1 Summary

Flash sales are increasingly becoming the norm for brick-and-mortar stores, mainly in an effort to redirect sales away from online stores. These strategies cause surges of shoppers during these sales, resulting in many accidents that both damage products and injure people as they rush to purchase their target goods. The importance of store layouts to minimize the degree of such damage in stores has thus emerged, like in this problem.

For our model, we sought to minimise the financial losses to the store from these item damages, referred to as the **total loss, the dependent variable.** We first identified the main factors in the surge of shoppers: the positions of the most desirable goods, the cashiers, and the presence of bottlenecking physical elements. The desirability of the goods was assessed based on an index coined the **Gross Happiness Index (GHI)**, which took into account discounts, consumer rating and quantity remaining of each product. This formed the basis of our model's functionality.

The layouts were converted into a 48x48 matrix to simplify computations, with each tile representing a 1m by 1m square.

We then explored various ways in which consumers may damage items, such as knocking them out of their shelves, fighting for the last remaining stocks and walking through the crowd with the items. We realised that these were all related to **shopper density** in the store. We thus derived a formula relating **shopper density, item fragility, number of unsold goods and the discounted price of the item** to obtain the monetary loss of one tile resulting from the damage of items. The sum of monetary loss of all the tiles in the store was taken as the **total loss.**

To simulate the passage of shoppers through the various layouts, we used the a* pathfinding algorithm to construct an agent-based model. The likelihood of a good being the desired good of each shopper was based on the GHI of the good. Each agent, representing an individual shopper, would go from the entrance to the shelf with the desired good, the cashiers, and finally the exit. The various paths were then merged onto one matrix to find the shopper density per tile and thus the total monetary loss.

A modular function was created to test large numbers of different item arrangements to arrive at the optimum item arrangement based on GHI, which was determined to be one with **decreasing GHI from the cashier**. The optimum "prominence" threshold (**0.9**), which indicated the range of popular goods which should be displayed separately from their department to minimise total loss, was also obtained.

Different floor plans were tested using these metrics. The **optimal item arrangement** from above was used in **different layouts** hand-created by the team. Without changing the size and shapes of shelves given, an optimal layout was found. However, given liberty to modify shelves completely, the grid layout format was then determined as the most optimal layout type and variants of it were tested to select the **optimum layout** which minimised total loss.

2 Table of Contents

3 Introduction

Black Friday is an annual tradition that has been held in the US for decades (Pruitt, 2015). Held the day after Thanksgiving, the holiday sees retailers slashing prices on many products in a bid to turn a profit. Consequently, many Americans hit the shops on Black Friday every year, with 135.8 million Americans indicating that they would spend the day shopping in 2015 (Pruitt, 2015). However, this rush for goods has resulted in many accidents, injuring over 100 people from 2006 to 2018 (Crockett, 2019). Not only has this resulted in injuries, but it has also resulted in the damage of many consumer goods, either due to shoppers knocking over goods through carelessness or through consumers fighting over them (Katz, 2019). Due to shoppers rushing to obtain a particular good and scrambling to pay for it so as to save time to buy more goods from other retailers, stampedes may even occur, causing injuries and damage to goods. Thus, Black Friday has caused many producers to suffer losses instead of turning over profits.

Furthermore, in this digital age, more and more consumers are turning to online shops like Amazon instead due to reasons such as convenience. As a result, more brick-and-mortar shops are implementing flash sales in a bid to attract customers to purchase from them instead of from online department stores, increasing the risk of suffering losses due to damaged goods as a result of an uncontrollable influx of consumers.

Therefore, it has become increasingly essential to manage such instances of 'human traffic' so as to ensure that the endangered brick-and-mortar shops can achieve the intended effect of turning a profit rather than suffering losses. The model developed by our team aids this by manipulating store layout and product distribution to redirect shoppers in a manner as to reduce the number of accidents.

Although similar studies have been done in the past where methods like Dijkstra's algorithm and Bellman-Ford's algorithm were used to find the shortest path to obtain goods in a store (Dela Cruz et al., 2016), the model developed by our team uses a novel method as it involves manipulating both the store layout and the customer path, which enhances its effectiveness.

4 Model

4.1 Operationalization of Consumer Preferences

Examining the store data provided, we identified the discount offered, consumer rating, and quantity available as factors likely to affect consumer preferences toward an item.

4.1.1 Discount offered

A consumer's perception of the "value" of a discount is based on two factors: the absolute monetary value of the discount (i.e. price − discounted price) and the percentage discount (i.e. $\frac{\text{price–discounted price}}{\text{price}} \times 100\%$). Intuitively, as the discount amount and percentage discount increases, the consumers' perceived gain from purchasing that good would increase as they save more money from the original price. We hence "double-count"

the discount in our modelling of consumer preferences by taking into account both discount amount and percentage—this is necessary to reflect consumers' tendency to place more significant weightage on the extent of the discount offered than other factors in the context of a flash sale situation. This is corroborated by Fam et al. (2019) who found that 'discounts and coupons are the two most highly ranked SP (sales promotion methods) across the sampled countries' in their study.

4.1.2 Consumer rating

Consumer rating is taken as a baseline measure of product attractiveness, which represents how consumers see an item regardless of whether it is on sale. Accordingly, consumers are more likely to buy a more highly rated item.

4.1.3 Quantity available

Conversely, the lower the quantity of an item offered, the higher its perceived rarity. As consumers are more likely to perceive their purchase as a better deal if they are among the few able to get the item, goods with a lower stock will be more attractive to consumers. This is a valid relationship. In fact, this is why many consumers are willing to queue up for hours in harsh conditions just to purchase limited edition products from US brand Supreme (Clifton, 2016).

4.1.4 Gross Happiness Index (GHI)

Taking into account all three factors above, we defined a Gross Happiness Index (GHI), a measure of the total pleasure or utility derived by consumers from successfully purchasing one unit of a given product. GHI was calculated with the following formula:

$$\text{GHI(item)} = \frac{\text{amount discounted} \times \text{percentage discount} \times \text{consumer rating}}{\text{quantity available}}$$

To verify that our operationalization is reasonable, we examined the GHI of a number of items in the provided dataset. The most popular goods were the 5.3cu ft Slide-In Electric Range, Stainless Steel (GHI = 212), the 30" Combination Double Electric Convection Wall Oven with Built-In Microwave (GHI = 330), and the 24.7cu ft French Door Refrigerator, Black Stainless Steel (GHI = 398).

This index could alternatively be interpreted as the relative desirability of an item—as such, we calculated the probability that someone will aim to buy a specific item as:

$$\text{P(buying a particular item)} = \frac{\text{GHI(of that item)}}{\sum \text{GHI}}$$

4.2 Encoding Schema for Store Layout

4.2.1 Fundamentals

In order to run a simulation of customer behaviour in a store, we had to encode its layout and item positions in a format which can be interacted with programmatically. As the problem statement provides us with a $48\text{m} \times 48\text{m}$ floor plan, we chose to use a 48x48 grid as the environment for our simulation, where each grid unit had dimensions of 1m by 1m.

In order to faithfully recreate the given store layout, we extracted the original image and converted the picture to black and white, removing all annotations and arrows. We then downscaled the image to a 48x48 pixel bitmap (.pbm) and inverted the colours. This process is shown in Figure 1.

With reference to Figure 1, the leftmost image is the original given image of the layout, the centre image is the image in black and white, and the rightmost image is the image converted to a bitmap.

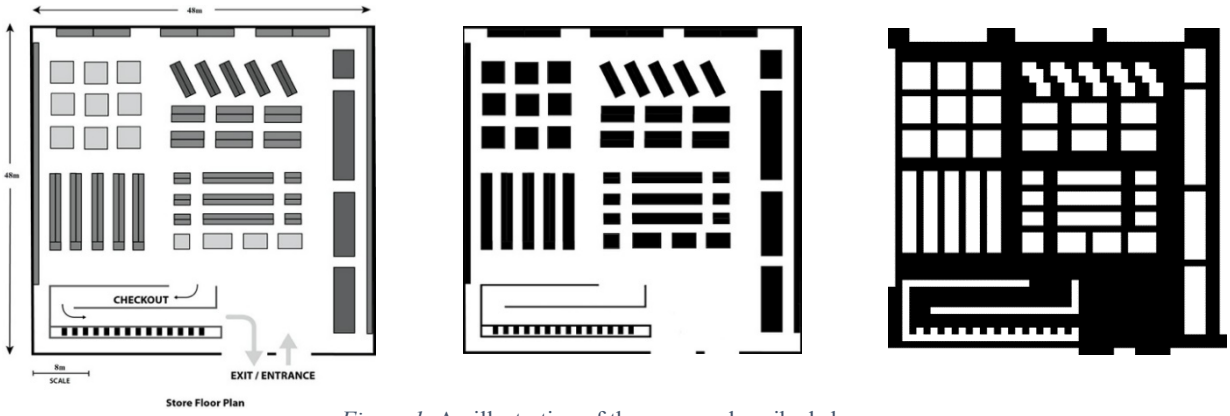

Figure 1: An illustration of the process described above.

4.2.2 Layout components

Hereafter, we refer to each pixel in the bitmap as a *tile*, representing a 1x1m area of the store. 8 different tile types were identified, and each assigned a unique colour. Our final encoding scheme for a store's layout is displayed below:

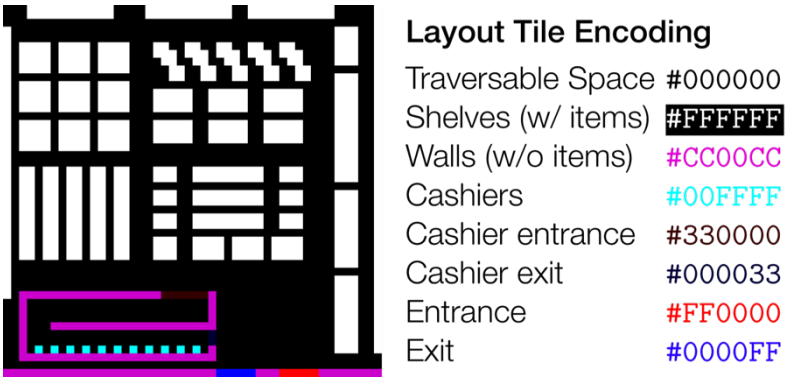

Figure 2: a legend of the tile colours in the bitmap

With reference to Figure 2, *traversable spaces* refer to the black tiles. These are the areas where shoppers can move. They include empty spaces and corridors between shelves. The *shelves* are illustrated as the white tiles where items are placed, and the *walls* are the purple tiles. They are just walls to divide/block shoppers from passing through, and no items will be placed on them. We made this distinction as large items like those listed in the provided spreadsheet (e.g. 85" TVs) are rarely placed along cashier queues or other

inconvenient locations. This also provides us with a utility to manually influence the placement of items—though this was generally performed programmatically, as will be elaborated upon later. *Cashiers* are illustrated as blue tiles, where people make their payments. The *cashier entrance* is illustrated by the maroon tiles, and the *cashier exit* is illustrated by the dark blue tiles. This allows us to optionally indicate where a queue for a cashier will start and end. The rationale for needing to define queues in this manner explicitly will also be elaborated upon further in later parts of the report. Finally, the entrance and exit to the store are the bright red and bright blue tiles respectively.

4.2.3 Data representation

Our custom simulation software imports a bitmap of a store layout via the function `read_img_map()`, separating each set of tiles of a given colour into its own 48x48 matrix, as shown in Figure 3. By primarily dealing with the store's layout as a set of matrices, most computations in our simulation are reduced to matrix operations, maximizing efficiency and minimizing simulation runtime. This is especially important due to the large number of layouts which need to be tested.

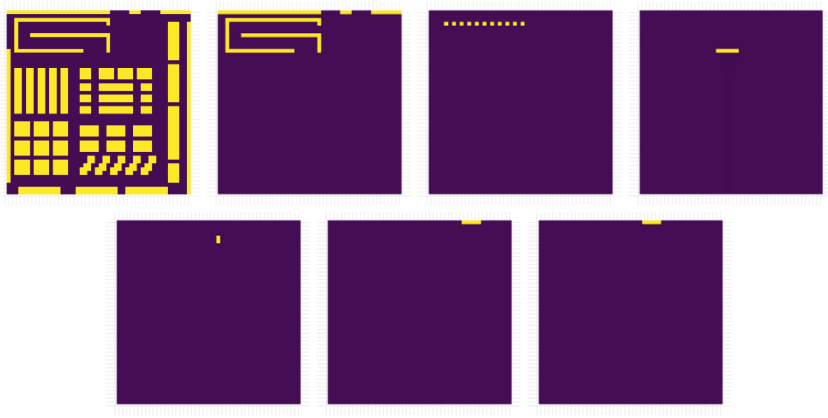

Figure 3: Internal matrices

4.3 Simulating Consumer Behaviour

4.3.1 Behavioural Assumptions

A number of assumptions about shopper behaviour were made, which informed the design of our models and simulation.

4.3.1.1 Assumptions with regard to desired items

Firstly, we assumed that consumers desired only one good and knew what the desired good was before entering the store. This also meant that the consumers did not impulse buy goods other than the originally desired good. This is not an unreasonable assumption, given that considering how crowded retail stores can be during retail sales, consumers will often strategise to target a specific item of interest, and rarely stick around after purchasing their desired good. This was a necessary assumption in simplifying the movement of the agents towards the desired shelf.

Secondly, it was assumed that the shoppers have no prior knowledge of the item positions

in the store. This is a valid assumption, considering that the store's layout is changed just before the flash sale begins. Hence, this meant that the shoppers needed to perform a search through the store to find their desired item. We used a*star search as an approximation of human pathfinding.

Thirdly, we assumed that consumers do not know if the good that they wanted was already out of stock before they entered the store. Even if a particular good is already out of stock, consumers would still make their way towards the particular aisle. This is also a valid assumption—this is the underlying reason why fights occur in the first place, as shoppers try to snatch goods from other people. This was a necessary assumption because it allowed us to simplify the code involved in modelling the behaviour of the agents.

4.3.1.2 Assumptions with regard to other shoppers

As shoppers have physical volume, they are unlikely to fit into the same region of space. This phenomenon is responsible for congestion and jam formation as shoppers try to squeeze past each other while moving through corridors, entrances and exits, drastically slowing movement in those areas. Hence, shoppers would, as far as possible, avoid these regions of congestion so as to get to their desired items quickly. This was a necessary assumption because it improves the accuracy of the model by more realistically simulating the behaviour of shoppers in real life.

4.3.1.3 Assumptions with regard to item damage

One final assumption is that the frequency of accidents and fights between consumers is assumed to follow a power law with respect to the density of people. This is because the more packed a particular walkway is, the more likely that shoppers would accidentally knock over a particular good due to carelessness. Moreover, the more crowded a particular aisle is, the more people there would be fighting over a particular good, hence the higher the frequency of fights occurring. These probabilities would only start increasing in extremely packed situations, because in moderately packed situations the likelihood of these accidents is still reasonably low—for example, if only 10 people walked past an item a day, the probability of an accident is negligible. Without a doubt, the frequency of fights is also dependent on factors like the temperament and personalities of the shoppers. However, it was not possible to take into consideration these subjective factors when designing the model; besides, as the model intends to estimate the average loss, these variables are unlikely to affect the final estimated loss in any case drastically.

4.3.2 Simulation Principles

4.3.2.1 Fundamentals

We adopted an agent-based approach for behavioural simulation, with parameters chosen so that each agent roughly represented 10 real-world shoppers. Each agent followed a simple set of rules based on the aforementioned assumptions of shopper behaviour, as described below:

4.3.2.2 In-store Pathfinding

Each agent was initialized with a stored list of target coordinates to a specified entrance tile, desired item tile, cashier tile, and exit tile. The likelihood of a good being selected as the desired item was based on its GHI, as described above, while specific entrance, cashier, and exit tiles were selected from their respective matrices with equal likelihood.

Agents travelled between these tiles in a set sequence (i.e. entrance→target item→cashier→exit), with routes between the tiles calculated via the a* pathfinding algorithm. Briefly, a* is a graph traversal algorithm which performs a heuristic search for the shortest path between nodes. It saves time by only checking a limited set of adjacent nodes (the "open set") to the current node at each step: for each of these nodes, it estimates the total path cost by summing the distance travelled to that node and the expected remaining distance to the target, as calculated by a distance heuristic (in this case, the Euclidean distance between a node and the target node), then designating the node with the lowest total path cost as the new current node.

We chose a* search as it is a better approximation of a human search for the shortest path (where we are limited by possible paths within our lines of sight) and time-optimality compared to other search algorithms, such as Dijkstra's or Bellman-Ford.

4.3.3 Crowd Avoidance

This process also meant that individual agents did not interact in a shared world every simulation tick; instead, interactions were iteratively computed between agents' complete paths through the shop, making the model "approximate".

To compute these "interactions", a shared density matrix was instantiated, and the drawing of paths commenced. Every time an individual path included a tile, that tile's density was incremented by 1. As previously assumed, people seek out areas of lower pedestrian density while searching for a path through an environment. Therefore, we defined an avoidance coefficient (A) based on the density in a tile. We used this coefficient to increase the distance to tiles with higher densities. For example, the distance to an adjacent tile of 0 density is simply the Euclidean distance (1), but the distance to an adjacent tile of density 1 would be the Euclidean distance + density × A, which would be (1+A). Hence, this coefficient has the effect of artificially increasing the distance travelled in more crowded routes, so that the agents would avoid them and take less crowded paths instead. We used A=0.1 for all our simulations. While the avoidance coefficient was arbitrarily determined, it does have a physical meaning—the value chosen means that an agent density of 10 at a tile would make it twice as difficult to get to that tile.

4.4 Estimating Monetary Loss

4.4.1 Causes

There are various ways consumers may damage goods during flash sales. As crowds of shoppers are pushed along the store in an effort to rush to their purchases, some shoppers may be pushed towards the shelves to knock down small products on the shelves,

damaging them. Larger products placed on the floor may also be knocked down or squashed between crowds to be damaged.

Products may also be damaged during the shoppers' efforts to acquire them. Removing goods from the shelves in the middle of such a crowd would be difficult and may result in the goods being dropped onto the floor. Fights may also occur between shoppers as they try to acquire the limited stock of goods, pulling on the goods and damaging them.

After the shoppers acquire the good, the good may be damaged as they try to wedge out of the crowd, squeezing against the good. Shoppers holding large products are especially prone to this as they may lose balance, injuring themselves and damaging the product.

Furthermore, in a sale situation where desirable goods are in limited quantity, fights may occur due to multiple consumers believing that they have the right to buy a particular unit. This can further cause damage to units being fought over as they may be roughly handled during a scuffle. In addition, if a fight were to break out amongst cramped shelves, the products on the shelves may be knocked off, causing damage.

Evidently, the above cases are more likely to occur as there are more shoppers, represented by a higher human density. As stated above, the frequency of cases would only increase in extremely packed situations. Hence, the function for the amount of damage inflicted is nonlinear with respect to density, with the amount of damage taken to be proportional to the human density squared.

$$\text{Amount of damage inflicted} = \text{density}^2$$

4.4.2 Factors Affecting Loss

Thus, we assumed that the monetary loss for a tile was a function of the amount of damage inflicted in that tile and the properties of items placed nearby the tile.

In all, the estimated loss due to the damage to an item from pedestrian density in a given tile was a function of the amount of damage inflicted, the item's unit monetary value, a decay term taking into account the attractiveness and quantity of the item, and the item's fragility.

The unit monetary value was taken to be the discounted price, as it would be the amount of money received by the store if the good was undamaged. The decay term was taken into account because the quantity of an item remaining will decrease at a higher rate if it is more sought after.

4.4.3 Item Fragility

Fragile goods are more likely to be damaged to a more severe extent, resulting in more significant monetary losses. We defined a fragility index, F_I, to represent the likelihood of an item being damaged in the store. This was calculated using real-life data of the percentage of the product damaged in shipping (Blumberg, 2005).

$$F_I(\text{item}) = \%\text{ of items returned due to shipping damage} \times \%\text{ of all items returned}$$

F_I can be understood as the probability that an item would be damaged as a result of being shipped once. Therefore, we need to relate the amount of damage inflicted due to pedestrian density to the "number of times shipped". We make an educated guess that 1000 people walking within 1 metre of an item on a busy day will cause an equivalent amount of damage to a product being shipped 10 times. We use this guess to define a constant, c, relating the arbitrarily defined amount of damage inflicted to our fragility index and "number of times shipped":

$$\frac{1000^2}{c} = 10$$

$$c = 100000$$

With this constant, we define a density index, D_I, calculated:

$$D_I(\text{tile}) = \frac{\text{density}^2}{100000}$$

4.4.4 Total Expected Loss

The total monetary loss due to density at a particular tile was equal to the summation of monetary loss to items in the adjacent tiles.

To find this monetary loss to items, we used the following formula:

$$\text{Monetary loss to item in adjacent tile} = (\text{Di} \times \text{Fi}) \times \text{discounted price of item} \times \int_0^1 \text{qty of item } dt$$

Where t is the proportion of the day passed.

If items are still remaining after the sale,

$$\int_0^1 \text{qty of item } dt = \text{remaining qty} + \tfrac{1}{2}(\text{initial qty} \times (\text{initial qty} - \text{remaining qty}))$$

Where

$$\text{remaining qty} = \text{item qty} - \text{expected purchased qty}$$

And

$$\text{expected purchased qty} = P(\text{item purchase}) \times \text{number of agents}$$

If items are all bought,

$$\int_0^1 \text{qty of item } dt = \tfrac{1}{2}(\text{initial qty} \times \text{time sold out})$$

Where

$$\text{time sold out} = \frac{\text{initial qty}}{\text{expected purchased qty}}$$

With these operations, we could evaluate the total monetary loss per tile, which could be summed to obtain the estimated monetary loss over the entire store. Hence, we had an operation that maps a matrix of pedestrian density to a matrix of estimated monetary loss.

4.5 Model Evaluation

4.5.1 Validation Against Real-world Behaviour

We found that our model could successfully simulate many observed pedestrian behaviours, including bottlenecking at tight corridors and jamming at intersections. An example with random item layout is shown below, demonstrating these characteristics.

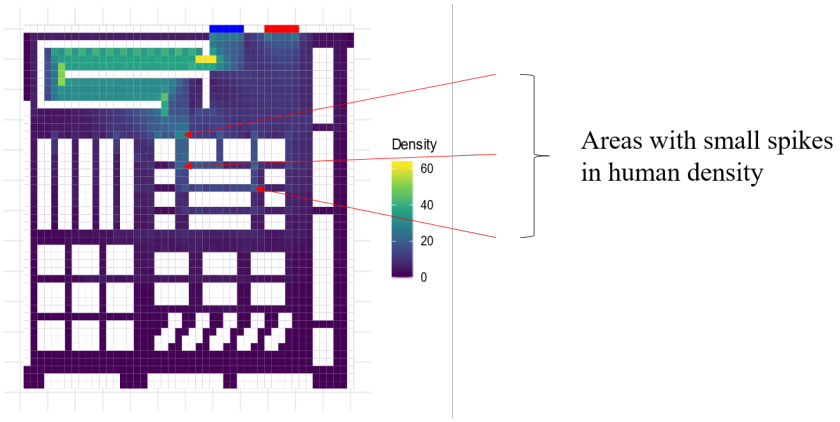

Figure 4: running our simulation on the given layout with randomised item arrangement

With reference to Figure 4, the legend indicates that dark blue areas have the lowest agent density, followed by green areas and yellow areas. The green colour of the walkways along the cashiers shows the high density of agents in the narrow corridor, especially at the even narrower exit. Smaller spikes in human density can also be seen in the intersections of the walkways between shelves, as labelled by the red arrows in Figure 4.

The simulation also produced results that matched intuitions about item positioning. As stated above, if popular items are placed near the entrance, exit, and cashier, the paths to obtain these goods would be shorter, resulting in the decrease of total population density from these paths and thus decreasing total loss. This is evidenced by Figure 5. The simulation was run twice: once for a layout where high GHI goods are placed near the entrance, exit and cashiers, and another time for a layout where low GHI goods are placed near the entrance, exit and cashiers. The results of the simulations are shown in Figure 5.

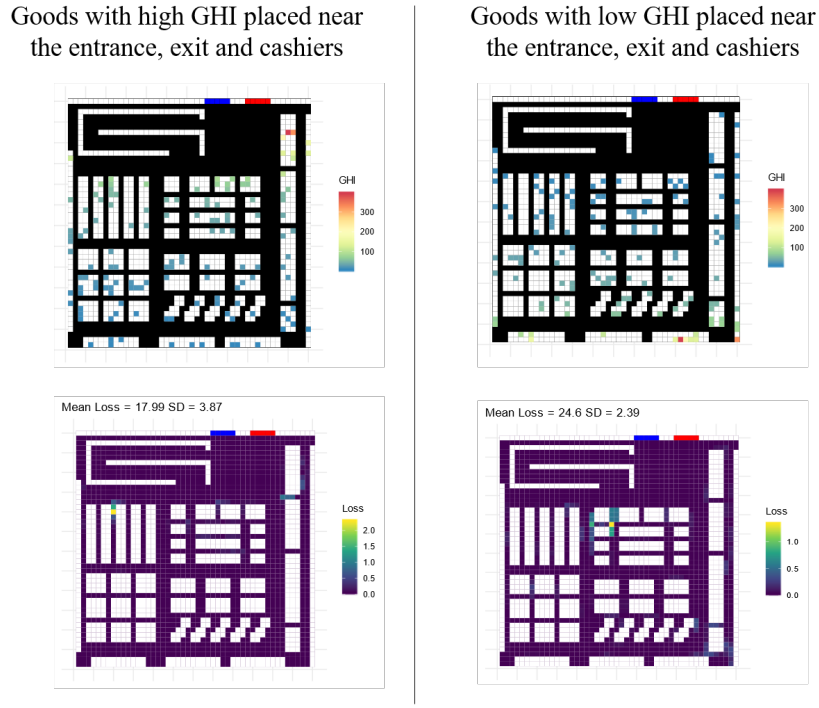

Figure 5: a comparison of simulations run for two different arrangements of goods.

The left side of Figure 5 is the simulation run for the layout where goods with high GHI are placed close to the entrance, exit and cashiers, while the right side is the same results for the low GHI layout. The top half shows the arrangement of the goods with different GHIs, and the bottom half shows the monetary loss. As can be seen, the top left image shows that high GHI items (red) are placed close to the entrance, followed by low GHI items (blue) placed further away. The opposite is true for the top right figure. Consequently, the bottom left image shows that the average monetary loss for the layout where high GHI goods are placed close to the entrance is 17.99, and the average monetary loss for the low GHI layout is 24.6, which is considerably higher.

Additionally, as we were trying to calculate a cumulative loss value, ultimately, the vast majority of time-varying behaviour did not benefit analysis and was computationally wasteful to simulate. For example, a simulation encompassing oscillation of passing direction at a bottleneck will average out to a diffuse region of high density around the bottleneck, producing the same result as our model. The model hence essentially simplified this process to save on computational power.

4.5.2 Robustness

However, since the simulation is iterative, results technically depend on agent order, though this effect likely diminishes with larger agent numbers. Furthermore, the item positions on each shelf were also randomized for each replicate, introducing further variance to the results of the model. This made the model stochastic as the results were heavily influences by random factors.

To account for this effect, we ran 10 replicates of each simulation each with randomized agent orders and item positions. This would have normalized the variations of the results stemming from the agent orders. Fortunately, this barely hindered computation, as the operation is fully parallelizable across multiple CPU cores. On top of this, we computed the standard deviation for the 10 replicates, allowing the comparison of preciseness between layouts and their corresponding item arrangements. The stochasticity was thus overcome to some degree, making the model robust.

4.5.3 Limitations

Despite the strengths of our model, every model has its limitations.

Firstly, the fragility index formulated by our team may not be completely accurate. This is because the fragility index was calculated based on shipping data, provided by Blumberg (2005). The data provided statistics about the average percentage of a particular product type getting damaged in shipping. For instance, according to the book, on average, 2.36% of all desktop computers get damaged in shipping. This percentage was used as an indication of the fragility of the good since goods which are more likely to get damaged during shipping are likely more fragile. However, this data has two main shortcomings. Firstly, according to the book, on average, 2.232% of all televisions get damaged during shipping. However, this percentage does not make a distinction between different types and sizes of televisions, while the store inventory provided includes televisions with sizes ranging from 30 inches to 85 inches. Since the data provided does not make a distinction between them, they were all assumed to have the same fragility index. This is not entirely accurate, since size may play a part in affecting the fragility of the item.

Another limitation of the data is that it does not include the statistics for certain goods. For instance, the average percentage of goods getting damaged in shipping was not provided for robot vacuums and gaming consoles, both of which appear in the store inventory. However, the data does provide such statistics for all consumer electronics and major appliances on average. Major appliances referred to necessities like refrigerators and washers, while consumer electronics generally referred to entertainment products like laptops and televisions. Hence, for items on the store inventory whose data was not provided, average data for all major appliances/consumer electronics had to be used, depending on the particular good. This could have resulted in inaccuracy since the actual average percentage of robot vacuums (for example) getting damaged in shipping may be different from the average percentage of all appliances getting damaged in shipping.

These two limitations mentioned above can be mitigated by referring to a variety of sources in determining the fragility index. However, due to the short period available in coming up with the improved layout, there was insufficient time to carry out a meta-review of real-life statistics.

Another limitation arises due to the inability of the layout/model in taking into account subjective factors. As mentioned above, the likelihood of damage also depends on the personalities of the customers, as short-tempered consumers are more likely to get into fights/arguments, which may lead to damaged goods. However, such subjective emotions cannot be modelled via mathematical means; thus, this is an inevitable limitation.

One other limitation of the layout/model is that it does not take into account the different brands of the goods on sale. According to the store inventory provided, the items on sale are manufactured by different brands like Brand FF, Brand M, Brand W etc. The branding will likely have a significant impact on the gross happiness index (GHI) because consumers will likely derive more pleasure when they purchase a brand that is more famous and well known, which, as mentioned above, is why many consumers go after Supreme products (Clifton, 2016).However, the different types of branding were not taken into account in formulating the GHI, because of the simple reason that based on the brand names provided, the more popular brands could not be distinguished from the less popular brands, and the significance of branding could not be factored into the formula for the GHI.

5 Model-guided Design

5.1 Approaches

5.1.1 Computational

Using software to randomise the layout of the shelves was considered. However, doing so was determined to be extremely time-consuming. This was because the total number of shelves and checkout stations was 53. Even accounting for shelves with the same dimensions, the total number of ways to permute their physical locations in space was still an extremely large number (order of magnitude ~40). Furthermore, other than the location of the different shelves, other factors also came into play like whether the shelves are positioned horizontally, vertically or at various angles, further increasing the total number of ways to arrange them. Machine learning approaches—genetic algorithms and simulated annealing among them—were considered, but ultimately abandoned due to massive size of the search space.

In the end, a modular function was created to allow for different organisational methods to be used for item placement within the space and item organisation based on GHI and fragility. The modularity of this function allowed for the addition differ

5.1.2 Human-guided

Thus, our team decided to manually design layouts which we deemed suitable for a flash sale, iterating upon them based on simulation results. This was possible as we were able to edit the layouts directly with an image editor rather than editing every value of the matrix, allowing for rapid creations of layouts.

We researched a list of commonly used store layouts, such as the forced-path layout, the grid layout, the geometric layout etc. (“Retail Store Layout Design and Planning”, n.d.).

Some of these layouts were deemed to be not optimal for flash sales, as seen below. We decided that the grid layout was the best for flash sales and designed floor plans around this layout. The matrix model mentioned was then used to test which layout allowed for the lowest likelihood and extent of damage.

The various layouts available for design are as follows:

Forced-path layouts are the layouts widely used by Dutch furniture retail giant IKEA where consumers are forced to move through the entire store. Such a layout is not optimal for the flash sale since consumers generally have something they desire and would not want to waste time moving around the entire store. An example of a forced-path layout is shown in Figure 6.

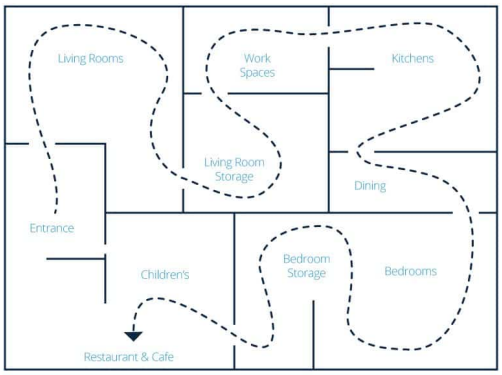

Figure 6: An example of a forced-path layout. Retrieved from "Retail Store Layout Design and Planning", n.d.

Similarly, loop layouts, which force customers to go one entire loop around the store, are also not feasible for the same reason.

Diagonal layouts where the shelves are arranged diagonally are also not optimal for flash sales, because arranging the shelves diagonally would result in narrower aisles which lead to an increased likelihood of damage. The given layout does include a row of diagonally arranged shelves. An example of such a layout is shown in Figure 7.

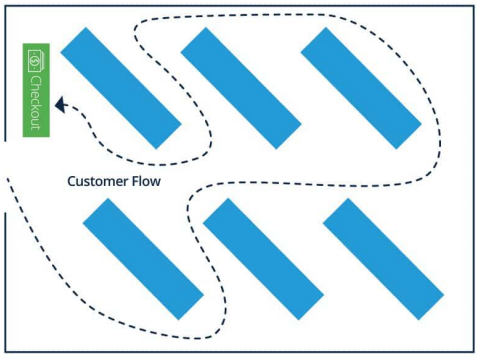

Figure 7: An example of a diagonal layout. Retrieved from "Retail Store Layout Design and Planning", n.d.

Grid layouts were deemed to be optimal for flash sales because it allows for more efficient movement of customers and greater distance between shelves allowing for reduced likelihood of damage. The only downside of the grid layout is the lack of visual and aesthetic appeal. However, this is generally not a consideration during a flash sale. An example of a grid layout is shown in Figure 8.

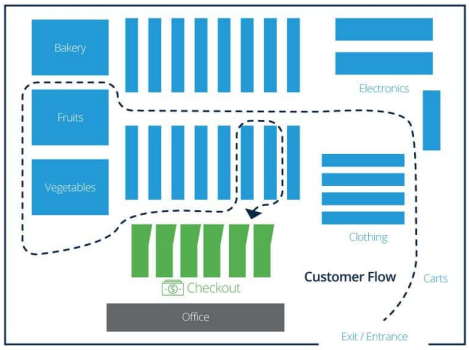

Figure 8: An example of a grid layout. Retrieved from "Retail Store Layout Design and Planning", n.d.

Various grid layouts were designed based on the principle that the most popular items with the highest GHI should have the greatest spacing around them, so as to accommodate higher numbers of agents (shoppers) and hence minimize human density and thus item damage.

5.2 Experiments (Note: in each experiment, the dependent variable was the monetary loss)

5.2.1 Independent Variable: Item Placement in Original Layout

In this section, the variable tested was the different methods of arranging items given their different GHIs on the default shelves provided.

Our baseline was a random initialisation of the shelf and item positions on the original layout, which resulted in a loss of 17.83. Visualisations of the layout generated, pedestrian density, and loss are provided in Figure 9.

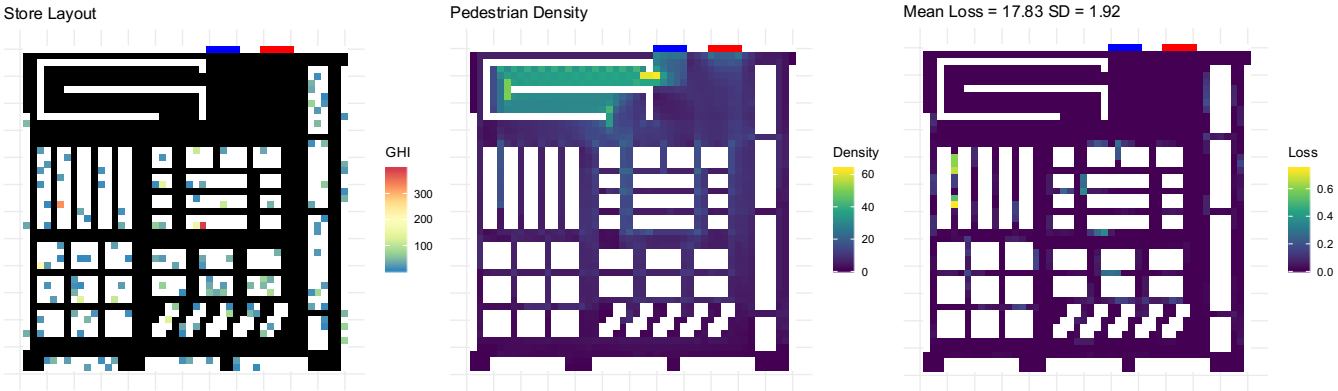

Figure 9: Maps of random initialisation of item arrangement

Using the mathematical model, the shelves of products were rearranged to yield the lowest total loss. We refined this arrangement further to place the items with higher GHI closer to the cashier (in terms of Euclidean distance), and items with lowest GHI further away from the cashier. This further lowered the monetary loss to 11.69.

Placing the high GHI goods closer to the cashier allowed the minimisation of total loss as the paths taken by the shoppers, which involved moving from the good to the cashiers, decreased in length. Thus, the total density contributed by these paths decreased, leading to the decrease in total density in the store and hence the decrease in total loss, which is visible when comparing Figure 10 to Figure 9.

Up to this point, item placement has not taken into consideration product category, as this factor is not taken into account in the agent pathfinding—that is, agents will find a path to an item just as easily, whether or not they are sorted into departments or not. Of course, actual shoppers would find it much easier to search for items according to category. Therefore, any practical store layout would only place a limited number of high GHI items in individual shelves, while all other items would be sorted by department for easier access.

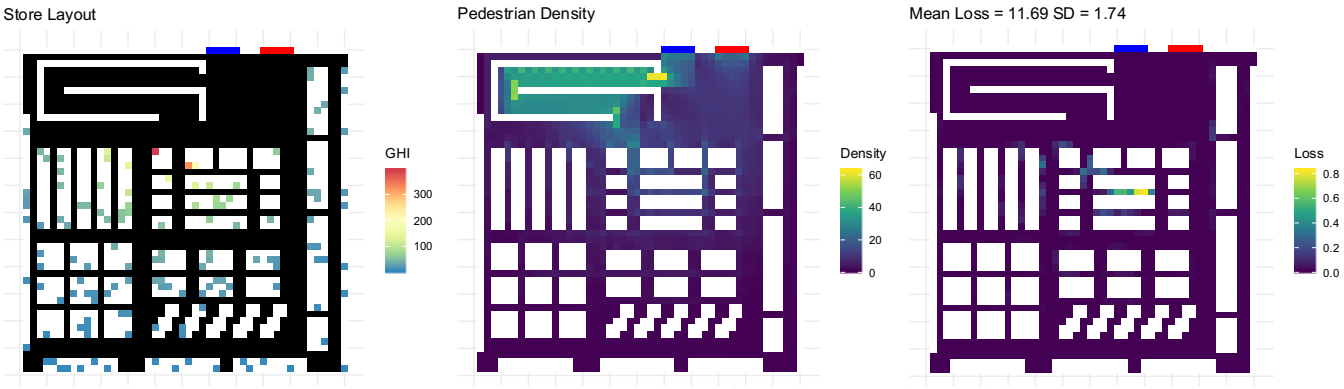

Figure 10: Maps of item arrangement of decreasing GHI from cashier

In our code, we accounted for this factor by determining a prominence threshold, P, representing the percentile of GHI above which we consider an item popular enough to be assigned its own display. First, items above that threshold were placed in positions closer to the cashier, in terms of decreasing GHI, regardless of category. Items were then assigned positions according to their category, with their distance from the cashier determined by the category's average GHI. Positions of items within their own category area were further sorted by item GHI. An illustration of this sorting procedure is shown below:

Prominence thresholds ranging from 0 to 0.9 were tested, with 0.9 being the optimal "prominence threshold" value, resulting in the lowest loss value of 11.2. A graph of loss against threshold is displayed in Figure 11.

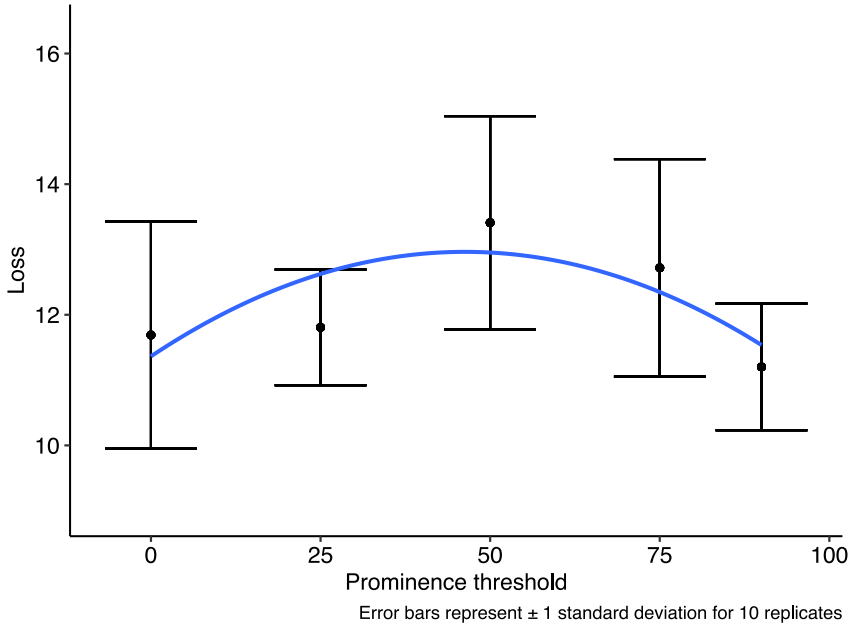

Figure 11: Prominence against loss

As seen in Figure 11, although a trend is observed, the large error bars indicate that it may not be significant. However, this is ideal as it indicates that sorting by department does not greatly affect the loss, allowing for easier identification by customers when they visit the store.

5.2.2 New possible layouts

5.2.2.1 Independent Variable: arrangement of shelves

Many designs were made by rearranging the given shelves (without changing the dimensions), and testing was then done on the layouts. The algorithm for item arrangement followed that from section 4.2.1, with a prominence threshold value of 0.9 and an arrangement of decreasing GHI from the cashier. 5 generated layouts and their associated loss values are presented below in Figure 12:

Figure 12: Maps of possible arrangements of shelves

The pedestrian heat-map of the final layout can be found in Figure 12. A spike in pedestrian density around the corner of the walls of the walkway and funnelling through the walkway can be seen.

A mean loss of 9.71 and a standard deviation of 0.74 was recorded as seen in the last map of Figure 13, the bottommost map, significantly lower than the mean loss of 11.69 of the original layout as shown in Figure 10.

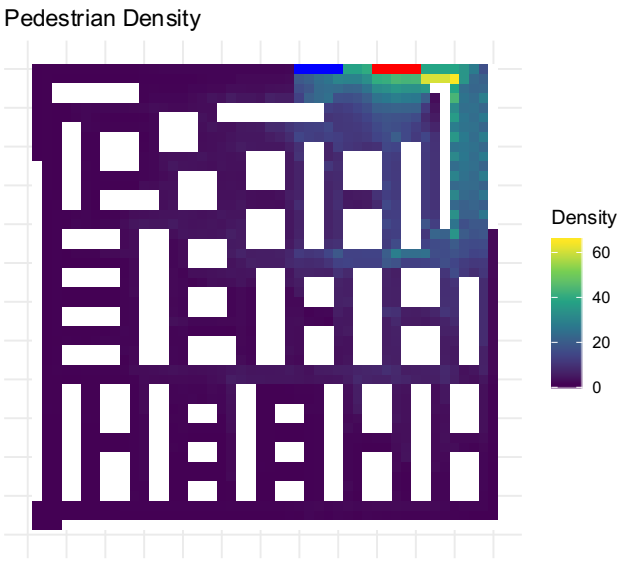

Figure 13: Pedestrian density heat-map of final layout

With reference to Figure 10, for the same item arrangement on the given layout, the monetary loss was 11.69. However, as can be seen in Figure 12, the monetary loss for the chosen final layout was only 9.71, which is objectively a lot lower than that of the given layout. This can be explained with various factors. Firstly, in the given layout, there is very little space in between each shelf, with most of the corridor widths being as narrow as 1 metre. In contrast, with reference to Figure 12, there is at the very least a 2 metre spacing between each shelf, with some spacings being as wide as 3 or 4 metres. This helps to greatly reduce the human density around the items, which is especially important for the more fragile items. In addition, another main difference lies in the fact that in the original layout, there are not many paths which consumers can take. For instance, when consumers are trapped in between two long shelves, they only have two available paths, which results in a high human density. In contrast, with reference to Figure 12, as far as possible, the long shelves were alternated and interspersed with shorter shelves. This is so as to provide shoppers (agents) with as many available routes as possible so that they can disperse themselves (according to the avoidance coefficient as mentioned above), allowing for minimized human density around the items. These are two main reasons why our final layout proves superior to the given layout.

5.2.2.2 Independent variable: Modifying the shelves (changing their dimensions)

The following experiments / tests assume that the dimensions of the shelves can be changed and modified accordingly. A forced-path layout was attempted, shown below in Figure 14. However, this led to a high loss value of 51.49 as all customers were forced

down the same route, increasing density.

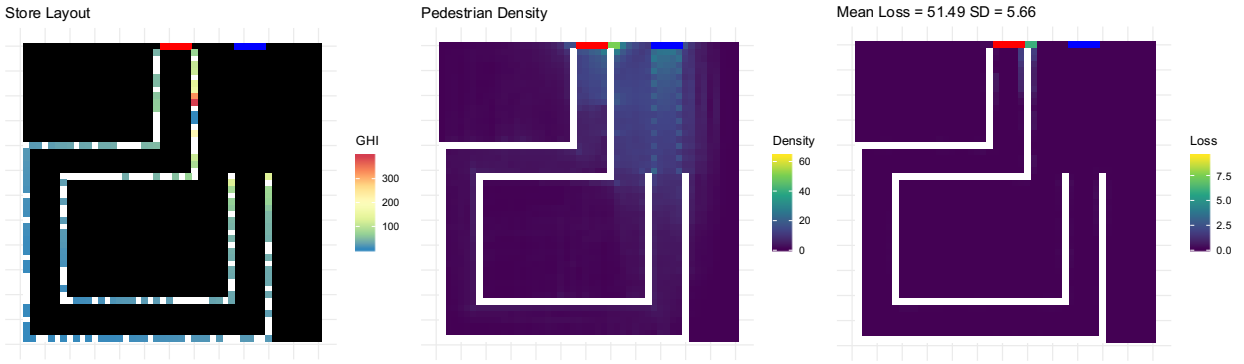

Figure 14: Forced-path layout

As such, this was improved by adding shortcuts between sections of the forced path without any goods, allowing customers with goods nearer the end of the forced path to avoid taking the main path, alleviating some of the density there. The simulation result is as shown in This significantly reduced the loss value from 51.49 to 29.59. However, this was still higher than the default. From this, it was theorised that a larger open space would allow for lower loss value. Hence, the forced-path layout was confirmed to be unsuitable and a more open concept was developed.

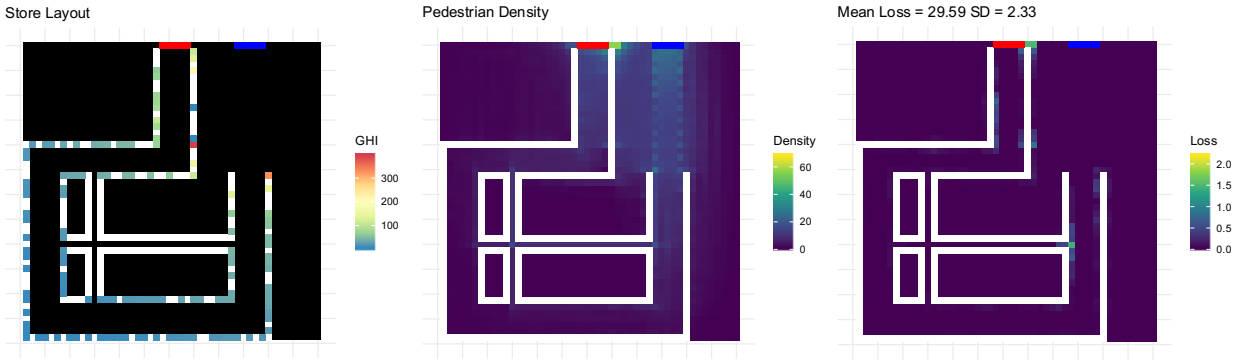

Figure 15*: forced path layout with shortcut*

The final experiment involved modifying the dimensions of the shelves to the point where each shelf only housed one item. Doing so allowed for the maximum space between shelves. The layout and arrangement of items on the shelves, the pedestrian density, as well as the loss value can be found in Figure 16

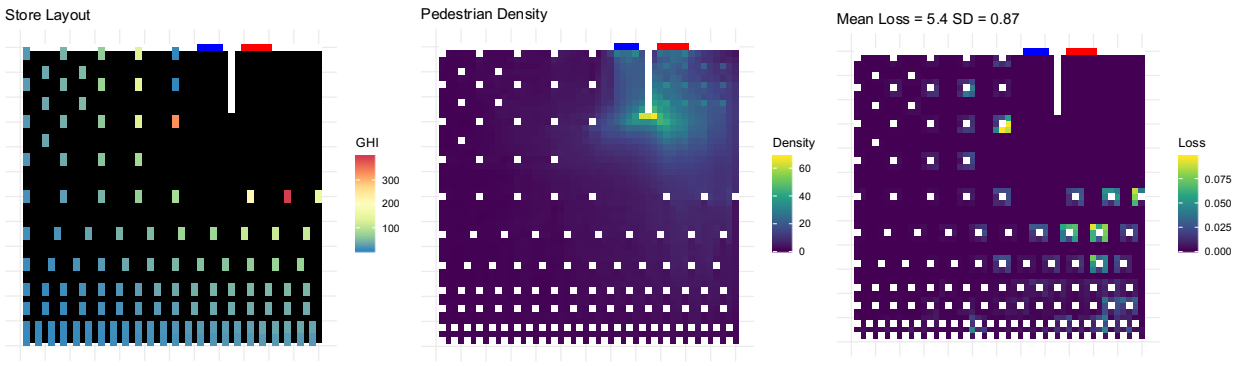

Figure 16: item arrangements, pedestrian density, and loss value diagrams for the 'open' layout.

With reference to Figure 16, it can be seen that for this layout, the shelves and items were arranged such that the items with high GHI were placed on shelves that were spread far apart, and items with low GHI were placed on shelves that were placed closer together. Thus, it prioritises minimizing human density around the shelves containing the high GHI goods. This strategy is shown to be highly effective, as evidenced by the pedestrian density simulation result, where most of the bitmap is dark blue in colour (low density). This is also evidenced by the mean loss result, where the mean monetary loss is only 5.4, the lowest result. Further iterations on this design were attempted, by selectively moving shelves with the highest loss, but no further improvements were achieved, indicating that this arrangement is close to the optimum possible.

6 Conclusions

The optimum item arrangement keeping the given layout is shown in Figure 5 (left side), with a loss of 17.99.

These shelves can be shifted around (while keeping the dimensions) to form the optimum layout with a loss of 9.71, as shown in Figure 12.

However, assuming modifying the dimensions of the shelves is allowed, it is possible to design a layout with even lower total loss of 5.4, as shown in Figure 16.

6.1 Recommendations

In this model, layouts were manually designed and tested using the simulation program. This was because testing each layout required considerable amounts of time; hence it was not feasible to use a randomizer program to come up with every possible layout combination and test every single one of them due to the sheer number of layout combinations. In future studies, given sufficient time and software resources, this approach can be made to make sure that the entire exhaustive list of possible layouts is tested, to come up with a definite optimum layout.

Additionally, in future studies, with more time resources, a larger number of studies can be consolidated to obtain more comprehensive statistics, such as the % of a particular

good getting damaged in shipping. This would allow for a more accurate determination and refining of the fragility index.

As mentioned above, due to the lack of reliable statistics, we also had to come up with a guess that 1000 people walking within 1 metre of an item on a busy day would cause the same amount of damage to the product as the product being shipped 10 times. In future studies, comprehensive analysis can be done on real-life customer behaviour, to more accurately determine how much damage would be done by 1000 customers walking within 1 metre of an item in 1 day. This was not done in this paper due to a lack of reliable sources. In addition, in this paper, we assumed that the agents, like shoppers in real-life, would tend to avoid crowded routes and use less-crowded routes instead to get to their destination faster. However, the extent to which shoppers in real-life exhibit this behaviour is unknown. In other words, in real-life, the precise 'threshold' of human density at which shoppers begin to utilize alternative routes is not known. Hence, the avoidance coefficient developed by our team might have been too large or too small. In future studies, more comprehensive research on shopper behaviour can be conducted to come up with a more accurate avoidance coefficient.

A similar issue was faced when coming up with the relationship between item damage and human density. As mentioned above, in this report, item damage was taken to be proportional to the square of the human density. Although it is a reasonable assumption that the amount of damage has a nonlinear relationship with density, the exact relationship is unknown. More analysis of consumer behaviour can be done in future studies to ensure the determination of a more accurate relationship. Nevertheless, the relationship between the amount of damage and human density can never be accurately determined, because as mentioned above, this is also influenced by subjective factors like the emotions and personalities of the shoppers.

7 References

Blumberg, D. F. (2005). Introduction to management of reverse logistics and closed loop supply chain processes. Boca Raton: CRC Press.

Clifton, J. (2016, July 19). Why Are So Many People Obsessed with Supreme? Retrieved March 20, 2020, from https://www.vice.com/en_us/article/5gq393/supreme-and-the-psychology-of-brand-devotion

Crockett, Z. (2019, November 23). The tragic data behind Black Friday deaths. Retrieved March 16, from https://thehustle.co/black-friday-deaths-injuries-data/

Dela Cruz, J. C., Magwili, G. V., Mundo, J. P. E., Gregorio, G. P. B., Lamoca M. L. L., & Villasenor J. A. (2016). Items-mapping and route optimisation in a grocery store using Dijkstra's, Bellman-Ford and Floyd-Warshall Algorithms. 2016 IEEE Region 10 Conference (TENCON), 2016. doi: 10.1109/TENCON.2016.7847998

Fam, K. S., Brito, P. Q., Gadekar, M., Richard, J. E., Jargal, U., & Liu, W. (2019). Consumer attitude towards sales promotion techniques: a multi-country study. *Asia Pacific Journal of Marketing and Logistics*, *31*(2), 437–463. doi: 10.1108/apjml-01-2018-0005

Katz, L. (2019, November 29). Black Friday 2019 fights prove shopping for deals is as perilous as ever. Retrieved March 20, 2020, from https://www.cnet.com/news/black-friday-2019-fights-prove-deal-hunting-is-as-perilous-as-ever/

Pruitt, S. (2015, November 23). What's the Real History of Black Friday? Retrieved March 16, 2020, from https://www.history.com/news/whats-the-real-history-of-black-friday

Retail Store Layout Design and Planning. (n.d.). Retrieved March 19, 2020, from https://www.smartsheet.com/store-layout

8 Letter to Store Manager

Dear Sir / Mdm,

Thank you for your continuous support to our team. Attached is our proposed layout for the upcoming 2020 Flash Sale.

After doing extensive research on the types of store layouts, we determined that the grid layout is the best choice for a flash sale. We determined that the factors affecting the popularity of a good are the customer rating, the discount amount, as well as the quantity of the good available. We took these into account to calculate the Gross Happiness Index (GHI), which measures the popularity of the good and hence the likelihood of the good being purchased by shoppers.

Armed with the above knowledge, our team tirelessly designed many grid layouts for the store. In designing these layouts, we followed various guidelines which we believed would lead to the lowest amount of damage to the items. For instance, shelves were arranged as spaciously as possible to minimize human density; we placed the checkout area close to the exit to ensure that the customers who have paid for their goods would leave efficiently and not obstruct those who have not purchased their goods; and finally, based on the GHI of the goods, we arranged the goods in such a way that would lead to the least overall monetary loss.

We then built a computer program to test the layouts that we had designed. The program simulated the different routes that the shoppers would be able to take for the different layouts. The agents in the model simulate the behaviour of shoppers in real-life, such as how shoppers would avoid more crowded paths and utilize longer, but less crowded paths to reach their destination more quickly. We also managed to obtain data from reliable published sources to calculate the fragility of the different goods on sale. Thus, the final layout you see is the one that has come out on top through all the rounds of rigorous testing. This layout allows for the lowest human density around the most fragile and expensive items, reducing the overall expected monetary loss.

However, do note that our floor plan cannot prevent fights between shoppers, as this depends entirely on the personalities of the shoppers. Hence, please consider hiring security guards and installing security cameras or warning signs in the store to deter fights between shoppers.

We wish you the best of luck with your event.

Most Sincerely, The IMMC team

9 Appendix

9.1 Code

9.1.1 Release of Code & Data

9.1.2 Image Processing

```
read_img_map <- function(img_path, targets_encoded=FALSE) {
  img <- read.pnm(img_path)

  #ffffff = Walls (all non-traversable tiles, including shelves and walls)
  green_minus_target = img@green
  green_minus_target[which(green_minus_target != 0)] <- 1
  walls_mat <- img@red + green_minus_target + img@blue
  walls_mat[which(walls_mat != 3)] <- 0
  walls_mat <- walls_mat/3

  #cc00cc = Blocked walls (walls where items cannot be placed)
  blocked_mat <- img@red + img@blue + img@green
  blocked_mat[which(blocked_mat!=1.6)]<-0
  blocked_mat <- blocked_mat / 1.6

  walls_mat <- blocked_mat+walls_mat

  #00ffff = Cashiers (for now, the second target for all agents)
  cashier_mat <- img@green + img@blue + img@red
  cashier_mat[which(cashier_mat != 2)] <- 0
  cashier_mat <- cashier_mat/2

  #330000 = Cashier inlet (for supporting queueing behaviour)
  cashier_in_mat <- img@red
  cashier_in_mat[which(cashier_in_mat!=0.2)] <- 0
  cashier_in_mat <- cashier_in_mat * 5

  #000033 = Cashier outlet (for supporting queueing behaviour)
  cashier_out_mat <- img@blue
  cashier_out_mat[which(cashier_out_mat!=0.2)] <- 0
  cashier_out_mat <- cashier_out_mat * 5

  #ff0000 = Entrances (possible source points)
  entrance_mat <- img@red
  entrance_mat[which(entrance_mat!=1)] <- 0
  entrance_mat = entrance_mat - walls_mat + blocked_mat

  #0000ff = Exits (final target for all agents)
  exit_mat <- img@blue
  exit_mat[which(exit_mat!=1)] <- 0
  exit_mat = exit_mat - walls_mat - cashier_mat + blocked_mat

  #00xx00 = Target objects (for now, the first target for all agents)
  # convert back to 0-255 encoding
  if(targets_encoded == TRUE) {
    target_mat <- img@green*255
    target_mat[which(target_mat == 255)] <- 0
    target_df <- make_df_full(target_mat)
    target_df <- target_df[which(target_df$value != 0),]
    target_df$value = 255 - target_df$value
    target_df <- target_df[order(target_df$value),]
    target_df$ghi <- storedata$ghi
  } else {
    target_df <- NULL
  }

  store_layout <- list("walls_mat" = walls_mat, "blocked_mat" = blocked_mat,
"cashier_mat" = cashier_mat, "cashier_in_mat" = cashier_in_mat, "cashier_out_mat" =
cashier_out_mat, "entrance_mat" = entrance_mat, "exit_mat" = exit_mat, "target_df" =
target_df)
```

```
  return(store_layout)
}

create_agent_list <- function(store_layout, n_agents) {
  walls_df <- make_df(store_layout[["walls_mat"]], 1)
  cashier_df <- make_df(store_layout[["cashier_mat"]], 1)
  entrance_df <- make_df(store_layout[["entrance_mat"]], 1)
  exit_df <- make_df(store_layout[["exit_mat"]], 1)
  target_df <- store_layout[["target_df"]]

  # we sample with replacement for each step to generate dfs of targets for all agents
  entrance_tiles <- entrance_df[sample(nrow(entrance_df), size=n_agents, replace =
TRUE),]
  target_tiles <- target_df[sample(target_df$value, prob=target_df$ghi, replace=TRUE,
size=n_agents),]
  cashier_tiles <- cashier_df[sample(nrow(cashier_df), size=n_agents, replace =
TRUE),]
  exit_tiles <- exit_df[sample(nrow(exit_df), size=n_agents, replace = TRUE),]

  agent_list = list()

  for(i in 1:n_agents) {
    entrance_coords = as.numeric(entrance_tiles[i,])
    target_coords = as.numeric(target_tiles[i,1:2])
    cashier_coords = as.numeric(cashier_tiles[i,])
    exit_coords = as.numeric(exit_tiles[i,])
    agent_list[[i]] = list(entrance_coords, target_coords, cashier_coords,
exit_coords)
  }
  return(agent_list)
}
```

9.1.3 Core Simulation Routine

```
library(dplyr)
library(grid)
library(reshape2)
library(pixmap)
library(ggplot2)
library(plotly)
source("astar.R", local=TRUE)
source("searchmaze.R", local=TRUE)
source("storedata.R", local=TRUE)

reverse_encode <- function(store_layout, img_w=48, img_h=48) {
  walls_mat <- store_layout$walls_mat
  entrance_mat <- store_layout$entrance_mat
  exit_mat <- store_layout$exit_mat
  target_df <- store_layout$target_df
  target_mat <- matrix(0, img_w, img_h)
  for(row in 1:nrow(target_df)) {
    coord = as.numeric(target_df[row, 1:3])
    target_mat[coord[1], coord[2]] <- coord[3]
  }
  img_red <- walls_mat + entrance_mat
  img_green <- ((walls_mat + cashier_mat)*255 - target_mat)/255
  img_blue <- walls_mat + cashier_mat + exit_mat

  img_array <- array(c(img_red, img_green, img_blue), dim=c(48,48,3))
  img <- pixmapRGB(img_array)
  return(img)
}

plot_mat <- function(mat) {
  df = make_df_full(mat)
  p<-ggplot(df, aes(x=x,y=y,fill=value)) +
    geom_tile() +
    scale_y_continuous(breaks = seq(0, 48, 1), limits = c(0, 48.5), minor_breaks =
NULL) +
    scale_x_continuous(breaks = seq(0, 48, 1), limits = c(0, 48.5), minor_breaks =
NULL) +
```

```
    coord_equal() +
    scale_fill_viridis_c() +
    theme_minimal() +
    theme(axis.title = element_blank(),
          axis.text = element_blank(),
          legend.position = "none")
  p<-ggplotly(p)
  return(p)
}

plot_df <- function(df) {
  p<-ggplot(df, aes(x=x,y=y,fill=value)) +
    geom_tile() +
    scale_y_continuous(breaks = seq(0, 48, 1), limits = c(0, 48.5), minor_breaks =
NULL) +
    scale_x_continuous(breaks = seq(0, 48, 1), limits = c(0, 48.5), minor_breaks =
NULL) +
    coord_equal() +
    scale_fill_viridis_c() +
    theme_minimal() +
    theme(axis.title = element_blank(),
          axis.text = element_blank(),
          legend.position = "none")
  p<-ggplotly(p)
  return(p)
}

plot_df_ghi <- function(df) {
  p<-ggplot(df, aes(x=x,y=y,fill=ghi)) +
    geom_tile() +
    scale_y_continuous(breaks = seq(0, 48, 1), limits = c(0, 48.5), minor_breaks =
NULL) +
    scale_x_continuous(breaks = seq(0, 48, 1), limits = c(0, 48.5), minor_breaks =
NULL) +
    coord_equal() +
    scale_fill_viridis_c() +
    theme_minimal() +
    theme(axis.title = element_blank(),
          axis.text = element_blank(),
          legend.position = "none")
  p<-ggplotly(p)
  return(p)
}

make_df <- function(mat, val) {
  df <- which(mat == val, arr.ind = TRUE) %>%
    as.data.frame() %>%
    transmute(y = row, x = col)
  return(df)
}

make_df_full <- function(mat) {
  df <- setNames(melt(mat), c('y', 'x', 'value'))
}

make_mat <- function(df, val_col, img_w=48, img_h=48) {
  mat <- matrix(0, img_w, img_h)
  for(i in 1:nrow(df)) {
    row = df[i,]
    coord = as.numeric(row[1:2])
    value = row[[val_col]]
    mat[coord[1], coord[2]] <- value
  }
  return(mat)
}


simulate_density <- function(store_layout, agent_list, coeff=0.1, plot=FALSE,
name="density_plot", img_w=48, img_h=48) {
  density_mat = matrix(0, img_w, img_h)
```

```
  walls_mat = store_layout[["walls_mat"]]
  cashier_in_mat = store_layout[["cashier_in_mat"]]
  cashier_out_mat = store_layout[["cashier_out_mat"]]

  walls_cashier_in = walls_mat + cashier_out_mat
  walls_cashier_out = walls_mat + cashier_in_mat

  for(i in 1:length(agent_list)) {
    current_agent <- agent_list[[i]]
    # Route btw first two targets (entrance to target 1)
    n_routes <- length(current_agent) - 1
    for(j in 1:n_routes) {
      source = current_agent[[j]]
      target = current_agent[[j+1]]
      if(j == 2) {mg <- SearchMaze2D$new(walls_cashier_in, density_mat, coeff)} else
if(j == 3) {mg <- SearchMaze2D$new(walls_cashier_out, density_mat, coeff)} else {mg <-
SearchMaze2D$new(walls_mat, density_mat, coeff)}
      current_path <- mg$run(source, target)
      if (is.null(current_path)) {
        current_path <- mg$run(target, source)
      }
      for(k in current_path) {
        density_mat[k[1], k[2]] <- density_mat[k[1], k[2]] + 1
      }
      if(plot==TRUE) {
        file_name = paste(name," ", i, " ", j, ".png", sep="")
        density_df = make_df_full(density_mat)
        walls_df = make_df(walls_mat, 1)
        p <- ggplot(density_df, aes(x=x,y=y,fill=value)) +
          geom_tile() +
          geom_tile(data=walls_df, fill="#FFFFFF") +
          scale_y_continuous(breaks = seq(0, 48, 1), limits = c(0, 48.5), minor_breaks
= NULL) +
          scale_x_continuous(breaks = seq(0, 48, 1), limits = c(0, 48.5), minor_breaks
= NULL) +
          coord_equal() +
          scale_fill_viridis_c() +
          theme_minimal() +
          theme(axis.title = element_blank(),
                axis.text = element_blank(),
                legend.position = "none")
        ggsave(plot=p, filename=file_name, width=1, height=1, units="in", dpi=150,
device="png")
      }
    }
    print(paste("Agent",i,"simulated..."))
  }
  return(density_mat)
}

get_monetary_loss <- function(row, item_df, total_ghi, n_agents) {
  coord = row[1:2]
  density = row[3]
  value_lost<-0
  if(density == 0) {return(value_lost)}
  adj<-c()
  for(i in 1:-1)
    for(j in 1:-1)
      if(i!=0 || j !=0)
        adj<-rbind(adj,coord+c(i,j))
  for(i in 1:nrow(adj)) {
    if(adj[i,1]==0 | adj[i,2]==0 | adj[i,1]>48 | adj[i,2]>48) {next}
    if(item_df[which(item_df$y==adj[i,1] & item_df$x==adj[i,2]),3]==0) {next} else {
      item <- item_df[which(item_df$y==adj[i,1] & item_df$x==adj[i,2]),3:7]
    }
    e_sold = n_agents * (item$ghi / total_ghi)
    e_left = item$qty - e_sold
    if (e_left >= 0) {
      avg_remaining = (item$qty + e_left)/2
      integrated_qty = 1/2 * (item$qty-e_left) + e_left
```

```
    } else {
      time_sold_out = item$qty/e_sold
      integrated_qty = 1/2 * item$qty * time_sold_out
    }
    value_lost = value_lost + (density^2/100000) * item$discounted_price * item$frag *
integrated_qty
  }
  return(as.numeric(value_lost))
}

# Some loss functions
get_loss <- function(densities) {
  loss = sum(densities)
  return(loss)
}

get_loss_sqrt <- function(densities) {
  loss = sum(sqrt(densities))
  return(loss)
}

get_loss_log <- function(densities) {
  loss = sum(log(densities+1))
  return(loss)
}

norm_loss <- function(loss, get_loss, worst_case) {
  loss_max = get_loss(worst_case)
  return(loss/loss_max)
}

get_loss_mat <- function(storedata, density_mat, target_df, n_agents) {
  density_df <- make_df_full(density_mat)

  item_mat <- make_mat(target_df, "value")
  item_df <- make_df_full(item_mat)
  item_df <- item_df[order(-item_df$value),]

  item_df$frag <- c(rev(storedata$frag), rep(0, nrow(item_df)-length(storedata$frag)))
  item_df$ghi <- c(rev(storedata$ghi), rep(0, nrow(item_df)-length(storedata$ghi)))
  item_df$discounted_price <- c(rev(storedata$discounted_price), rep(0, nrow(item_df)-
length(storedata$discounted_price)))
  item_df$qty <- c(rev(storedata$qty), rep(0, nrow(item_df)-length(storedata$qty)))

  total_ghi = sum(storedata$ghi)

  loss_mat <- matrix(0, nrow=48, ncol=48)
  loss_mat[] <- apply(density_df,1,get_monetary_loss, item_df=item_df,
total_ghi=total_ghi, n_agents=n_agents)
  return(loss_mat)
}

simulate <- function(pbm_path, store_layout = NULL, storedata, n_agents=100,
loss_fn=get_loss, max_routes=3, coeff=0.1, reps=5, plot=FALSE, name="density_plot",
from_bitmap=TRUE) {
  # This value is technically not necessarily the same for all agents, but we're
assuming it is
  print("Reading store layout from bitmap...")
  if(from_bitmap == TRUE) {
    store_layout <- read_img_map(pbm_path)
  } else {
    store_layout = store_layout
  }
  img_w = dim(store_layout$walls_mat)[1]
  img_h = dim(store_layout$walls_mat)[2]
  print("Store layout loaded.")

  print("Randomly generating agents...")
  agent_list <- create_agent_list(store_layout, n_agents)
  print("Running density simulation...")
```

```
  density_mat <- simulate_density(store_layout, agent_list, coeff, plot, name, img_w,
img_h)
  print("Density simulation complete.")

  print("Computing estimated loss...")

  loss_mat <- get_loss_mat(storedata, density_mat, store_layout$target_df, n_agents)

  loss_df <- make_df_full(loss_mat)

  loss <- loss_fn(loss_df$value)

  print("Simulation completed successfully.")

  output = list(density_mat, loss_mat, loss)
  return(output)
}

plot_output <- function(output, filename) {
  results <- output[[2]]
  store_layout <- output[[1]]
  density_mat_avg <- matrix(0, 48, 48)
  loss_mat_avg <- matrix(0,48,48)
  loss <- c()
  for(i in 1:length(results)) {
    density_mat_avg <- density_mat_avg + results[[i]][[1]]
    loss_mat_avg <- loss_mat_avg + results[[i]][[2]]
    loss <- c(loss, results[[i]][[3]])
  }
  loss_avg <- mean(loss)
  loss_sd <- sd(loss)
  density_mat_avg <- density_mat_avg/length(results)
  loss_mat_avg <- loss_mat_avg/length(results)

  density_df_avg <- make_df_full(density_mat_avg)
  loss_df_avg <- make_df_full(loss_mat_avg)

  walls_df <- make_df(store_layout$walls_mat, 1)
  entrance_df <- make_df(store_layout$entrance_mat, 1)
  exit_df <- make_df(store_layout$exit_mat, 1)
  target_df <- store_layout$target_df

  p1<-ggplot(walls_df, aes(x=x, y=y)) +
    geom_rect(xmin=0.5,xmax=48.5,ymin=0.5,ymax=48.5, fill="#000000") +
    geom_tile(data=walls_df, fill="#FFFFFF") +
    geom_tile(data=entrance_df, fill="#FF0000") +
    geom_tile(data=exit_df, fill="#0000FF") +
    geom_tile(data=target_df, aes(fill=ghi)) +
    scale_y_continuous(breaks = seq(0, 48, 4), limits = c(0, 48.5), minor_breaks =
NULL) +
    scale_x_continuous(breaks = seq(0, 48, 4), limits = c(0, 48.5), minor_breaks =
NULL) +
    coord_equal() +
    theme_minimal() +
    scale_fill_distiller(palette="Spectral") +
    theme(axis.title = element_blank(),
          axis.text = element_blank()) +
    labs(fill="GHI", title = "Store Layout")
  ggsave(plot=p1, filename=paste(filename,"_layout.svg",sep=""), width=5, height=5,
units="in", dpi=300, device="svg")

  p2<-ggplot(density_df_avg, aes(x=x, y=y, fill=value)) +
    geom_tile() +
    geom_tile(data=walls_df, fill="#FFFFFF") +
    geom_tile(data=entrance_df, fill="#FF0000") +
    geom_tile(data=exit_df, fill="#0000FF") +
    scale_y_continuous(breaks = seq(0, 48, 4), limits = c(0, 48.5), minor_breaks =
NULL) +
    scale_x_continuous(breaks = seq(0, 48, 4), limits = c(0, 48.5), minor_breaks =
NULL) +
```

```
    coord_equal() +
    theme_minimal() +
    scale_fill_viridis_c() +
    theme(axis.title = element_blank(),
          axis.text = element_blank()) +
    labs(fill="Density", title = "Pedestrian Density")
  ggsave(plot=p2, filename=paste(filename,"_density.svg",sep=""), width=5, height=5,
units="in", dpi=300, device="svg")

  p3<-ggplot(loss_df_avg, aes(x=x, y=y, fill=value)) +
    geom_tile() +
    geom_tile(data=walls_df, fill="#FFFFFF") +
    geom_tile(data=entrance_df, fill="#FF0000") +
    geom_tile(data=exit_df, fill="#0000FF") +
    scale_y_continuous(breaks = seq(0, 48, 4), limits = c(0, 48.5), minor_breaks =
NULL) +
    scale_x_continuous(breaks = seq(0, 48, 4), limits = c(0, 48.5), minor_breaks =
NULL) +
    coord_equal() +
    theme_minimal() +
    scale_fill_viridis_c() +
    theme(axis.title = element_blank(),
          axis.text = element_blank()) +
    labs(fill="Loss", title = paste("Mean Loss =",round(loss_avg, 2),"SD
=",round(loss_sd,2)))
  ggsave(plot=p3, filename=paste(filename,"_loss.svg",sep=""), width=5, height=5,
units="in", dpi=300, device="svg")

  plots <- list(p1, p2, p3)
  return(plots)
}
```

9.1.4 Item Placement

```
source('simulate.R',local=T)
# Get the store_layout for editing
store_layout <- read_img_map("example.pbm")

# Core place_items function; no need to edit this code
place_items <- function(store_layout, storedata,select_positions,
order_positions, order_items,threshold,x,y) {
  #differentiate and sort prominent and normal items
  prominent_items<-storedata[which(storedata$ghi>=threshold),]
  prominent_items<-prominent_items[order(prominent_items$ghi),]
  normal_items<-storedata[which(storedata$ghi<threshold),]
  normal_items<-order_items(normal_items)
  #create the sorted layout
  storedata_sorted<-rbind(prominent_items,normal_items)
  #import layouts
  walls_mat <- store_layout$walls_mat
  blocked_mat <- store_layout$blocked_mat
  shelf_mat <- walls_mat-blocked_mat
  # Zero out the entrance wall
  # Convert the available spots into df format for sampling
  shelf_df <- make_df(shelf_mat, 1)
  # Select 134 shelf positions from possible positions
  shelf_positions <- select_positions(shelf_df)
  # Insert an item into each shelf position
  ## Sort positions by increasing y position
  shelf_positions<-order_positions(shelf_positions,x,y)
  ## Bind positions to items
  shelf_positions$value <- storedata_sorted$item_id
  shelf_positions$ghi <- storedata_sorted$ghi
  shelf_positions <- shelf_positions[order(shelf_positions$value),]
  store_layout$target_df<-shelf_positions
  return(store_layout)
}
```

```
# Some example selection/ordering functions (you'll have to write more of
these yourself)

# A select_position function takes in a shelf_df, and selects 134 positions
from it somehow
select_positions_random <- function(shelf_df) {
  shelf_positions <- shelf_df[sample(nrow(shelf_df), 134),]
  return(shelf_positions)
}

select_positions_adaptive <- function(shelf_df){
  decay<-5
  items<-134
  tally<-0
  shelf_positions<-NULL
  for (i in 1:48){
    n<-48-i
    s<-shelf_df[which(shelfdf$y==n),]
    shelf_temp<-s$x%%decay==0
    tally<-tally+nrow(shelf_temp)
    if ((items-tally<items/10)&(decay>1)){
      decay=decay-1
      tally=0
    }
    items=items-nrow(shelf_temp)
    shelf_positions<-cbind(shelf_positions,shelf_temp)
  }
  return(shelf_positions)
}

select_positions_static<-function(shelf_df){
  shelf_df<-shelf_df[order(shelf_df$x),]
  shelf_df<-shelf_df[order(shelf_df$y),]
  shelf_positions = shelf_df[seq(1, nrow(shelf_df),
floor(nrow(shelf_df)/134)),]
  return(shelf_positions)
}

# An order position function takes in a shelf_positions df with 134 rows, and
orders them somehow
order_positions_ascending_y <- function(shelf_positions) {
  shelf_positions<-shelf_positions[order(shelf_positions$y),]
  return(shelf_positions)
}

order_positions_descending_y <- function(shelf_positions) {
  shelf_positions<-shelf_positions[order(-shelf_positions$y),]
  return(shelf_positions)
}

order_positions_euclidean<-function(shelf_positions,x,y){
  shelf_positions$euclidean<-(shelf_positions$x-x)^2+(shelf_positions$y-y)^2
  shelf_positions<-shelf_positions[order(-shelf_positions$euclidean),]
  shelf_positions$euclidean<-NULL
  return(shelf_positions)
}
# An order_items function sorts store_data based on some attribute (e.g. ghi,
fragility, wtvr else)
order_items_ascending_ghi <- function(storedata) {
  storedata_sorted<-storedata[order(storedata$ghi),]
  storedata_sorted<-storedata[order(storedata$dpmt),]
  return(storedata_sorted)
```

```
}

order items random <- function(storedata) {
  storedata_sorted <- storedata[sample(storedata$item_id),]
  storedata_sorted<-storedata[order(storedata$dpmt),]
}
```