The algorithm will discard the suboptimal path when it finds a shorter path. The main disadvantage of a pure heuristic search is that it is not guaranteed to find the optimal solution, because it does not take the cost of the path all the way to node *n* into the calculations.

3.3.4.2- A* Algorithm

The A* algorithm is the basis of many modern pathfinding algorithms (the family of "*" algorithms) and the most well-knowns. It was developed by Hart, Nilsson, and Raphael in 1968 (Hart, Nilsson and Raphael 1968). They combined features of uniform-cost search and pure heuristic search to find the optimal solution more efficiently. A* associates a cost function with a node. The cost function of node n is $f(n) = g(n) + h(n)$, where $g(n)$ is the cost of the path from the start node to node n, and $h(n)$ is the heuristic approximation of the cost of traveling from node n to the goal node. At each step the node with lowest f value is selected to be expanded, and the previous node will be considered as parent of the current node. If the f values of several nodes are similar, the node with lowest h value will be chosen for expansion. This process continues until the destination node is chosen for expansion. Eventually, path generation occurs by recursively following the parent pointers from destination node to start node.

A* finds an optimal path to the destination without overestimating the actual cost ($h(n)$ must be admissible). For instance, the Euclidean distance never overvalues the real highway distance, and the Manhattan distance never overestimates the actual moves in the sliding-tile puzzles. By using these evaluation functions, A* can find the optimal solution to those problems (Hart, Nilsson and Raphael 1968).

The main drawback of A*, like other best-first search algorithms, is the amount of memory required for the algorithm to operate, since the open and closed lists are stored in memory. A* is extremely space-limited in practice and will exhaust the available memory quickly. In addition, the process of managing the open and closed lists is computationally complicated and time consuming, which reduces the speed of A*.

Accuracy of the heuristic function establishes the time complexity of a heuristic search algorithm. Therefore, A* operates on linear time if the heuristic evaluation function is an accurate estimator and expands only the nodes on an optimal solution path. Contrarily, A* operates similar to a brute-force uniform-cost search, and the time complexity becomes exponential if the heuristic function returns zero for every node. Generally, it is possible to reduce the effective depth of search by defining a good heuristic function (Korf, Reid and Edelkampz, Time complexity of iterative-deepening-A∗ 2001).

3.3.4.3- *AP-Theta** Algorithm

As mentioned before, the A* algorithm quickly finds paths that are constrained to the grid edges. However, these paths are not the optimum paths, because possible headings of the paths are artificially limited to multiples of 45 degrees (Figure 8) (Yap 2002). In addition, limited headings make generated routes unrealistic in appearance, which is not appropriate for simulating human's pathfinding behavior (Figure 9).

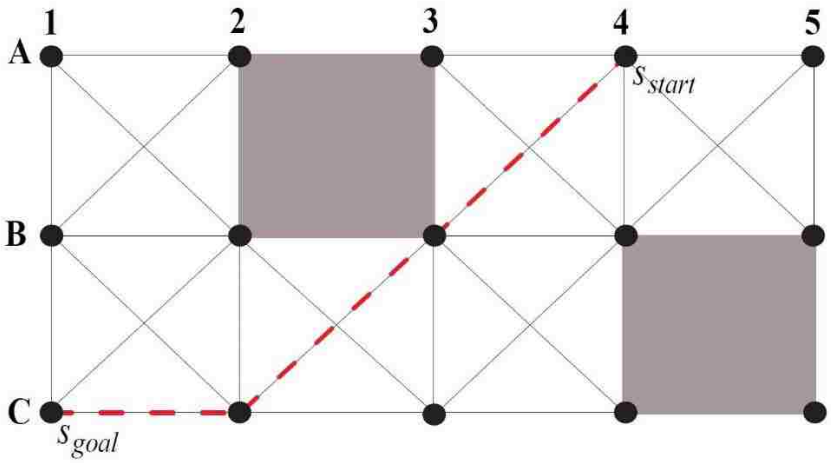

Figure 8: Limited headings of the generated path by A algorithm.*

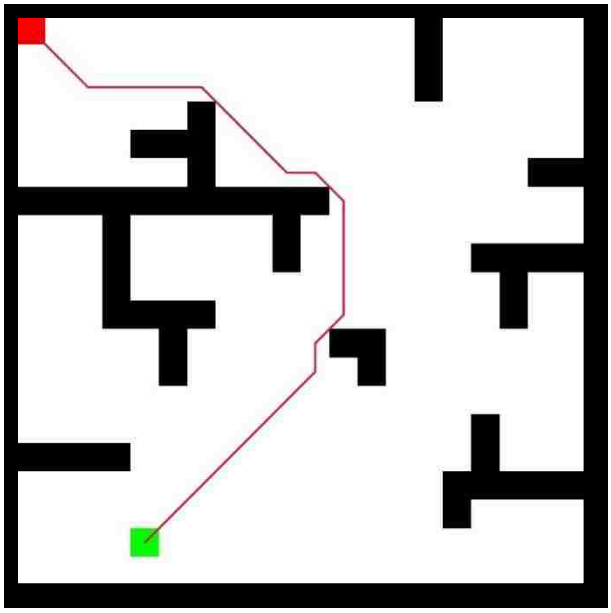

Figure 9: Sub-optimal and unrealistic path generated by A algorithm.*

In order to overcome A* algorithm's drawbacks, any-angle path planning was introduced (Nash, et al. 2007). AP-Theta*, or Angle-Propagation Theta* is a variant of the A* algorithm that generates routes without limiting the headings by propagating information along grid edges (to achieve a short runtime).

The AP-Theta* algorithm is built upon the A* algorithm; the main difference between them is that in AP-Theta* the parent of a node can be any vertex, in contrast to A* where the parent must be one of the eight neighbor nodes of the vertex. AP-Theta* calculates two extra values beside cost value *g(n)* and heuristic value *h(n)* for each node of the grid: a lower angle limit *lb(n)* and an upper angle limit *ub(n)*. These two extra values create the angle range property of each node [*lb(n), ub(n)*]. Angle range property will be used to determine whether or not two nodes can be connected by *line-of-sight*.

A simple analogy for a line-of-sight test between nodes is putting a lamp on top of a node. Light cannot pass through blocked cells, so all the nodes that are not located in the shadow of blocked cells have line-of-sight to the node under the lamp. It is possible to identify the nodes with line-of-sight to the lamp-node using the angle range property of the lamp-node. Angle range is calculable using two light rays emitted from the lamp. Figure 10 illustrates line-of-sight test of node *S*. All the nodes in the yellow area defined by the angle range property $[\theta_1, \theta_2]$ have line-of-sight to vertex *S*. The angle range of each node will be calculated and propagated along grid edges

at the time of expansion of the node by AP-Theta*. This process results in a constant runtime per node expansion, because both line-of-sight test and angle range propagation occur in constant time.

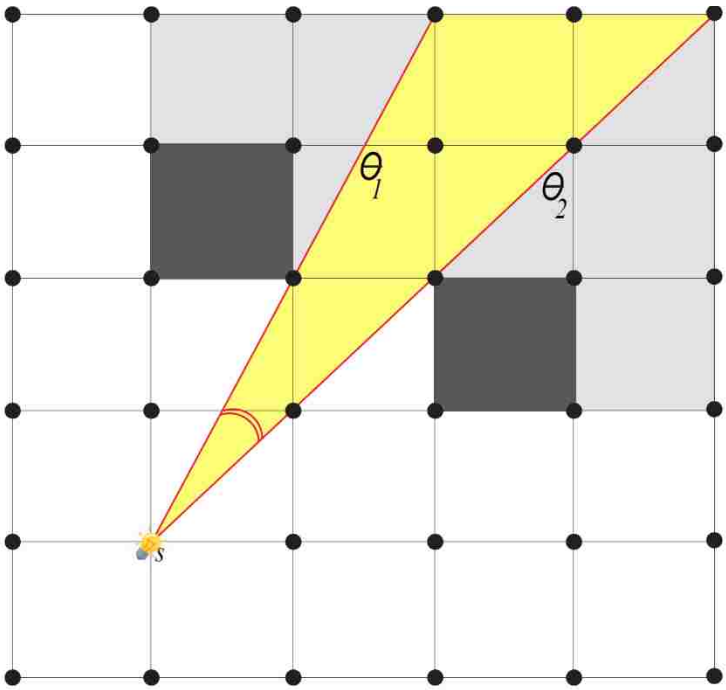

Figure 10: Line-of-sight test of node S.

As mentioned above, AP-Theta* calculates one additional property, named "angle range," which consist of a lower angle bound and an upper angle bound for each node. This property is related to heading rays from the parent of node *S* to the node itself measured in degrees. A visible neighbor of node *S* has line-of-sight to node *S* if (but not necessarily only if) the angle between the ray from the parent of node *S* to the visible neighbor of node S, and the ray from the parent of node *S* to node *S* is contained in the angle range of node *S*.

In order to define the angle range property of node S, it is required to define Θ *(S, P, X)*, which gives the algorithm its name. If node *P* is the parent of node *S* and node *X* is its successor, Θ *(S, P, X)* indicates ∠ *(S, P, X)* which is limited to between 180° and -180°. This angle is positive if the ray from *P* to *X* is counterclockwise from the ray connecting *P* to *S*, and it is negative if the ray from *P* to *X* is clockwise from the ray connecting *P* to *S*. AP-Theta* constrains the angle range of each node before expanding it, based on the blocked cells adjacent to the node and successors of the

node. Then the algorithm checks the following condition: for successor *X* of node *S*, if *lb(S)* ≤ Θ *(S, P, S ´)* ≤ *ub(S)* then *X* has line-of-sight to parent node of *S*.

For example, in Figure 11 where A4 is parent node of C3, the lower bound of node C3 is $\measuredangle(B3, A4, C3)$ equal to -18° and upper bound of C3 is $\measuredangle(C4, A4, C3)$ equal to 27°. Therefore, all the neighbors of C3 that pass the line-of-sight condition are guaranteed to have line-of-sight to parent of C3 which is A4.

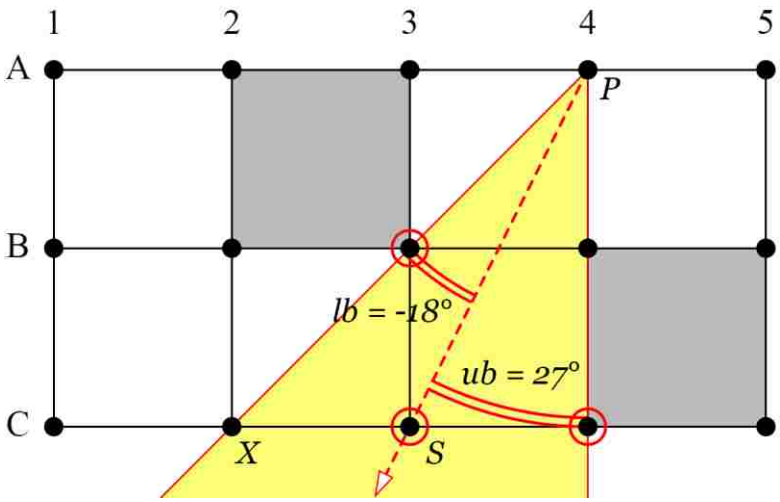

Figure 11: Maintaining angle range of node S.

Figure 12 illustrates the paths generated by both Theta* and A* algorithms from node A4 to node C1. Since in A* algorithm parents and successors have to be neighbors, the sequence of parent and successor nodes in the generated path, which is shown by red line from goal node to start node is: C1→C2→B3→A4. On the other hand, Theta* maintains the angle range of each node before expansion. In the generated path by Theta*, which is illustrated by the blue line, node B3, which is the parent of node C2 has line-of-sight to node C1, so it will be assigned as the parent node of C1. Consequently, the sequence of parent and successor nodes in the generated path by Theta* is: C1→B3→A4. Comparing routes generated in this simple example indicates that, not only is the route generated by Theta* shorter than the one generated by A*, but also it is more realistic looking.

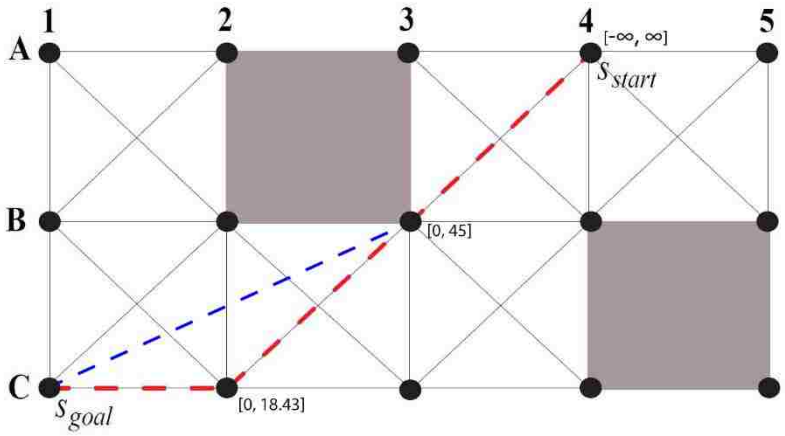

Figure 12: Generated path by Theta (blue), and A* (red).*

3.4- Indoor Localization

Tracking a user's location with high precision in urban and indoor environments has a variety of applications in the healthcare, logistics, and entertainment industries since GPS service can be either blocked or adversely affected by multipath propagation in such environments.

A variety of methods has been proposed for providing functional pedestrian tracking services with less than a few meters error by utilizing various technologies as mentioned previously. This chapter provides more detailed explanations about the method used in this research to track a user's location and movements dynamically.

3.4.1- Dead Reckoning

The term "dead reckoning" *(DR)* describes a position solution that is gained by calculating movements from a known starting point based on the motion of the user. It was originally a simple

navigation method that served European mariners for centuries. Starting at a known or assumed position, they tracked three parameters utilizing simple but reliable instruments:

- The ship's compass heading.
- The speed of the ship.
- The time spent on each heading and at each speed.

The navigators were able to calculate the route and distance the ship had moved and mark a sea chart, if he had one. This method was used by Columbus and most other mariners of the Age of Exploration (Reckoning 2018).

A simple device for use in pedestrian dead reckoning *(PDR)* is the pedometer, which usually is an electronic device that counts the number of steps that the user takes by detecting the motion of person's hand or hips. It can be used to calculate distance but not direction.

The ubiquity of accurate and cheap sensors, including the accelerometers, gyroscopes, and magnetometers found in smartphones these days, has reinvigorated interest in DR for pedestrian navigation in environments where other navigation systems are unavailable.

3.4.1.1- Accelerometer

An accelerometer is an instrument capable of measuring the proper or physical acceleration experienced by an object (Tinder 2007). Proper acceleration is the velocity change rate of a body in its instantaneous rest frame (Rindler 2013), and it is different from coordinate acceleration, which is acceleration in a static coordinate system. A static accelerometer on the Earth's surface will show approximately 9.81 m/s^2 upward since any point on the earth surface is accelerating upwards relative to the local inertial frame (due to gravity). Theoretically, an accelerometer is a damped mass on one or more springs. When the system experiences an acceleration, the mass pushes on a calibrated spring to the extent that the spring can accelerate the mass at the same rate as the initial acceleration (Figure 13). The dislocation of the mass is then measured to calculate the acceleration. Modern accelerometers are often small *micro electro-mechanical systems* (MEMS) that consist of little more than a cantilever beam with a proof mass (*seismic mass*).

Sensors incorporating three such devices oriented at right-angles to each other are able to measure acceleration in all three spatial directions at the same time.

Figure 13: Schematic structure of accelerometer.

3.4.1.2- Magnetometer

Magnetometer is a device capable of measuring the strength (in unite of Gauss) and direction of the local magnetic field. The simplest type of magnetometer is the traditional compass (Figure 14) consisting of a magnetized needle, which changes orientation based on the earth's magnetic field and shows the direction of the field.

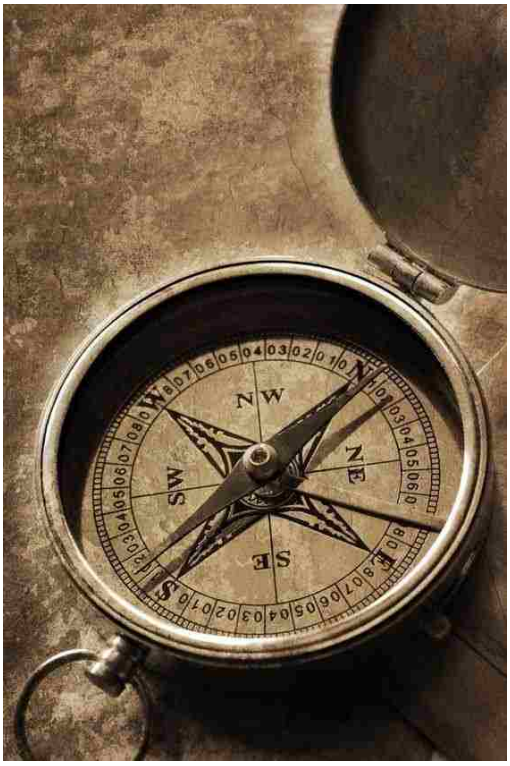

Figure 14: Traditional compass (Mizaralkora 2017).

Since electronic magnetometers have become extremely cheap, most modern smartphones contain a *Tri-axis* electronic magnetometer, which measures the strength of the magnetic field and acts as a compass (Magnetometer 2018). The sensor uses the Earth's magnetic field as a reference to discover the phone's orientation along x-, y-, and z-axes. Fortunately, Tri-axis magnetometers are not affected by orientation or elevation. "Otherwise users would have to hold the phone precisely parallel to the ground or in some other position that may not correspond to how they normally use it," according to Mark Laich, vice president of worldwide sales at Memsic, a maker of electronic compasses based in Andover, Massachusetts (Jones 2010).

3.4.1.3- Gyroscope

A gyroscope measures and maintains orientation and angular velocity. It consists of a wheel or disc that spins while the rotation axis is free to assume any orientation (Figure 15). During

rotation, the rotation axis is not affected by sloping or rotation of the mounting because of the conservation of angular momentum (Worthington 1906). Due to their precision, gyroscopes have a variety of applications in science and practice. For instance, they are included in the inertial navigation system of Hubble telescope. Also, it is possible to build *gyrocompasses*, which can be added to, or replace magnetic compasses in transportation vehicles as part of an inertial guidance system. Gyroscopes have captured the attention of many designers who have integrated them into modern technologies like smartphones, game consoles, and virtual reality sets in the last decade, since they provide the possibility of calculating orientation and rotation (Gyroscope 2018).

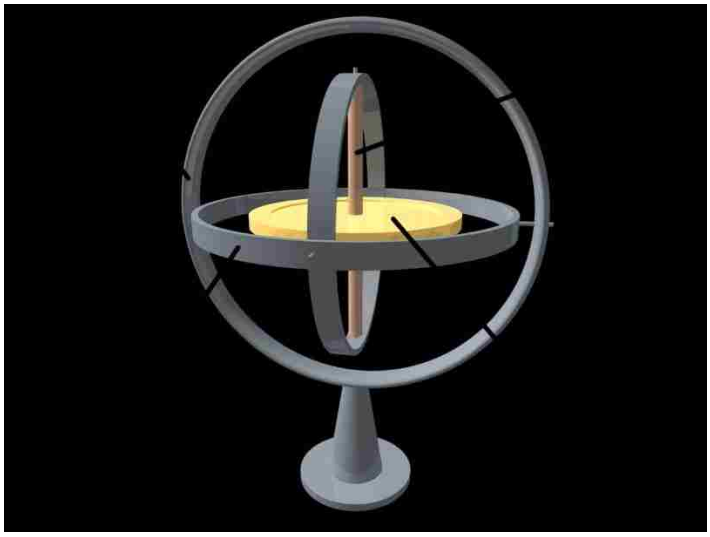

Figure 15: Gyroscope schematic structure (Gyroscope 2018).

It is possible to utilize a gyroscope as a heading indicator. This type of gyroscope is called a directional gyroscope. It has a horizontally set axis of rotation, pointing north instead of seeking north similar to magnetic compass. Their main drawback is that, when being used, it slowly drifts away from north, and it is required to use a magnetic compass to reorient them periodically (Feynman, Gottlieb and Leighton 2013).

4- Methodology

These days, computational and algorithmic tools are being used to improve various aspects of architecture design by simulating real world events before their occurrence using the digital world of computers. The results of these simulations not only demonstrate how the building will perform after construction, but also provide insights about the effect of various design changes on the performance of the building. The only other way for accessing such information is by empirical experiments, which are extremely costly, and their results might not repeat on future scenarios. For instance, architecture firms are using daylight simulation programs like Diva or Insight360 to evaluate the performance of their designs based on the amount of natural daylight it receives.

Circulation design, as one of the most important aspects of architecture design process, can benefit from these digital simulation tools since inappropriate and complex circulation design may affect functionality of the building on various levels and buildings with proper circulation improve the human experience inside the space. For instance, in a well-designed building, individuals would spend less time interpreting the direction information shown by graphical signs or given by staff inside a building, eliminating some of the efficiency problems of the building. In addition, suitable circulation design may affect the level of congestion, which is one of the most important occupant-level spatial qualities inside the space. While appropriate circulation design may cause serendipitous congestion in some building layouts, unsuitable circulation may cause discomfort sensation for occupants of other building layouts. Apart from that, individuals, especially the ones with sensory impairments tend not to attend buildings with complex circulation, since they get the fearsome sensation of being lost in those settings. Furthermore, occupant safety of a building is extremely affected by the wayfinding design of that building. Not only it is much more problematic for occupants to evacuate buildings with complicated circulation during emergency situations, but first responders have a harder time providing their services in those buildings. Moreover, analyzing and evaluating designed floorplan of a building based on the codes and standards related to circulation, accessibility, and safety provide important insights and information for designers and building owners who pursue Living Building Certification. However, digital tools that simulate human behavior and human experience inside the built environment are extremely rare and expensive. FlexSim for example, is a very sophisticated simulation tool that costs tens of thousands of dollars per year.

Beside all the benefits that digital tools might provide for circulation and wayfinding design during architecture design process, they also may improve utilization of the existing buildings with complex circulation by providing navigation assistance for the occupants similar to outdoor PNS. However, due to challenges in the way of developing digital indoor navigation assistance, like unavailability of GPS signals inside buildings, absence of appropriate maps, which designate the position of the obstacles, routes, walls, plus destination inside a building, and complexity of indoor environment and freedom of individuals' motion inside the building, indoor pedestrian navigation systems have not been developed adequately.

In order to improve circulation design process of a building and provide insights about human wayfinding behavior, which is one of the goals of this research, an open source toolkit for the Rhino/Grasshopper platform was developed based on heuristic search algorithm (AP-Theta*). In addition, an IOS smartphone application was developed utilizing the same algorithm to provide indoor navigation assistance for users of complex buildings.

4.1- Workflow

The strategy taken for developing the toolkit in this research required choosing a workflow for connecting the regular architectural design documents of a building to the analysis toolkit, since different firms use various methods and software programs in their design process. Apart from that, it was decided that toolkit must be as easy as possible to use by automating most of the process.

Although architecture firms have various workflows in design process, most of them use Revit, a BIM software developed by Autodesk for developing their design after concept development stage of the design. In addition, BIM models contain more information (semantic) related to the design than CAD drawings which usually only contain geometrical information of the design. Therefore, Revit was chosen as the starting point of the workflow, meaning that designed floor plans will be extracted from Revit model for analyzing and evaluating.

On the other hand, Rhino is a popular software for analyzing and evaluating various aspects of the architectural design, since geometrical manipulations are easy in Rhino environment. Apart

from that, the existence of Grasshopper, which is a visual programming environment in Rhino, provides the possibility for developing various analysis tools that are directly involved with the geometry and information of the design. Consequently, Rhino\Grasshopper platform was selected as the end-point of the workflow, meaning that wayfinding package will be developed as Grasshopper plugin and circulation analysis prototypes will be developed in Rhino\Grasshopper environment. On the other hand, iOS was chosen as the final platform for developing the indoor pedestrian navigation system.

4.1.1- Data Transfer

The fundamental requirement for developing the wayfinding package on a floorplan is identifying the walkable spaces of the floorplan. Fortunately, this information is part of almost every professionally made Revit model. *Room-Data* is a property of each space in each floor of the model in Revit, which highlights the boundaries of different spaces in Revit. Figure 16 presents a floorplan in Revit that contains Room-Data for all the walkable spaces of that specific floor with distinct colors. Therefore, the main requirement of data transformation process on the Revit side, is to transfer Room-Data along with the reset of the floorplan.

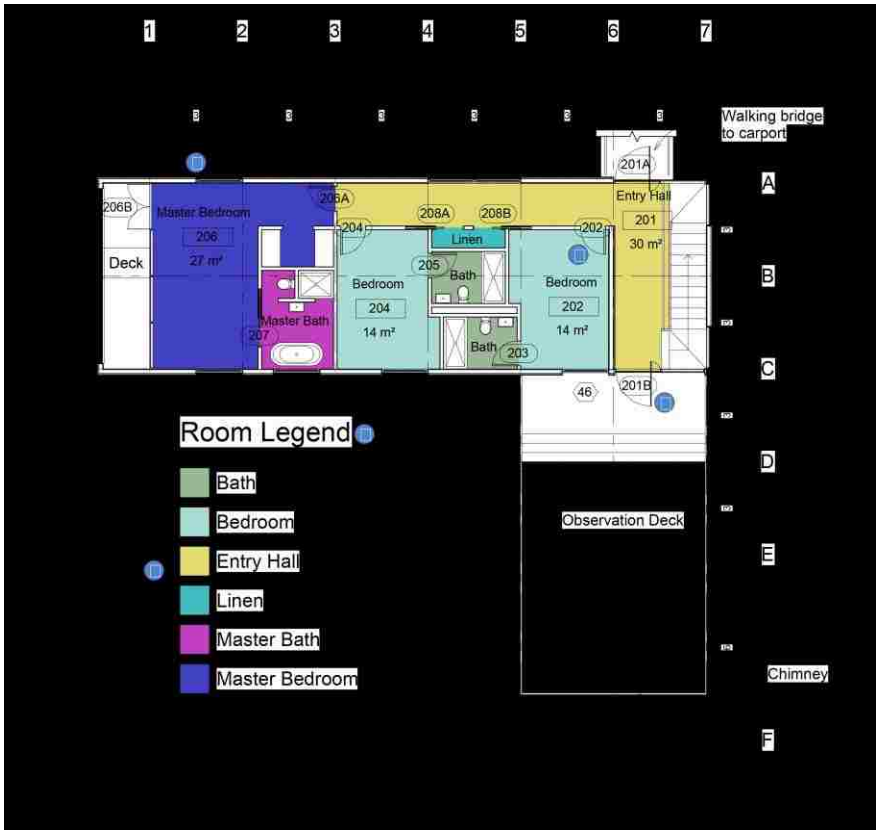

Figure 16: Room-Data in Revit floorplan.

It is possible to export data from Revit in many different formats. For example, FBX is the format that captures geometry, material, and organization of objects (layers) more completely than other formats. Accordingly, it is the best format for transferring 3D models between Revit and other modeling software, specifically other Autodesk products. However, FBX is not a decent format for transforming 2D information of a floorplan, since Room-Data is not included in it. In contrast to FBX, DWG and DXF are the formats that capture most of the 2D information, like room data, or material hatches in plan, but they operate poorly in transferring 3D information.

Since the wayfinding package requires a data-rich 2D floorplan, which contains information related to the position of the walls, furniture, and room-data, DWG was selected as the transfer format for this project. Figure 17 illustrates the floorplan of Figure 16 in Rhino environment after transferring from Revit as DWG format. In this transfer Revit encodes Room-Data as a hatched polygon object that, as Figure 17 shows, includes the full floor area, ignoring the location of furniture and other objects (cabinets, sink, toilet, etc.) that would actually reduce the walkable floor area.

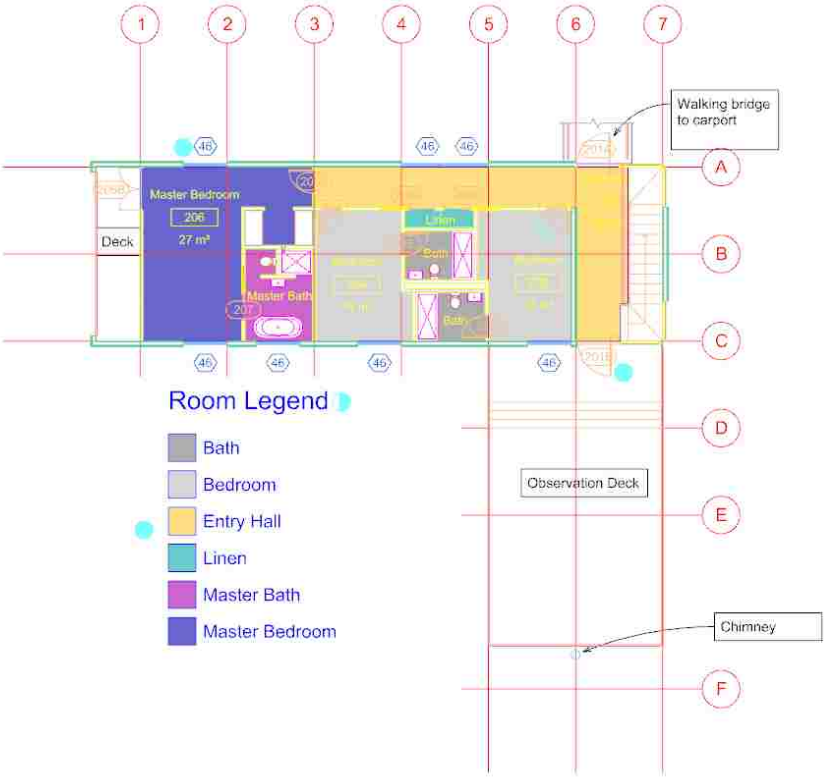

Figure 17: Room-Data in Rhino as Hatch object.

4.2- Extracting the Navigation Model

As mentioned before, most of the top heuristic pathfinding algorithms require a navigation grid to operate on, consisting of *nodes* that represent locations and *edges* that connect the location nodes inside the space together. The navigation grid, which is called navigation model in the literature must discriminate walkable space from obstacles in any landscape (outdoor, indoor, or other planet). Therefore, there are two types of nodes in the navigation model:

1- Location nodes, which are walkable.

2- Wall nodes, which are unwalkable.

Consequently, the first step in developing the floorplan pathfinding package is the generation of the navigation model on top of the floorplan in the Rhino environment. In addition, the model generation process must be as automatic as possible to be useful, since generating the model manually on top of large floorplans is extremely tedious and time consuming.

Considering all the required features of the navigation model, a Grasshopper script was designed to generate the model on top of any floorplan that contains Room-Data in Rhino as Hatch object. The script is composed of several 2D geometrical operations that will be demonstrated one by one.

Step 1: importing Room-Data into Grasshopper. The only manual operation of the navigation model generation process is transferring Hatch object to Grasshopper. Unfortunately, Hatch objects are not recognized by Grasshopper. Thus, they must be converted into another geometry type. As Figure 18 shows, Rhino changes a Hatch object to an identical *Brep* (boundary representation) to the initial Hatch if user explodes the Hatch. Since Brep is a data type recognized by Grasshopper, the first step of navigation model generation is to change hatches representing the Room-Data to Brep by exploding them and import the Breps into Grasshopper.

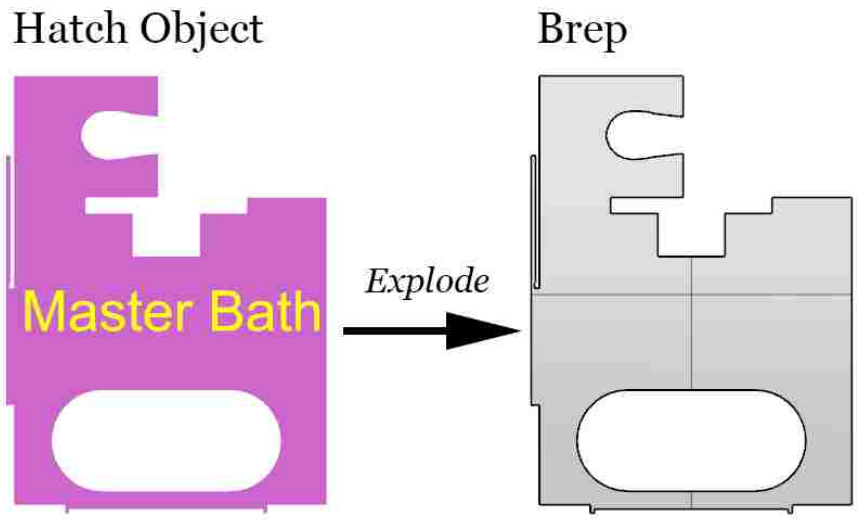

Figure 18: Changing hatch to Brep by explode operation in Rhino.

Step 2: Generating the context. Once the Breps were imported into Grasshopper, it is required to generate the context of the navigation model. Context model is a square surface that covers all the Breps. Simply put, it is the *bounding box* of all the Breps, which was scaled in the

direction of the shorter edge to become a square with edges length equal to longer edge length of bounding box (Figure 19).

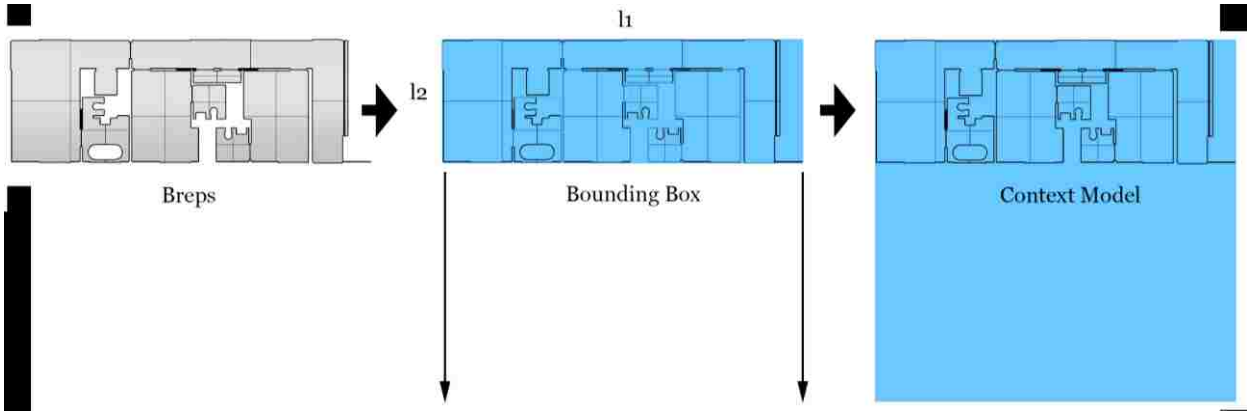

Figure 19: Process of generating Context of the navigation model.

Step 3: Raw-grid generation. The second geometrical operation in creating the navigation model is generating the navigation grid inside the context model. Grid size must be selected based on the complexity of the floorplan. For instance, complex floorplans with many furniture items in them require smaller grid size than free floorplans with no furniture in them. The recommended range for grid size is between 8" to 24", since human cannot pass bottlenecks narrower than 8' and grid cells bigger than 24" will make the navigation model excessively abstract (Figure 20). A two-dimensional matrix was utilized for storing the grid cells.

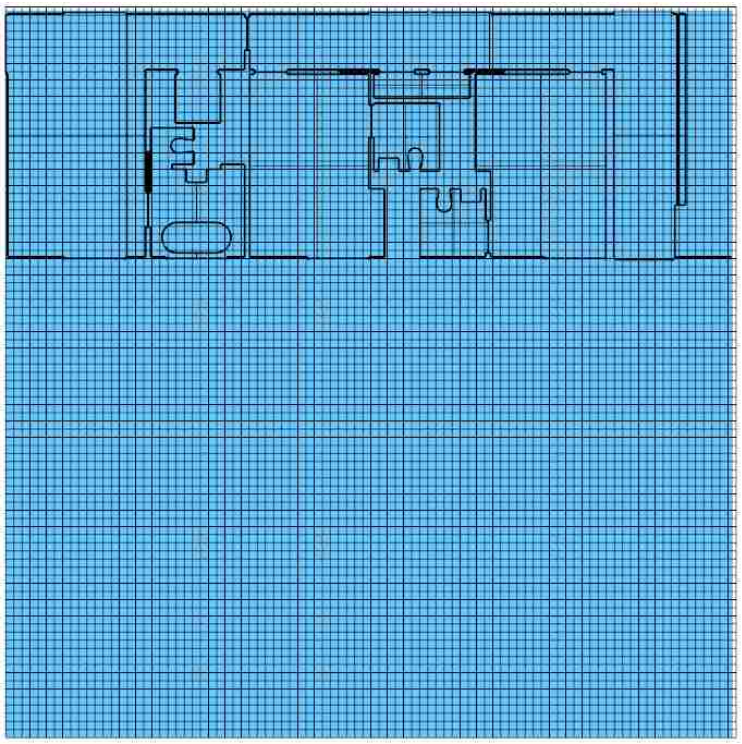

Figure 20: 8" raw-navigation grid inside the context model.

Step 4: Discriminating the search area. To discriminate grid cells that are walkable from the unwalkable ones and wall-cells, a test was designed for each of the cells. First, every edge of each cell was divided to 7 equal segments by 8 points. Simply put, each cell now contains 32 points equally distributed on its four edges. Next, the distance between each one of these 32 points and their closest points on each Brep was calculated and added together. This operation will provide one number for each cell. If the number is equal to zero, then all 32 points of the cell are guaranteed to be located on top of at least one Brep, which means that cell is walkable and part of the search area. If the number is bigger than zero, then at least one of the 32 points of the cell is outside the walkable area, so the cell is considered as unwalkable (a wall-cell). This method distinguishes not only walkable cells and wall-cells, but also creates a buffer about the size of one cell around walls and obstacles, which makes the navigation model more realistic, since it is unlikely that people will walk clinging to the walls or furniture inside a space. Eventually, results of the test for each cell was stored as zeros and ones (one for wall-cells and zero for walkable cells) in a two-dimensional matrix associated with the position of that cell in the matrix of the cells. Figure 21 illustrates the final navigation model generated in Grasshopper. Walkable cells are

illustrated in green while wall-cells are shown in orange and red. Orange cells are the offset cells around walls and furniture, which like red cells are not walkable.

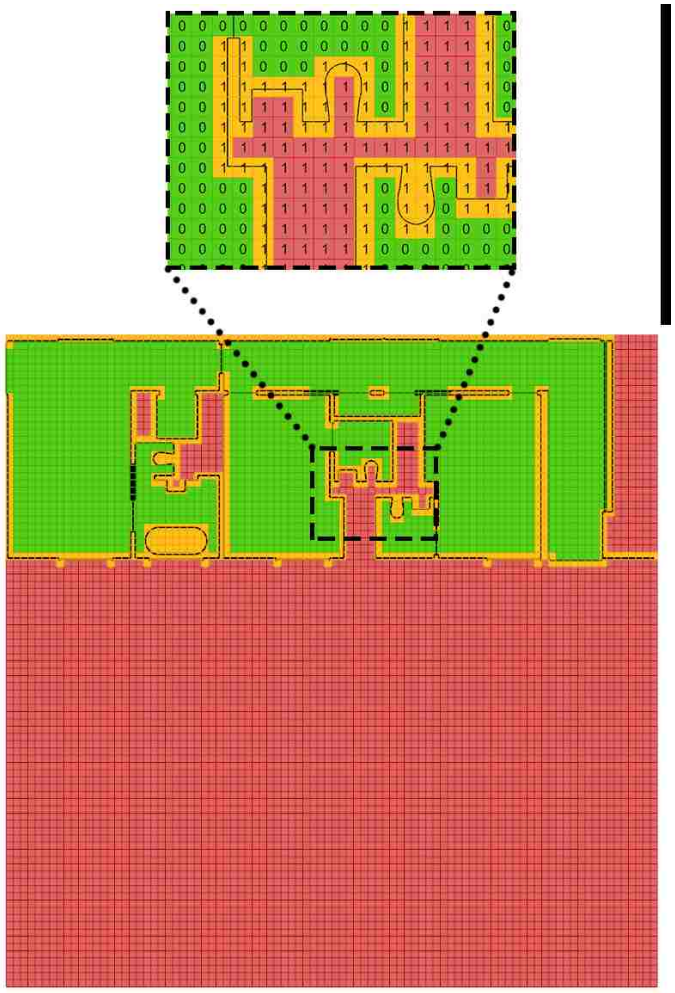

Figure 21: Final navigation model. Zeros illustrates walkable cells (Search area) and ones illustrates wall-cells.

4.3- Data Structure and Methods

In order to develop the pathfinding package, I decided to use C# programming language, which proved to be most effective, since Grasshopper has been developed with this language. Therefore,

Microsoft Visual Studio 2015 was used as the programming platform for developing the wayfinding package for Grasshopper.

4.3.1- Data Structure

Since C# is an *object-oriented programming* language, the wayfinding package uses an object-oriented format. Object-oriented programing or *OOP* is a programming model based on the concept of *objects*, which may contain data in the form of attributes, and code in the form of procedures or methods. In OOP, computer programs are designed based on instances of defined objects that interact with one another.

Since the AP-Theta* algorithm discriminates nodes from cells and uses them separately to calculate the angle range for each node. Two types of objects for storing the whole navigation model:

1- Spot object type.

2- Cell object type.

Although each of these data objects has its own attributes and methods, they are also interwoven, since one of the attributes of the Spot object is a list of adjacent Cell objects. Similarly, one of a Cell object's attributes is a list of Spot objects representing the four corner Spots of the Cell.

4.3.1.1- Spot Object Attributes

The Spot object was designed to store and represent nodes of the navigation model. Therefore, its attributes and methods were defined based on the requirement of AP-Theta* algorithm.

Spot objects have to be comparable, since AP-Theta* needs to compare each Spot instance to another one based on the heuristic value or *h(n)* of each. Thus, the "IComparable" interface, which allows object to be compared or sorted based on a value was implemented into the Spot data type.

All the Spot object's attributes are mentioned below:

id: An array with two empty slots, in which the algorithm stores the column and row indices of the Spot instance in the initial matrix. For instance, the *id* of the second Spot object in the third row is [1, 2] (Figure 22).

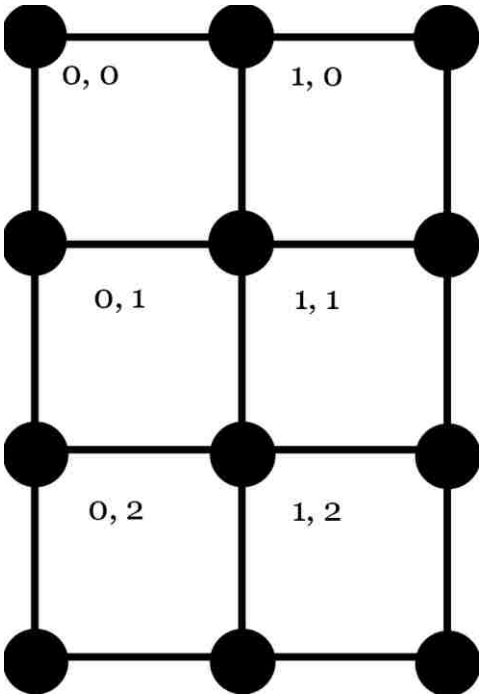

Figure 22: id attributes of Spots.

name: A string representing name of each spot of the navigation model. This attribute is empty by default.

pos: A Point3d object representing the (x,y,z) coordinates of the Spot object on the Cartesian system.

cellSize: A double? precision number representing the distance between each Spot instance, which is equal to the size of the cells in the navigation model.

lb: A precision number presenting the lower angle bound attribute of each Spot instance in degree. This attribute is required by the pathfinding algorithm to evaluate the line-of-sight range of the Spot.

ub: A precision number presenting the upper angle bound attribute of each Spot instance in degree. This attribute is required by the pathfinding algorithm to evaluate the line-of-sight range of the Spot.

los: A Boolean representing the Line-of-sight attribute of the spot relative to it's 'parent' spot. If a Spot has line-of-sight to its parent, this attribute is true; otherwise, it is false. At the beginning of the algorithm all the model's Spots have true as their *los* value.

closedSet: A Boolean that shows if the spot was expanded before or not.

f: A number representing the total cost function of the Spot.

g: A number representing the cost of going from the start Spot to this Spot.

h: A number representing the heuristic guess value of the Spot (the estimated cost of going from this Spot to the destination Spot).

parent: A property of Spot object for storing another Spot object as parent of each node.

neighbors: Spot ids of the eight (cardinal and diagonal) neighboring Spots of the current Spot (Figure 23).

Figure 23:Neighbors attribute of a Spot.

adjCells: A list of Cell data types where the algorithm stores the four neighbor Cells of the Spot (Figure 24).

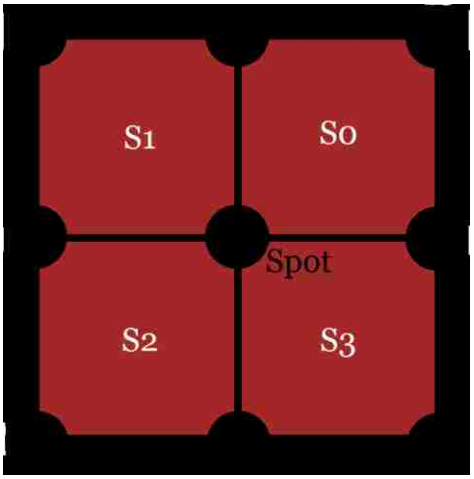

Figure 24: Adjacent Cells of a Spot.

4.3.1.2- Spot Object Methods

Construction Method: All objects require a construction method to be instantiable. A Spot object's construction method requires three inputs:

1- A two-slot array of numbers related to the location of the spot in the Cartesian system. The construction method stores this array in the "*pos*" attribute of the Spot object (Spots are always at Z=0).

2- A two-slot array of numbers related to the id of the Spot in the initial matrix of Spots. This array will be stored in *id* attribute of the Spot by construction method.

3- A precision number, which is related to the cell size of the navigation model. The construction method stores this value in the "*cellSize*" attribute of the Spot object.

getNeighbors Method: This method has been designed to find all eight neighbors of each Spot in the navigation model and store them in the "*neighbors*" attribute of that Spot.

CompareTo Method: This method gets one Spot object other than the current Spot object and compares the current Spot's *f* attribute with the *f* attribute of the other Spot. Eventually, it returns (0) if two Spots have the same *f* value, (1) if the current Spot has a bigger *f* attribute, and (-1) if the

current Spot has a smaller *f* value. It is possible to sort a list of Spots based on their *f* attributes using this method or to store them in a priority queue data structure.

4.3.1.3- Cell Object Attributes

The Cell object was designed to store and represent Cells of the navigation model. Therefore, its attributes and methods were defined based on the requirement of AP-Theta* algorithm.

All the Spot object's attributes are mentioned below:

wall: An integer that represents if the Cell is an obstacle or not. Obstacle Cells have (1) as their wall attribute, while walkable Cells' wall attribute is equal to (0).

cornerSpots: An array of Spots with four slots, in which the algorithm stores Spots located on the corner of the Cell (Figure 25).

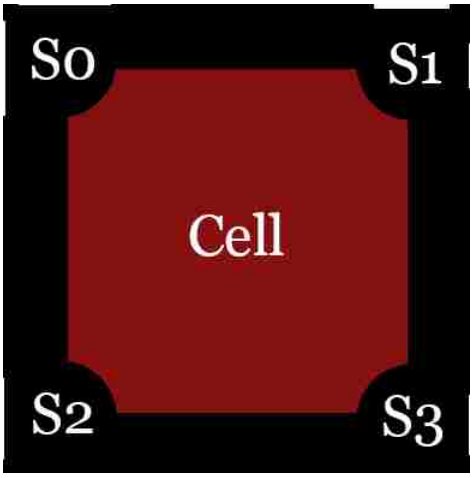

Figure 25: Spots located on the corners of a Cell object.

4.3.1.4- Cell Object Methods

Construction Method: The construction method of the Cell objects requires two inputs:

1- A precision number, which is related to cell size of the navigation model. This value will be used only for visualizing the Cell.

2- An integer that defines the wall attribute of the Cell. The construction method stores this value in the wall attribute of the Cell.

display Method: This method visualizes the Cell object in the Rhino viewport. The method extracts the four stored Spots from the "*cornerSpot"* attribute of the Cell and draws a shape in the Rhino viewport: an empty rectangle if the wall attribute is (0) or a colored rectangular surface if the wall attribute is (1).

4.3.2- Methods

This section introduces and explains methods that were developed specifically for this project. Other methods are similar to the ones explained in the original AP-Theta* (Nash, et al. 2007).

4.3.2.1- Theta

This method was designed to get three Spots and calculate the angle between them. The angle then will be used by the AP-Theta* algorithm to define the angle range of the Spots or evaluates if a Spot has line-of-sight to another Spot.

If *S1*, *S2*, and *S3* are three input points in a sequence, then the method needs to calculate four values before calculating the angle between three spots:

1- ***DP*:** Dot-product of the vector from *S2* to *S1* and the vector from *S2* to *S3*.

2- ***Z*:** Z value of the cross-product of the vector from *S2* to *S1* and the vector from *S2* to *S3*.

3- ***M1:*** Magnitude of the vector between *S2* and *S1*.

4- ***M2:*** Magnitude of the vector between *S2* and *S3*.

If ***Z*** is positive, then the method returns:

Acos (DP / (M1 * M2)) * (180 / PI)

If the ***Z*** is negative, then the method returns:

Acos (DP / M1 * M2)) * (-180 / PI)

4.3.2.2- Build Path

The "Build Path" method was designed to build a sequence of Spots that are on the shortest path found by the algorithm. The Shortest path will be generated by connecting these Spots together. It requires three inputs:

1- An empty list of Spots.

2-The Goal Spot.

3- The Start Spot.

First the method stores the Goal Spot into the empty list, then recursively stores parents of each Spot into the list starting from the Goal Spot until it reaches the Start Spot. Eventually, it returns the list, which is now filled with the sequence of Spots from The Goal Spot to the Start Spot. The route or path can be visualized in plan by drawing over the original CAD/BIM data. The path is simply produced by connecting the Spots with a polyline.

It is worth mentioning here that, in order to evaluate results of the algorithm, several test prototypes were generated in Processing programming language. After making sure that all parts of the algorithm work as intended, it was rewritten in C# for Circulation toolkit and in Swift 4 for indoor PNS. The test prototypes are available in Code appendix of the project.

4.4- Congestion Analysis prototype

Congestion is one human-level spatial quality of the interior space, which is directly related to circulation design of the building. In work environments, congestion can have a positive effect on human experience and improve performance of the workplace by activating the space and creating productive interaction between employees. On the other hand, in some public buildings like courtyards or malls, being informed about the locations within the space with a high chance of congestion helps architects and interior designers to decide on the position of certain amenities and security features. For instance, architects can improve positioning security or information centers in courtyards or locate amenities, small vendors and cafe shops utilizing information about highly congested places in the designed floorplan. Therefore, the purpose of the first prototype developed in this work is to identify the locations with a high chance of congestion inside a given floorplan by calculating the routes between locations defined by the architect and measuring the density of routes inside the space as discussed in (Nagy, et al. 2017).

All the analysis results provided in this section are about the floorplan of Architecture Hall building at the University of Washington, and a Grasshopper plugin called Mouse which was developed by the author is used in the analysis prototype (Figure 26).

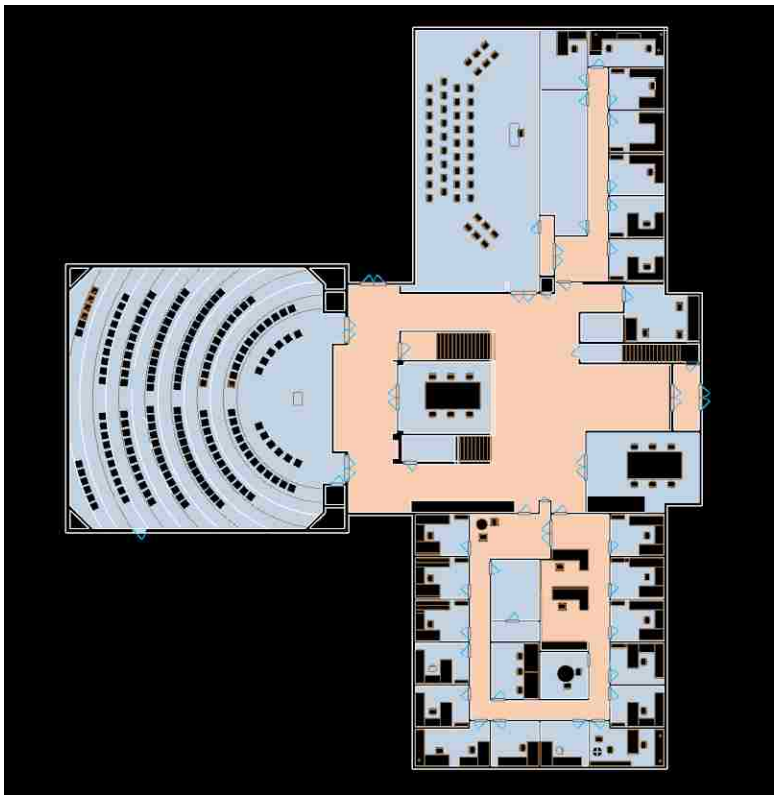

Figure 26: Architecture Hall first floor.

4.4.1- Navigation Model (*MouseTrap*)

After importing the floorplan that contains Room Data from Revit to Rhino and changing the hatch objects representing the Room Data, the first step is to generate the navigation model using the Room Data of the floorplan. This can be done using the MouseTrap component of the Mouse plugin. Required inputs of this component are: number of subdivisions of the navigation model's grid cells, which will be used for discriminating walkable-cells from wall-cells, Breps representing the Room Data of the floorplan, and the cell-size of the navigation model (Figure 27).

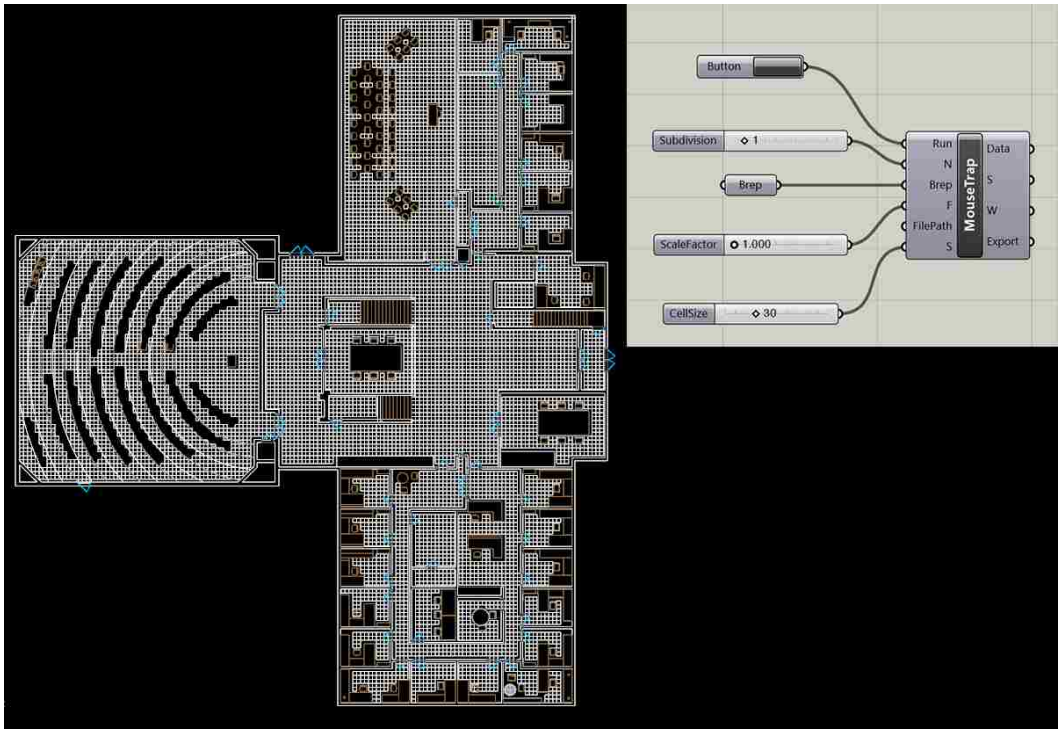

Figure 27: Architecture Hall navigation model generated by MouseTrap component.

The cell-size must be in the same units as the Rhino model. For instance, if the model is in the metric system, cell-size must be defined in the metric system too.

Since zones of the floorplan with higher chance of congestion are the ones that occupants of distinct parts of the building pass through to access other parts regularly, the metric defined for measuring the level of congestion in this prototype is density of routes inside a portion of the floorplan.

4.4.2- Super Cells

The first step for measuring route density was to define a grid with super cells on top of the plan. Since these cells are larger than the navigation model's cells, they are called super cells in this work. The main reason behind this step is that, congestion occurs between people when they pass within a certain distance of each other. In other words, if two individuals pass each other when

they have for example less than 3 meters distance, it is safe to say that congestion occurs between them, but if they pass each other while they have more than 5 meters distance from one another, no congestion happens between them. To the best of my knowledge, no research has been done for measuring the maximum distance between people when congestion occurs. Apart from that, congestion's maximum distance depends on functionality of the space and it changes in various projects. Therefore, it was considered as an input for the analysis tool, which the architect defines.

Therefore a 9 m^2 grid was generated on top of Architecture Hall example. Since areas of the walkable parts of the plan located in each of the super cells are different, the number of cells from the navigation model's walkable cells in each super cell determines the area used in measuring the route density in the super cell (Figure 28). This effectively ignores portions of the super cell extending outside the building.

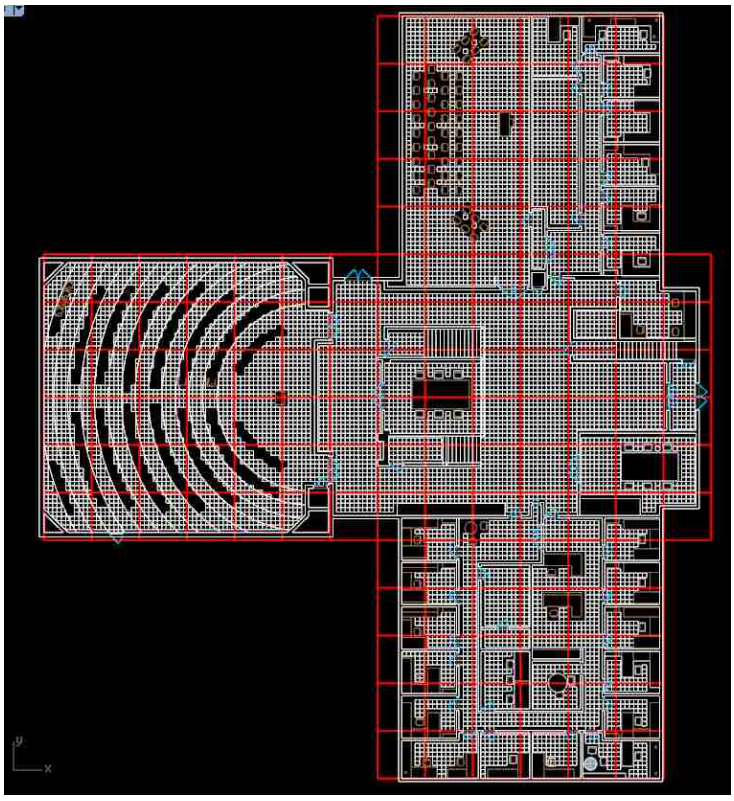

Figure 28: Super cells colored in red on top of the navigation model.

4.4.3- Start and Destination Points

Next, it is required to define start points and destination points for occupants of the building, as these become paths. This can be easily done by identifying initial locations of the occupants and all the possible places that they would go. For example, all the work stations on the first floor of Architecture Hall are considered as start points, since professors and students use them during a regular day. They are also parts of the destination points too, because it is fairly common for students to go visit their professors in their offices, or professors regularly go to their colleagues' offices for meetings. In addition, existing doors, stairs, elevators, and bath rooms are all feasible start points and destination points. In Figure 29, green circles represent occupants of Architecture Hall's first floor. Although all the work stations and classroom/auditorium seats are not occupied regularly, it is essential to identify all of them to the congestion analysis tool, since it will make results of the analysis more accurate.

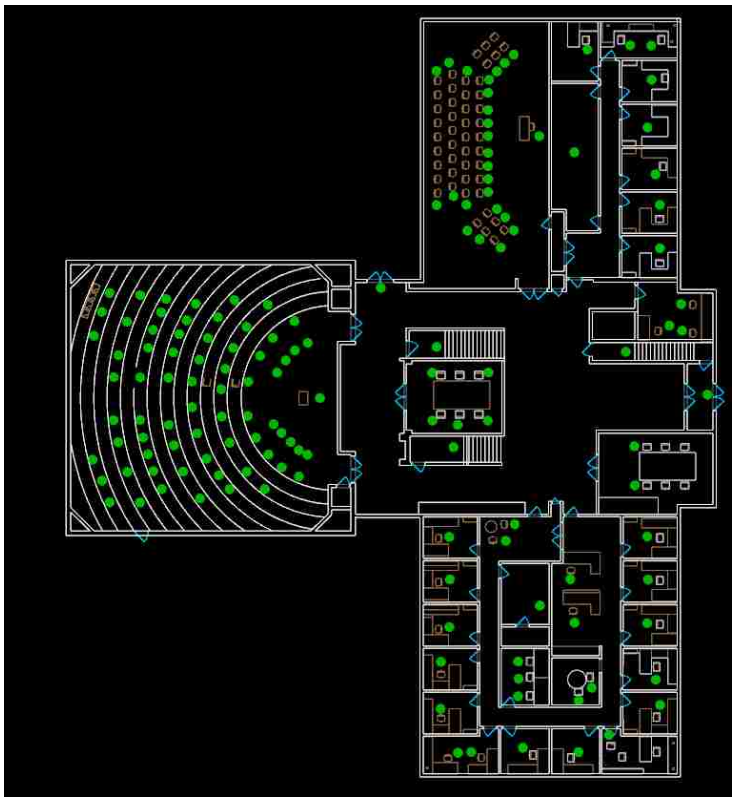

Figure 29: Possible start points and destination points in the floorplan.

4.4.4- AP-Theta* Solver (*Mouse*)

The second component of the Mouse plugin is an AP-Theta* solver called Mouse. Generating the data structure required for the algorithm, and execution of the algorithm itself will happen in this component. Thus, it will calculate the shortest walking distance between all the points defined as start locations and destinations and represent them as Polylines in Grasshopper. Apart from start and destination points, context, and wall-data of the navigation model, which are outputs of the MouseTrap component, plus the cell size used in generating the navigation model are required inputs of Mouse component. Figure 30 illustrates all the 23,350 possible routes in the Architecture Hall floorplan based on the position of the occupants and their possible destinations from Figure 29. In other words, if an occupant decides to go to each of the destination points, the path he or she would take is extremely similar to one of the paths generated by the Mouse component.

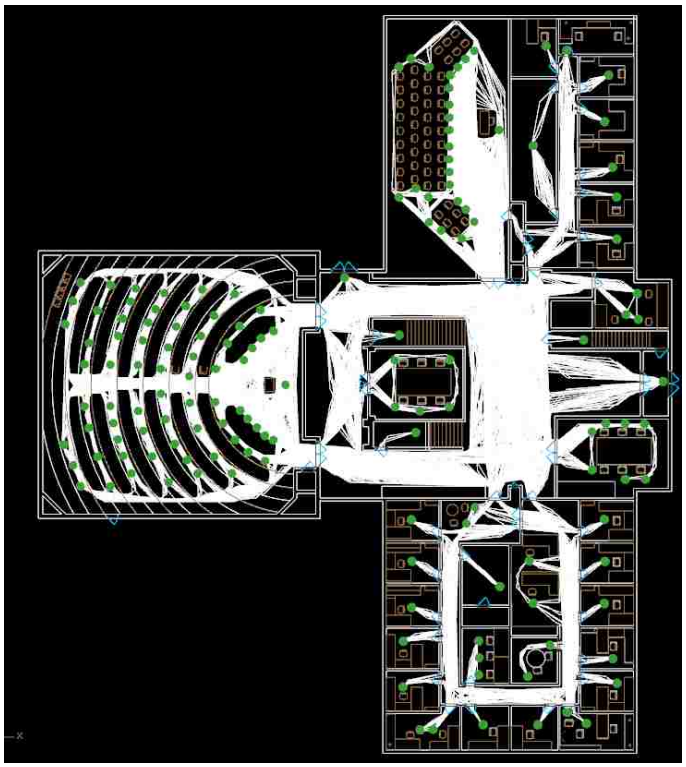

Figure 30: All the possible routes, generated between start and destination points

4.4.5- Route Density and Data Interpretation

As mentioned previously, the metric defined for identifying locations with a high chance of congestion is the density of routes in super cells. Route density in each of the super cells can be calculated as a function of the number of routes that pass through each super cell, and the area of walkable parts of the floorplan located in each super cell, as expressed in the following equation, where $\boldsymbol{D}$ represents density of routes, $\boldsymbol{R}$ is number of routes in each super cell, $\boldsymbol{N}$ is number of walkable navigation cells in super cell, and $\boldsymbol{S}$ is cell-size of the navigation model.

$$\boldsymbol{D = \frac{R}{N * S^2}}$$

In order to visualize the highly congested locations on the floor plan, a heat-map was generated on top of super cells' grid based on the route density of each super cell. A color in the range from dark blue to bright red was selected for each supper cell, such that the super cell with highest density value will be illustrated as red and the super cell with the lowest value will be illustrated as blue. Figure 31 demonstrates the congestion heat map of the first floor of Architecture Hall, and Figure 32 illustrates the number of routes, plus walkable areas in highly congested super cells around the core of the building.

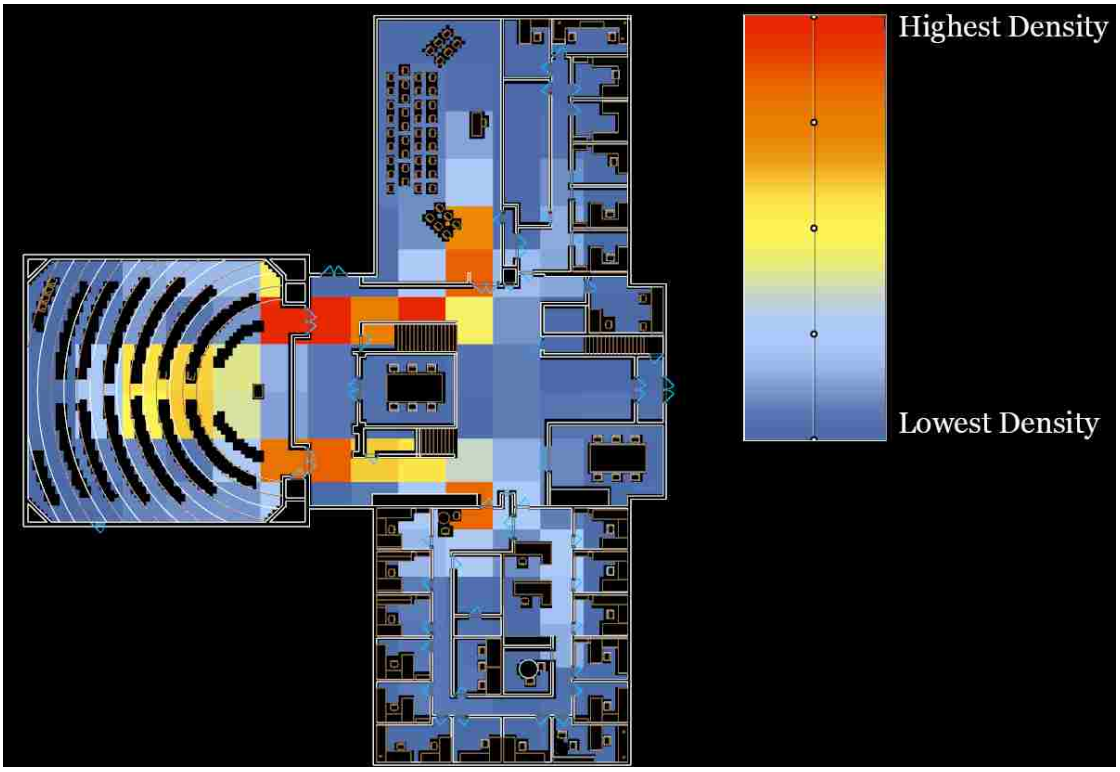

Figure 31: Heat map demonstrating congestion chance on a floorplan.

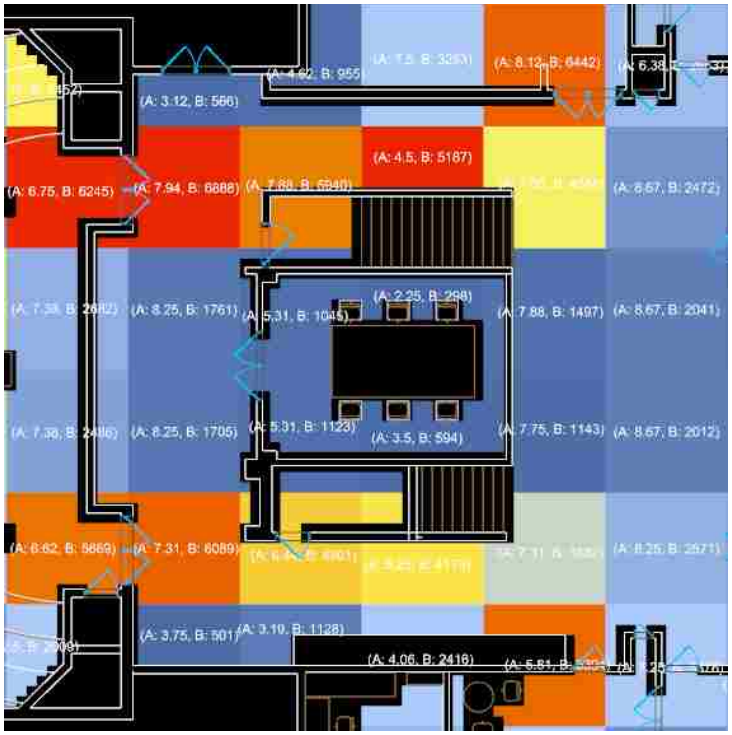

Figure 32: Walkable area in each (A) and number of routes (B) in super cells around the core.

Comparing the results of the analysis with what students and faculty experience during a regular day in the first floor of the building indicates that the results are fairly close to reality, especially during class changes. For instance, the areas around the core of the building are extremely congested locations from the morning until noon when all the classes are operating. In addition, the super cells that contain the stairs are highly congested locations too, since individuals need to use them in order to access the building's rest rooms. Therefore, clearly these results could provide some insights about the quality of the circulation design for the architects during the design process of the building.

4.5- Emergency Evacuation Prototype

Emergency evacuation is another situation which is extremely sensitive to the circulation design of the building. While complicated and inappropriate circulation design might increase casualties during a disaster, proper circulation design might save lives and reduce confusion during a catastrophe. Therefore, many regulations and rules related to emergency evacuation of a building are provided in building codes. However, rules related to congestion and traffic during egress, which can be more dangerous than the disaster itself in some scenarios, are missing in building codes (Canter 1980).

Literature related to emergency egress indicates that the most crucial parameter during evacuation of a building is time, which is affected by occupants' uncertainty on deciding which exit path to take, length of the route to exits, and congestion during an evacuation (Canter 1980). The circulation design of the building has a direct impact on all these parameters. Hence, the purpose of the second circulation analysis prototype developed in this research is to provide insights for architects during the design process related to evacuation time, and congested areas of the floor plan during emergency egress. As with the first prototype, the analysis in this section was done on Architecture Hall's first floorplan.

The first step in developing an emergency evacuation analysis tool was to design a new metric for identifying locations in the floorplan with a high chance of congestion, because density of routes is not a proper metric for this purpose. This is due to the fact that, no congestion in locations with high density of routes would occur during egress if people pass through them in various times.

Similarly, locations with lower density of routes might be highly congested locations during egress if many individuals pass them at the same time. Therefore, the highest density of bodies that occupy each super cell at the same time during the total egress process was chosen as a congestion measurement in this prototype. The equation below was used for calculating body density in each super cell. ***D*** represents highest density of bodies that attend each super cell at each time during the egress process, ***H*** is the highest number of bodies present at the same time in each super cell during the evacuation process, ***N*** is number of walkable navigation cells in each super cell, and ***S*** is cell-size of the navigation model.

$$D = \frac{H}{N * S^2}$$

In order to calculate the density of bodies in each super cell, it was required to simulate a complete emergency evacuation process on the floorplan.

4.5.1- Start Points and Destinations

In contrast to the first prototype in which defined start and destination points were one set of points and routes between all of them were required for the analysis, start points and destination points in the emergency evacuation prototype have to be distinct sets of points. All the possible locations of individuals in the floorplan, like work stations and offices, are start points, and all the exit doors are destination points during emergency egress. Figure 33 demonstrates defined start and destination points in the Architecture Hall example.

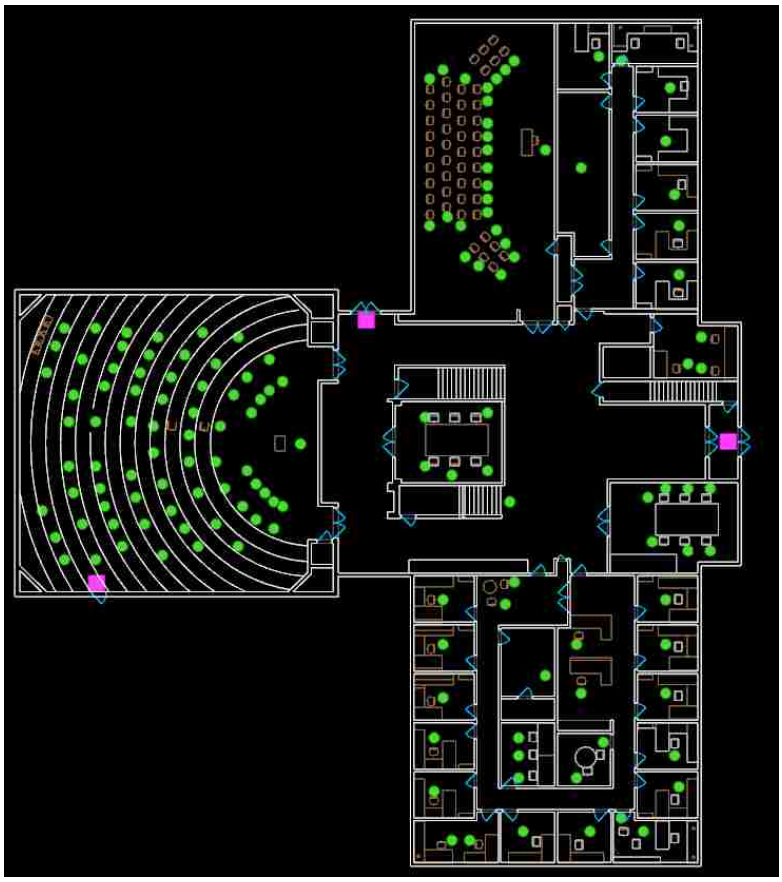

Figure 33: Start points (circles) and exit doors (squares) for emergency egress.

4.5.2- Route Generation

After defining start and destination points, the shortest paths to the exit door closest to each individual had to be generated. In order to do that, shortest paths between each start point and all the available exit doors were generated, and the shortest path was selected for each start point. The closest exit door for each occupant was found based on the shortest realistic walk between doors and occupants instead of Euclidean distance between them. Figure 34 illustrates the calculated shortest path between each occupant and the closest exist to him/her.

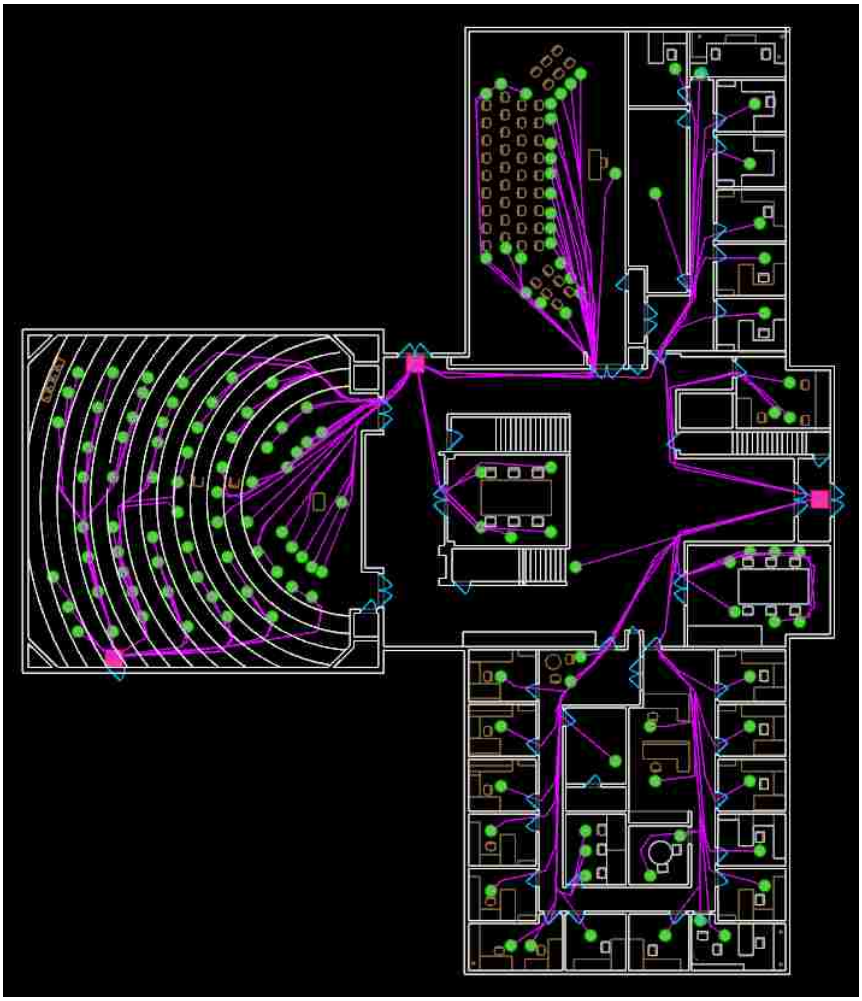

Figure 34: Shortest paths to closest exits for each occupant.

4.5.3- Body Density and Data Interpretation

As mentioned before, the body density in each super cell is equal to highest number bodies in each super cell divided by the walkable area of floorplan located in that super cell. The size of the super cells in this analysis was reduced to 4 m^2, since congestion during emergency egress occurs in smaller areas than regular congestion. To calculate the highest number of bodies that a super cell ever experiences during the evacuation, the entire process has to be simulated. In other words, all the individuals must start evacuating the building at the same time using the paths to the closest exits, which were generated by the AP-Theta* solver for them. The simulation also requires

specialized human travel speed data during evacuation. Much research has been done on measuring human's travel speed on distinct types of surfaces (horizontal, incline, etc.). Several factors influence a person's movement speed, like age, gender, grouping (family, friends, etc.), physical ability, and environmental conditions. However, based on the experimental travel speed data base provided in (Fahy and Proulx 2001) human movement speed during emergency egress inside public buildings (theaters and educational buildings) are between 0.33 m/s and 2.33 m/s. Hence, 1.33 m/s was selected as the average speed of occupants in the Architecture Hall example.

When the simulation starts, occupants begin to move to the closest exit at the same time with the defined travel speed. While time continues to run, bodies move from one super cell to another until they reach their exits. Therefore, the number of bodies in each super cell changes. At each second, the algorithm computes the number of bodies in each super cell. This is compared to the maximum that super cell has seen and stored if larger. The highest numbers will always be stored and replace the lower numbers. The moment the last occupant reaches an exit, time will stop, and the density of bodies in each cell will be calculated. For visualization purposes, an upper limit was defined for the body density of each cell. In other words, cells with density higher than the upper limit will be illustrated as red, while a color between red and dark blue will be selected for the rest of the super cells. For instance, the upper limit selected for Architecture Hall example is 5 bodies in a super cell (4 m^2) as illustrated in Figure 35.. Furthermore, Figure 36 demonstrates the number of walkable navigation model cells, as well as the highest number of bodies that each super cell around the core of the building experienced during the whole simulation.

It is worth mentioning here that, the value for heatmap's upper limit, and size of the super cells are inputs for the analysis tool, which will be defined by the architect based on the functionality of the project.

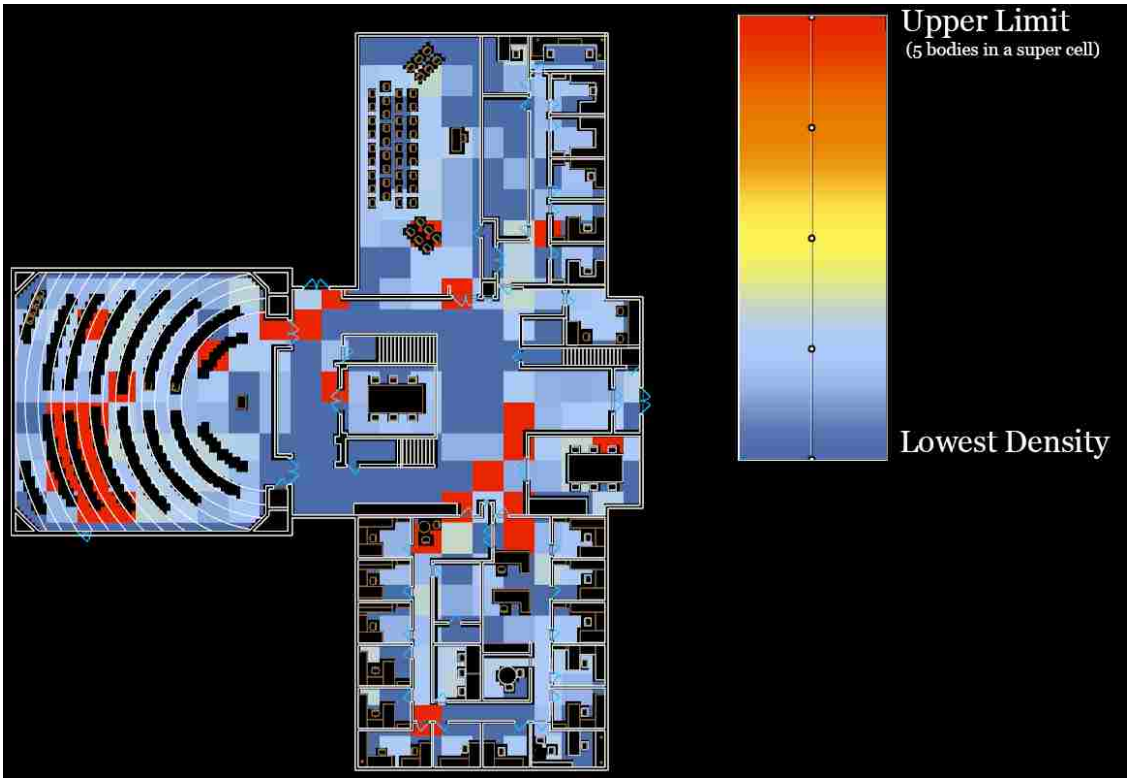

Figure 35: Heat map illustrating the congested locations of the floorplan during the egress process.

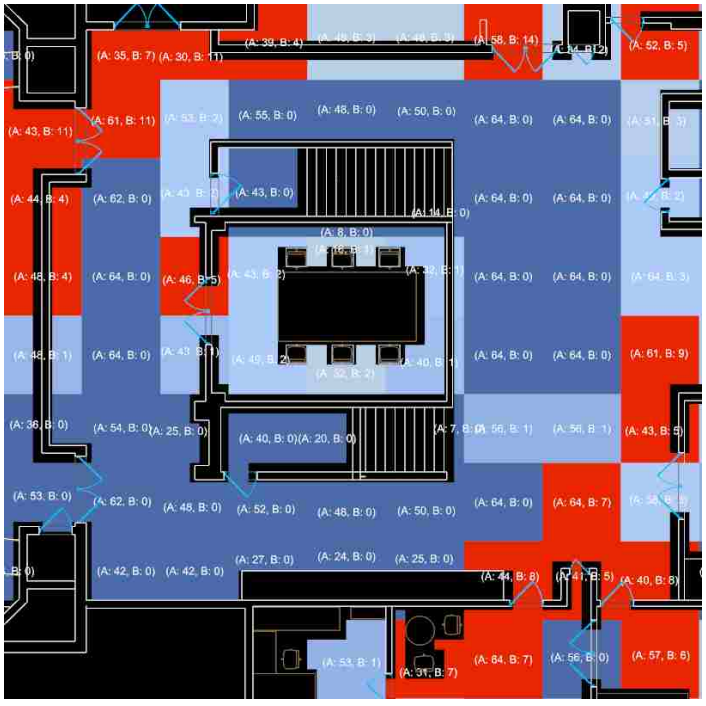

Figure 36: Number of walkable navigation model cells (A) and highest number of bodies (B) in super cells around the core.

Associating the heat map generated by the analysis prototype with what may happen in an actual emergency evacuation scenario indicates that the results of the analysis are believable, since most of the red areas are locations with limited walkable area close to crowded locations of the floorplan. For example, most of the inner doors of the space are colored as red since they have constrained space around them and many people arrive to those areas at the same time, also the locations between the seats of the amphitheater are extremely congested locations in real world which was identified by the algorithm. Tools like the emergency evacuation analysis prototype in this project may provide useful information about emergency evacuation of a building and identify dangerous locations during emergency egress.

4.6- Personal Navigation Assistant

People avoid going to places that they feel they will get lost. This is similar to an actual barrier that stops people from using buildings with complicate circulation. This problem was addressed by Pedestrian Navigation Systems like Google Maps, or Maps by Apple in outdoor environment and urban layouts. However, in order to navigate complex public buildings, individuals must rely on their general understanding of structure and layout of the building, and their past experiences. Hence, an indoor pedestrian navigation system was developed using the in the third prototype.

In this prototype, circulation toolkit will be used to address three main challenges in the way of developing a practical PNS for indoor environment. Hence, navigation model generated by circulation toolkit will be used to identify walkable areas of the navigation model and discriminated them from location of furniture and walls inside the floorplan.

Apart from that, pathfinding algorithm of the circulation toolkit will be used to generate paths that connect any two locations inside the building without any need for predefined network of routes.

Finally, Data from smartphone's sensors will be utilized to track movement of the user and locate him/her inside the building using dead reckoning localization method.

4.6.1- Visualizing the Floorplan

The first step in developing the navigation application was to visualize the floorplan in the application. A vector image is better than a raster image for this purpose, because it does not pixelate when a user zooms in on a specific part of the plan. Fortunately, the iPhone API supports two vector image formats, which are SVG and PDF. However, in order to visualize each of these formats various user interface elements had to be used. To show a SVG image on the iPhone display, a WKWebView element must be used and to show a PDF, a UIImageView element has to be used.

Since the purpose of WKWebView is to provide the possibility of creating an internet browser, it does not support all the required features for developing an application that allows user to interact with the display. For instance, it does not provide the possibility to access information about the location of the user's touch on the screen. Therefore, it was decided to use PDF as the format for visualizing the floorplan in the application. Figure 37 illustrates the PDF generate from the first floorplan of Architecture Hall for the navigation application.

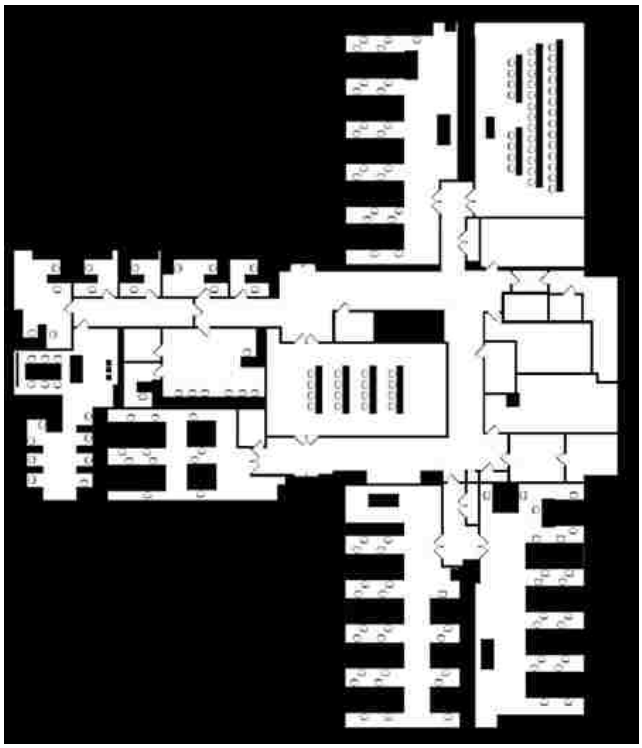

Figure 37: PDF file generated from the floorplan.

4.6.2- Proper Data (Maps) From Inside the Building

In contrast to outdoor environments and urban layouts, where huge databases of maps and information about size and location of obstacles and buildings are available, indoor spaces often lack navigational maps and plans. This problem was addressed by the automated process of generating the navigation model from the Room-Data of the floorplan developed for the circulation analysis toolkit in this project. However, the navigation model was generated in Grasshopper using the C# programming language. Since C# objects are not compatible with the Swift programming language, in order to transfer the navigation model to the iOS platform, a JSON object was created in Grasshopper, which contains wall property values of the navigation model's cells.

The JSON object consists of a two-dimensional array of zeros and ones that represent the sequence of navigation model 's cells in rows and columns. Zeros represent walkable cells and ones represent unwalkable cells in the JSON object. Figure 38 shows the value representing each cell in the JSON object.

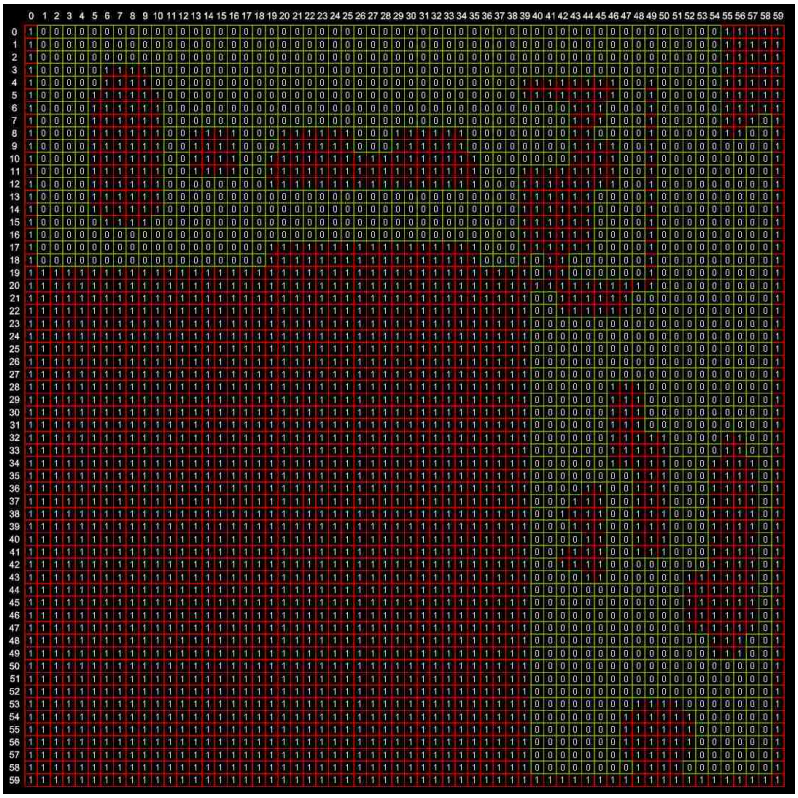

Figure 38: wall values of the JSON object.

Eventually, the JSON object was deserialized in the XCode, which is the Integrated Development Environment for all the Apple products like iOS or MacOS and generates the same two-dimensional array as the one generated in the first place in Grasshopper. It is possible to create the navigation model for the smartphone application using this information. To do that, the width of the PDF file generate from the floorplan was extracted from the UIImageView element that contains it, then that number was divided to the number of columns of the array, which will return the size of the navigation cells on the iPhone display. Then, a UIView element with the same size as the UIImageView that contains the visualization PDF was added as a sublayer of the UIImageView and placed on top of it. This UIView element is the container of the navigation model. Eventually, nodes and cells of the navigation model were generated inside this UIView. Figure 39 shows the sequence of layers used in the process of visualizing the floorplan on the phone's display and generating the navigation model on top of the visualization layer. The UIView that contains the navigation model also detects the user's touch interaction with the phone's screen and converts the location of user's touch to the closest node of the navigation model to that

location. It is worth mentioning here that in the final product, similar to the circulation analysis prototypes, cells and nodes are hidden to the user.

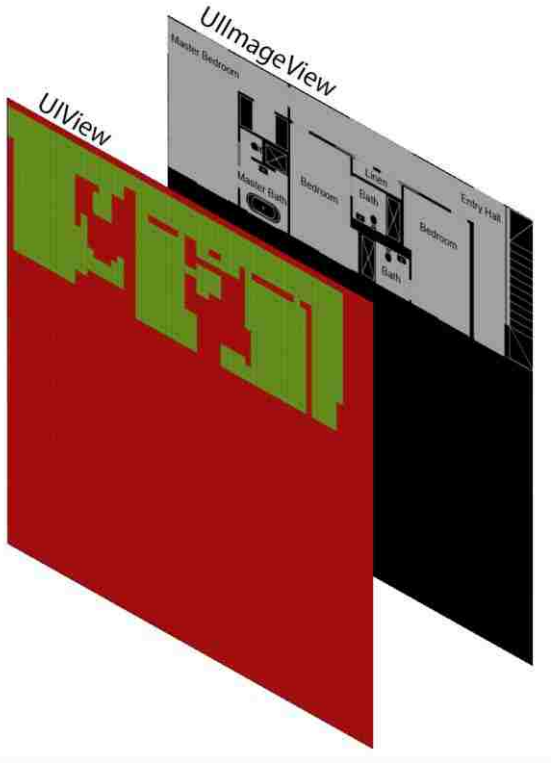

Figure 39: UIView elements of the navigation App.

4.6.3- Localization

Dead reckoning is a localization method that calculates the location of a user based on a known initial position, direction of the user's movement, and distance of the user's motion. Figure 40 demonstrates the dead reckoning localization method. In this technique, a recent location of the user will be calculated by moving from the previous location using a vector created with direction and distance of the movement.

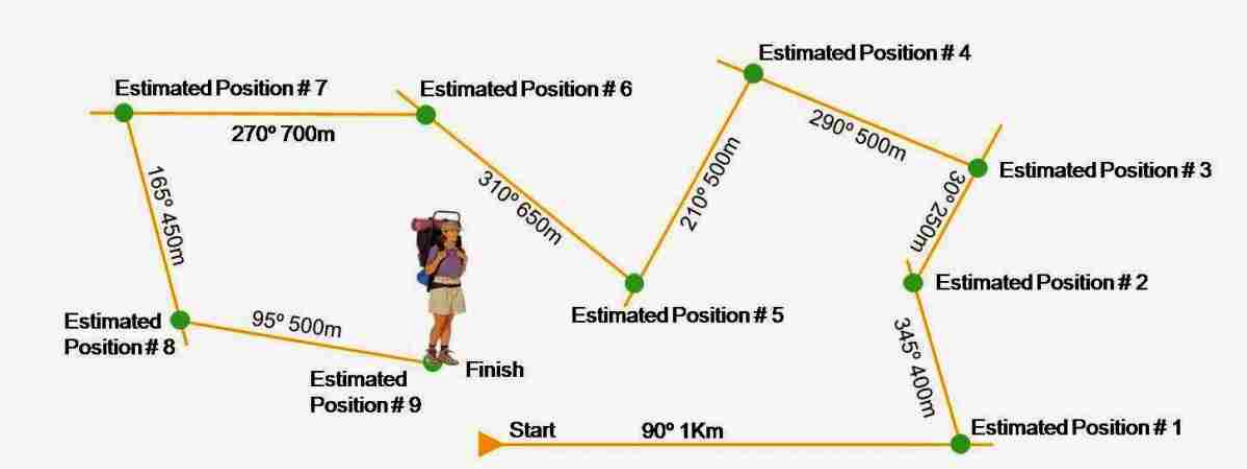

Figure 40: Dead reckoning localization process.

4.6.3.1- Initial Location

In order to define the initial location of the user for the navigation application, three methods were designed, after:

First, the user can identify his initial location by touching his approximate location in the floorplan displayed on the phone's screen (Figure 41).

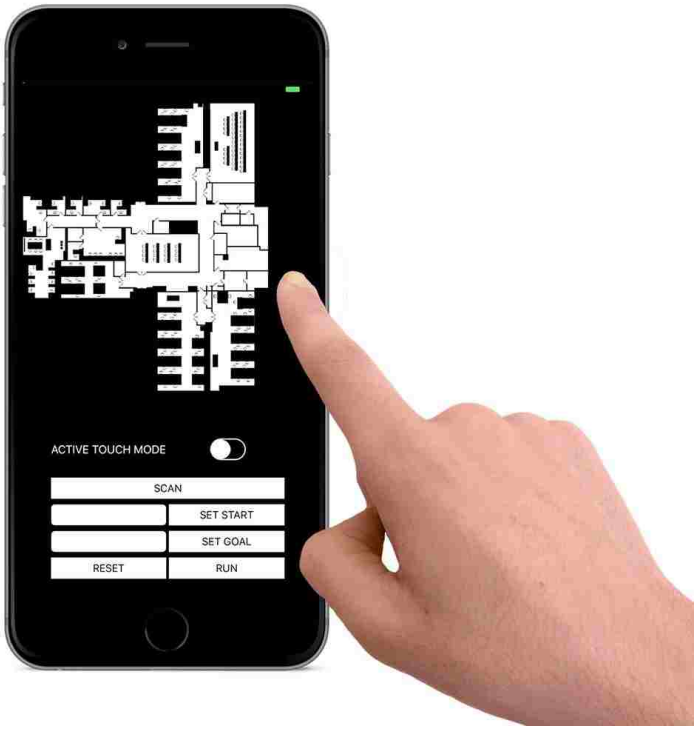

Figure 41: Initial location defined by touching phone's screen.

The second method for defining the start location is by entering the name of that location. As mentioned before, “name” is a string property of a spot object, which is empty by default. However, in this prototype the “name” attribute of one spot-object located in each room was assigned to the name of that room. Hence, the second method for defining the start location is by entering the name of any room in the UITextBox element provided in front of the “SET START”. Eventually, by pressing the “SET START” the spot with “name” attribute equal to what was entered in the UITextBox will be selected as the initial location (Figure 42).

Figure 42: Defining the start location by entering the name of that location.

The final technique that allows users to define their initial location is by scanning a QR Code provide in their location using the phone's camera. QR Codes provided in a predefined location in each room of the floorplan. These QR Codes have the "id" attribute of the nodes that they are associated with embodied in them. Pressing the "SCAN" button provides the possibility to scan the QR Code (Figure 43).

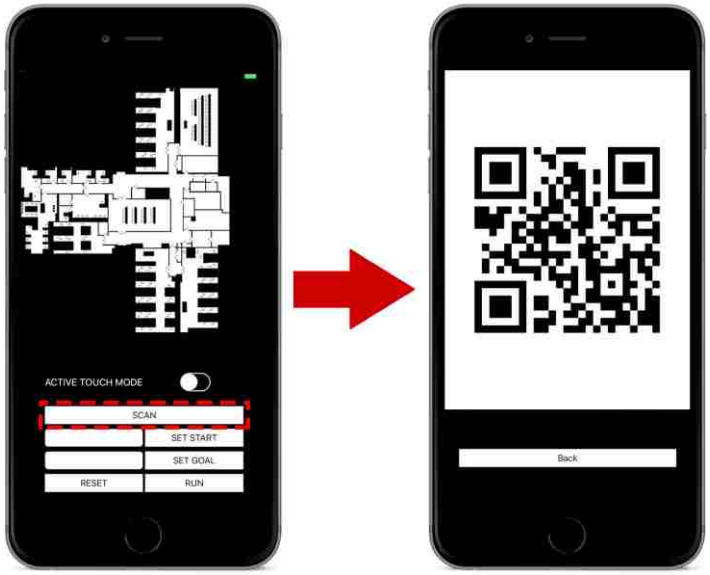

Figure 43: Defining the start location of the user by scanning a QR Code.

After defining the initial location by using one of these three techniques, the start location will be displayed on the phone's screen as a green circle (Figure 44).

Figure 44: The start location displayed on the phone's screen by green circle.

4.6.3.2- Destination Location

In the next step, the user's destination in the floorplan has to be defined. The navigation application provides two methods for defining the goal location. First, similar to first method for defining the initial location of the user, it is possible to identify the destination by touching the approximate location of the end point on the floorplan.

Second, the goal point can be defined by entering the name of the location in the "UITextBox" element provided in front of the "SET GOAL" button and pressing the button. After identifying the destination, it will be illustrated by a red circle on the phone's display (Figure 45).

Figure 45: The destination displayed on the phone's screen by red circle.

4.6.3.3- Direction and Distance of the Motion

By pressing the "RUN" button after designating the initial and goal locations, the application generates the path that connects them using the Theta* algorithm. In order to track the location of the user with the dead reckoning localization method, information about the direction and distance of the user's motion are required in addition to initial location. Fortunately, iOS *SDK*[2] provides the possibility to access this information.

"CoreLocation" is the default library of the iOS platform that supplies information and services for determining an iPhone's geographical location, altitude, orientation, or position relative to a close *iBeacon*[3]. All the available onboard hardware, including Wi-Fi, GPS, Bluetooth, magnetometer, barometer, and cellular hardware will be employed by the framework to gather data (Apple 2018).

The "locationManager" method of the "CoreLocation" library is the method that generates information about the phone's orientation in the format of angle from true north. In order to visualize the orientation of the phone relative to the orientation of the floorplan in the navigation application, a small black circle is generated inside the green circle representing the location of the user. The black circle rotates around the center of the green circle based on the information generated by the "locationManager" method (Figure 46). It is worth mentioning here that the PDF representing the floorplan of the building in the navigation application is oriented with true north aligned with top of the phone (Figure 47).

[2] Software Development Kit.
[3] iBeacon is an Apple technology that creates a way for providing location-based information and services to iPhones and other iOS devices.

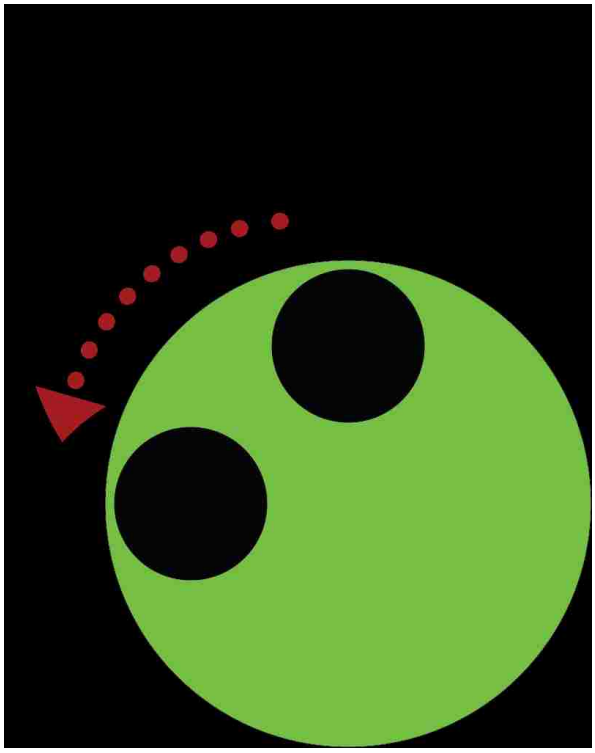

Figure 46: Orientation of the phone's visualization.

Figure 47: Orientation of the floorplan's visualization.

"CoreMotion" is the default library of the iOS platform that reports data related to motion and the environment from the onboard hardware of iOS devices, including accelerometer, gyroscope, magnetometer, and barometer. While some services of this framework let developers access raw values recorded by the hardware, others provide processed version of those values (Apple 2018).

The "CMPedometer" object of the "CoreMotion" library, which provides information about distance of the motion, number of steps, and type of user's activity (running, walking, taking stairs, etc.) was instantiated and the motion's distance was combined with the motion's direction generated by "locationManager" to create the user's motion vector. Eventually, the green circle that represents the location of the user moves using the generated motion vector (Figure 48).

Figure 48: User's localization inside the navigation application.

5- Discussion and Conclusion

In this project I have done a survey on how the circulation design of a building might affect its performance during use. I explained three aspects of circulation that influence design of the building.

First, I described the concept of congestion and its effects on human-experience inside a building. Then I clarified how circulation design during the architectural design process affects occupant safety during a disaster, and how more sensitive circulation design is essential for reducing casualties during a catastrophe. Eventually, I explained the concept of wayfinding in complex architectural settings and how people avoid confusing and complicated environments.

Next, I proposed an algorithmic approach to circulation and developed a computational toolkit that can be used during the design phase of the building, or after construction to improve its circulation performance.

The circulation toolkit extracts a navigation model from an architectural floorplan in the format of CAD or BIM and generates optimal and realistic looking routes that mimic human movement behavior between any two locations inside the building.

Finally, I developed three prototypes using the circulation toolkit to provide some information and insight about the challenges that interior circulation creates for performance of the building.

5.1- Advantages and Opportunities

The main advantages of the proposed circulation toolkit in comparison to other circulation computational tools are:

1. The proposed toolkit is relatively quick and computationally cheap in comparison to agent-based crowed simulation tools that are popular in the architecture industry. Although, with emergence of modern graphic processors and GPU computing methods,

agent-based crowd simulations are not as slow and time consuming as they used to be, these tools are still slower than the proposed method.

2. The proposed circulation toolkit mimics human behavior inside a building more accurately than agent-based crowd simulation tools. This is due to the fact that human movement behavior happens when the destination location and the path to it are already imagined by the person. However, agents of agent-based tools are moving inside the space mindlessly according to rules that define their interaction with other agents and the environment of the simulation in most scenarios.
3. The pathfinding algorithm used in developing the circulation toolkit, which is called Theta* is the most compatible pathfinding algorithm for architectural analysis, because in contrast to other pathfinding algorithms, which are limited to the edges of the navigation grid and generated zig zag and unrealistic looking paths, Theta* generates routes which are similar to human movement behavior.
4. Since the circulation toolkit is developed as a plugin for design software, that is commonly being used in most architecture firms and schools, it is more likely for it to become part of standard architectural framework than agent-based simulation tools, which mostly are expensive independent software programs. In addition, data transfer problems between architectural design software and simulation tools do not exist for the circulation toolkit.
5. While considering furniture of the building is a rare feature in most indoor pedestrian navigation systems, in the PNS prototype developed in this project all furniture of the building become part of the navigation model. The method used for generating the pedestrian navigation model in this project also provides the possibility for architects to consider furniture of the building in their analysis.
6. The dead reckoning localization method used in the PNS prototype for tracking the user omits expensive infrastructure required for commonly used localization methods like indoor GPS, Wi-Fi finger printing, Wi-Fi triangulation, etc.

5.2- Directions for Future Research

As mentioned before, the fundamental requirement of using the circulation is having access to architectural floorplan in the computational formats like CAD or BIM. While this data is available for most recently constructed buildings, some of the older ones lack this data, since their architectural documents and drawings were developed originally on paper. There are huge databases of scans from public buildings' floorplans available online, which are not useable by the circulation toolkit. Hence, developing an automated process for generating navigation model from architectural documents and drawings in formats other than CAD or BIM is one of the areas to continue this research.

While agent-based crowed simulation tools are slower and more computationally expensive than the circulation toolkit, they are becoming faster day by day with the advent of extremely fast and inexpensive CPUs and GPUs. Therefore, computational power and speed is hardly going to be a problem for agent-based simulation tools. Consequently, comparing the results of analysis done by the circulation toolkit with the results of the same analysis done by agent-based simulation tools to evaluate their precision and speed is one area of required research for the future. Moreover, integrating the circulation toolkit approach with agent-based simulation tools to develop a new approach to analyzing and evaluating human behavior inside architectural space is an interesting area of research for the future.

Evaluating the results of circulation analysis using the circulation toolkit based on actual human experience inside the space requires more research for the future too. Although the circulation toolkit mimics human movement behavior accurately, there are many parameters in real-world events that influence individuals, which do not exist in the computational simulation tools that mimic human behaviors and experiences. Hence, in order to evaluate how accurate and reliable the results of the circulation analysis are, in future research they have to be compared against what people actually feel in the space.

Finally, although localizing the user in the pedestrian navigation system prototype by the dead reckoning method omits the need for expensive infrastructures inside the building, this localization method might not be accurate enough for a practical indoor PNS. This is due to the fact that magnetic fields of electrical objects and devices present in the space influence the gyroscope of the phone. Therefore, direction information generated by the phone's gyroscope alone always contains some error. In addition, since information about the distance of the user's

movement is an approximation based on numbers of steps he/she takes, it also contains some level of errors. Although this error is not significant for each step, it accumulates during long walks, therefore the user might end up in wrong locations after travelling long distances. Consequently, finding a more accurate localization method, or developing new algorithms that generate distance and direction information from the gyroscope and accelerometer more precisely is another area of future research.

References

Abdelgawad, H., B. Abdulhai, G. Amirjamshidi, M. Wahba, C. Woudsma, and M. J. Roorda. 2011. "Simulation of exclusive truck facilities on urban freeways." *Journal of Transportation Engineering* 137: 547-562.

Accelerometer. 2018. *Wikipedia*. 01 15. Accessed 01 15, 2018. https://en.wikipedia.org/wiki/Accelerometer.

Ahlquist, Sean, and Achim Menges. 2011. "Methodological Approach for the Integration of Material Information and Performance in the Design Computation for Tension-Active Architectural Systems." *In Proceedings of the 29th Conference on Education in Computer Aided Architecture Design in Europe (eCAADe 2011)*. Ljubljana: eCAADe (Education and Research in Computer Aided Architectural Design in Europe) and UNI Ljubljana, Faculty of Architecture. 800-807.

2018. *Apple*. Accessed 03 07, 2018. https://developer.apple.com/documentation/corelocation.

2018. *Apple*. Accessed 03 08, 2018. https://developer.apple.com/documentation/coremotion.

Bellman, Richard. 1958. "On a routing problem , no. 1 (1958): 87-90." *Quarterly of applied mathematics 16* 87-90.

Brown, J.S., and P. Duguid. 2000. *The Social Life of Information*. Boston: Harvard Business School Press.

Canter, David V. 1980. *Fires and human behaviour*. John Wiley & Sons.

Carpman, Janet R., and Myron A. Grant. 2016. *Design that cares: Planning health facilities for patients and visitors*. Vol. 142. John Wiley & Sons.

Chen, Guoliang, Xiaolin Meng, Yunjia Wang, Yanzhe Zhang, Peng Tian, and Huachao Yang. 2015. "Integrated WiFi/PDR/Smartphone using an unscented kalman filter algorithm for 3D indoor localization." *Sensors 15* 24595-24614.

Chen, Po-Han, and Feng Feng. 2009. "A fast flow control algorithm for real-time emergency evacuation in large indoor areas." *Fire Safety Journal 44* 732-740.

Chen, Zhenghua, Han Zou, Hao Jiang, Qingchang Zhu, Yeng Chai Soh, and Lihua Xie. 2015. "Fusion of WiFi, smartphone sensors and landmarks using the Kalman filter for indoor localization." *Sensors 15* 715-732.

Claxton, G. 2000. *Hare Brain, Tortoise Mind: How Intelligence Increases When You Think Less*. New York: Harper Perennial.

Diakité, Abdoulaye A., and Sisi Zlatanova. 2016. "EXTRACTION OF THE 3D FREE SPACE FROM BUILDING MODELS FOR INDOOR NAVIGATION." *ISPRS Annals of Photogrammetry, Remote Sensing & Spatial Information Sciences 4*.

Diaz, Estefania Munoz. 2015. "Inertial pocket navigation system: Unaided 3D positioning." *Sensors 15* 9156-9178.

Dijkstra, Edsger W. 1959. "A note on two problems in connexion with graphs." *Numerische mathematik 1* 269-271.

Doran, Jim E., and Donald Michie. 1966. "Experiments with the graph traverser program." *In Proceedings of the Royal Society of London A: Mathematical, Physical and Engineering Sciences*. The Royal Society. 235-259.

Drucker, P. 1959. *Landmarks of Tomorrow*. New York: Harper.

Fahy, Rita F., and Guyléne Proulx. 2001. "Toward creating a database on delay times to start evacuation and walking speeds for use in evacuation modeling." *2nd international symposium on human behaviour in fire*. Boston, MA, USA. 175-183.

Feynman, Richard, Michael Gottlieb, and Ralph Leighton. 2013. *Feynman's Tips on Physics, A Problem-Solving Supplement to the Feynman Lectures on Physics*. Basic Books.

Goetz, Marcus, and Alexander Zipf. 2011. ""Formal definition of a user-adaptive and length-optimal routing graph for complex indoor environments.", no. 2 (): ." *Geo-Spatial Information Science 14* 119-128.

Gyroscope. 2018. *Wikipedia*. 01 17. Accessed 01 17, 2018. https://en.wikipedia.org/wiki/Gyroscope#Heading_indicator.

Hart, Peter E., Nils J. Nilsson, and Bertram Raphael. 1968. "A formal basis for the heuristic determination of minimum cost paths." *IEEE transactions on Systems Science and Cybernetics 4* 100-107.

Johnson, G. Miles, and William W. Happ. 1977. "Digital simulation for detecting congestion in hospital facilities." *In Proceedings of the 9th conference on Winter simulation-Volume 2*. Winter Simulation Conference. 848-853.

Jones, Willie D. 2010. *IEEE SPECTRUM*. 01 29. Accessed 01 17, 2018. https://spectrum.ieee.org/semiconductors/devices/a-compass-in-every-smartphone.

Khoshelham, Kourosh, and Sisi Zlatanova. 2016. "Sensors for indoor mapping and navigation." 655.

Korf, Richard E. 2010. *Artificial intelligence search algorithms*. Chapman & Hall/CRC.

Korf, Richard E., Michael Reid, and Stefan Edelkampz. 2001. "Time complexity of iterative-deepening-A∗." *Artificial Intelligence 129* 199-218.

Li, Fan, Chunshui Zhao, Guanzhong Ding, Jian Gong, Chenxing Liu, and Feng Zhao. 2012. "A reliable and accurate indoor localization method using phone inertial sensors." *In Proceedings of the 2012 ACM Conference on Ubiquitous Computing*. ACM. 421-430.

Lindsay, Greg. 2013. "Engineering serendipity." *New York Times* 5.

Liu, L., and S. Zlatanova. 2011. "A "door-to-door" path-finding approach for indoor navigation." *In Proceedings Gi4DM 2011: GeoInformation for Disaster Management*. Antalya, Turkey.

Magnetometer. 2018. *Wikipedia*. 01 17. Accessed 01 17, 2018. https://en.wikipedia.org/wiki/Magnetometer.

Makri, A., S. Zlatanova, and E. Verbree. 2015. "An approach for indoor wayfinding replicating main principles of an outdoor navigation system for cyclists." *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences, Indoor-Outdoor Seamless Modelling, Mapping and Navigation*. Tokyo: the International Society of Photogrammetry and Remote Sensing (ISPRS).

Mizaralkora. 2017. *tumblr*. 09 20. Accessed 01 17, 2018. http://hugsforbears.tumblr.com/post/61746588103.

Mortari, Filippo, Sisi Zlatanova, Liu Liu, and Eliseo Clementini. 2014. ""Improved geometric network model"(IGNM): A novel approach for deriving connectivity graphs for indoor navigation." *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences 2* 45.

Nagy, Danil, Villagi Lorenzo, Stoddart James, and Benjamin David. 2017. "The Buzz Metric: A Graph-based Method for Quantifying Productive Congestion in Generative Space Planning for Architecture." *Technology| Architecture+ Design1* 186-195.

Nash, Alex, Daniel Kenny, Koenig Sven, and Felner Ariel. 2007. "Theta^*: Any-Angle Path Planning on Grids." *In AAAI*. Vancouver: AAAI Press. 1177-1183.

Newell, Allen, and Herbert Alexander Simon. 1972. *Human problem solving*. Vol. 104. 9 vols. Englewood Cliffs, NJ: Prentice Hall .

Passini, Romedi. 1996. "Wayfinding design: logic, application and some thoughts on universality." *Design Studies* 17 (3): 319-331.

Qian, Jiuchao, Ling Pei, Jiabin Ma, Rendong Ying, and Peilin Liu. 2015. "Vector graph assisted pedestrian dead reckoning using an unconstrained smartphone." *Sensors 15* 5032-5057.

Reckoning, Dead. 2018. *Time and Navigation*. Accessed 01 15, 2018. https://timeandnavigation.si.edu/navigating-at-sea/navigating-without-a-clock/dead-reckoning.

Rindler, W. 2013. *Essential Relativity: Special, General, and Cosmological*. Springer.

Silverman, Rachel Emma. 2013. "The science of serendipity in the workplace." *Wall Street Journal* B6.

Stentz, Anthony. 1994. "Optimal and efficient path planning for partially-known environments." *In Proceedings of the 1994 IEEE International Conference on Robotics and Automation*. IEEE. 3310-3317.

Suchman, L. 2000. *Making a case: 'knowledge' and 'routine' work in document production, in P. Luff, J. Hindmarsh and C. Heath (eds): Workplace Studies: Recovering Work Practice and Informing System Design* . Cambridge: University Press, Cambridge.

Tinder, Richard F. 2007. *Relativistic Flight Mechanics and Space Travel: A Primer for Students, Engineers and Scientists*. Morgan & Claypool Publishers.

Worthington, Arthur M. 1906. *Dynamics of Rotation*. London, New York, Bombay: Longmans, Green and Co.

Wu, Hua, Alan Marshall, and Wai Yu. 2007. "Path planning and following algorithms in an indoor navigation model for visually impaired." *In Second International Conference on Internet Monitoring and Protection, ICIMP*. IEEE. 38-38.

Xu, Man, Shuangfeng Wei, and Sisi Zlatanova. 2016. "AN INDOOR NAVIGATION APPROACH CONSIDERING OBSTACLES AND SPACE SUBDIVISION OF 2D PLAN." *International Archives of the Photogrammetry, Remote Sensing & Spatial Information Sciences 41*.

Xu, Man, Shuangfeng Wei, Sisi Zlatanova, and Ruiju Zhang. 2017. "BIM-BASED INDOOR PATH PLANNING CONSIDERING OBSTACLES." *ISPRS Annals of the Photogrammetry, Remote Sensing and Spatial Information Sciences 4* 417-417.

Xu, Zhiguang, and Michael Van Doren. 2011. "A Museum Visitors Guide with the A* pathfinding algorithm." *In International Conference on Computer Science and Automation Engineering (CSAE)*. IEEE. 62-66.

Yap, P. 2002. "Grid-based path-finding." *In Proceedings of the Canadian Conference on Artificial*. Heidelberg, Berlin: Springer. 44-55.

Zhu, Q., Q. Xiong, S. Zlatanova, Z. Du L. Huang, and Y. Zhou. 2013. "Multi-dimensional indoor location information model." *Acquisition and Modelling of Indoor and Enclosed Environments 2013*. Cape Town, South Africa: ISPRS Archives Volume XL-4/W4.

Zlatanova, Sisi, and Safiza Suhana Kamal Baharin. 2008. "Optimal navigation of first responders using DBMS." *In 3rd international conference on information systems for crisis response and management 4th international symposium on geoInformation for disaster management*. Washington, DC: ISCRAM. 541-54.