Automated Planning for Menu Planning

[redacted]

April 2021

Abstract

Automated planning is a sub-field of artificial intelligence that allows us to move from a current state to a desired state in a logical way. Menu planning is a complex and time-consuming process by which a suitable menu is drafted, selecting appropriate meals to include according to budget, dietary requirements, and time constraints, among other factors. The aim of this project is to create a better system for menu planning using artificial intelligence that will allow a user to generate a suitable menu based on cost, variety, dietary requirements, and preference.

I certify that all material in this dissertation which is not my own has been identified.

Contents

1 Introduction

1.1 Motivations

In event catering, menu planning is a time-consuming, complicated process. Factors such as dietary requirements, group size, allergies, budget, and timings must all be considered, and last-minute changes to any of these often involve a complete redo of all this careful planning. As an event caterer myself, I spend a lot of time on this task, which usually involves multiple spreadsheets and cross-referencing with various cookbooks and supermarket websites. My job involves travelling to the site where the event is being held and catering for groups of 12 — 60 people for the weekend (usually Friday dinnertime through to Sunday lunchtime), or sometimes for a week. The event sites sometimes do not have their own kitchen equipment, meaning that on top of planning the menus and shopping for the ingredients I need to pack all the kitchen equipment that I will require, making sure I think of everything I will need but also that I do not over-pack (which often happens due to the complexities of working out exactly which equipment I will need and when it will be needed).

1.2 Objectives

The goal of my project is to create a better system for menu planning for event catering using artificial intelligence. My project will be a program that allows input of the size of the group to be catered for; any dietary requirements that members of the group have; the "per person" budget; the specific meals to be provided; and the meal timings. The program will then use automated planning to create a suitable menu from a database of meals and their recipes. In addition to the program's basic function of generating a menu, it should be able to generate a shopping list, minimal list of kitchen equipment needed for the event and a detailed schedule showing exactly which food preparation needs to be completed and when. My objectives for this project will therefore be as follows, in order of importance:

1. Using artificial intelligence, create a menu planner to suggests a suitable menu based on user inputs.
2. Automatically generate an accurate timeplan, equipment list and shopping list for these menus.
3. Create a process to suggest suitable products for each ingredient in a menu

I am designing this with event catering in mind, but if the project is successful then there are many other applications that it could have – hospital catering and military food provisions are two possibilities.

1.3 Literature Review Summary

In my literature review I explored the field of automated planning, which is a way of reaching a goal state by finding a logical set of actions to move from an initial state to this desired end state. The purpose of the literature review was to evaluate existing approaches to automated planning and assess their suitability to a menu-planning problem to determine the most suitable design and back-end logic for my chosen project. Automated planners have significantly advanced in both power and complexity since their conception, allowing them to make use of temporal planning [1], user preferences [2][3], and human-like planning features such as hierarchical planning and learning from experience [4][5][6], factors which are relevant to my project. Machine learning is perhaps the most advanced form of automated planning to date due to its ability to improve itself. It is evident that a reinforcement machine learning approach will be the most suitable due to its ability to produce plans without access to explicit training data [7]. One way of doing this is through reinforcement learning, a form of artificial intelligence in which a system learns how to make good choices via a reward function, which tells the system whether a choice it has made is good or bad depending on the context of the task [8]. I intend to use this process to create an automated menu planner that can generate a suitable menu and accompanying plans for event catering based on user inputs.

2 Specification

2.1 Requirements

ID	**Functional Requirements**
F1	The software has a user interface for initial data input from the user
F2	The software connects to a database to store necessary data
F3	A user can input and save detailed information on new recipes
F4	A user can specify how many people will be catered for and define their dietary restrictions
F5	A user can specify the meal timings and the start and end dates for the event
F6	A user can specify a preferred budget
F7	The software uses artificial intelligence to generate a suitable menu and schedule
F8	Every group member stage must have a suitable meal at each mealtime
F9	A time plan and shopping list for the final menu must be produced
F10	A user can edit suggested menus before saving or exporting them
F11	A user can export menus, shopping lists, and time plans
F12	A user can save menus within the application
F13	A user can create an account via the application to access their saved recipes

ID	**Non-Functional Requirements**
NF1	The software should produce its suggested meal schedule within a suitable timeframe
NF2	The menu should contain a reasonable level of variety
NF3	The user interface should be intuitive and easy to use
NF4	The meals chosen for groups with dietary restrictions must comply with those restrictions
NF5	User account data should be stored securely in the database
NF6	The software must behave predictably and appropriately for its given function

2.2 Overview

2.2.1 Approach and Languages

A reinforcement learning approach will be used to select meals for each group member in each timeslot of the menu by defining a reward function that has a positive output if the menu choices are good and a negative output if the choices are bad. Through this, my software will learn to create suitable menus and improve its meal choice selections. The application will consider cost, preference score, variety, and number of concurrent meals to be cooked to determine whether a menu is good.

The structure of the program will consist of a front-end for user input, a back end for the menu planning and program logic, and a MySQL database to store data. I will use MySQL for the database. The front and back-end of the system will be written in Python, as it provides good functionality for both machine learning and front-end web development. I will be using the Flask framework, which does the 'heavy lifting' of web development and allows me to concentrate on the function of my application.

2.2.2 User Inputs

The software will be initialised with predefined recipes and special diets (sourced from my personal catering cache), with the option for the user to add more recipes if required. The user will input the group size, event start and end dates, required meals, meal timings, and budget, before identifying group members with special diets.

2.2.3 Representation of Data

Ingredients will have an ingredient id, a name, and a list of allergens (represented by ingredient ids). **Products** will be accessed via the Tesco Product Information API to allow for accurate pricing. **Dietary restrictions** will be represented by a list of forbidden ingredients. A recipe satisfies a dietary requirement if it contains none of these ingredients and if its ingredients also contain none of these ingredients in their lists of allergens. **Recipes** will be represented by an id, name, description, number of servings, mealtime (i.e. breakfast or dinner[1]) and category (e.g. soups, stews etc). Associated with each recipe will be a list of ingredients, a list of dietary requirements satisfied by the recipe and a list of steps. **Steps** will be defined by their recipe id, step number, start time, end time and description.

2.2.4 Planning Method

A very basic algorithm for my software will involve the following steps:

1. Produce a list of possible meals for each group member at each mealtime slot.
2. Combine the meal lists for group members with identical meal lists.
3. Fill in the meals schedule with suitable meals according to the budget and variety constraints.
4. Allow the user to edit the menu if desired.
5. Produce a shopping list, equipment list, and time plan for the final menu.

2.3 Evaluation Criteria

The functional and non-functional requirements form part of the evaluation criteria for this project and will be assessed via a combination of unit testing, integration testing and user testing. If my software also satisfies the following criteria to the required degree, then it will be a useful and time-saving alternative to manual menu planning for catering and the project can be considered a success.

Variety
Variety is an important metric to consider when evaluating menu suitability [9]. To ensure variety between meals, the software will use set minimum intervals between meals that are similar and meals that are the same. Since variety in breakfasts is usually seen as unimportant, I will only apply this evaluation criteria to lunches and dinners, disregarding breakfasts when calculating the intervals. Treating the timeline of meals as a cycle of lunch-dinner-lunch-dinner, the minimum interval for meals that are the same will be 7, and for similar meals it will be 3. These values can be changed by the user if desired.

Cost
Once the software has found a suitable menu and a shopping list of required ingredients and their quantities, the total cost can be calculated using the Tesco Product API. If the cost of the menu is below the user's budget, then this is a suitable plan. It may occasionally be impossible to generate a menu within the user's budget. In this situation, a menu can be a good menu if it is as close as possible to the budget.

Number of Concurrent Meals
The program will therefore need to minimise the number of concurrent meals. Group members with identical dietary requirements will be combined and for each group in a mealtime the meal selection process will favour meals already in the menu (if they are suitable for that group).

[1]This project assumes that for event catering, all lunch and dinner foods will be hot, cooked meals. Since hot lunch meals are suitable for dinner, and vice versa, lunch and dinner meals will be combined in a single 'dinner' category for simplicity.

3 Design

3.1 User Interface

The project is structured as a web application, written with the Flask Python web framework. The user interface is a simple set of HTML pages consisting of a login page, a registration page and a homepage which leads to options to view or add menus and recipes.

3.1.1 Registration/Logging in

For security, passwords are hashed before being stored in the database. Hashing here refers to converting a plain-text password into an unreadable string of characters which represent the original password string. This measure increases the security of stored passwords and helps to protect user data.

3.1.2 Homepage

The homepage is a simple menu, with options for a user to add new recipes to the database, view saved recipes, create a new menu, view/edit existing menus, update estimated ingredient costs, and logout.

3.1.3 Recipes

The 'View Recipe' page presents users with a list of recipes saved in the database, consisting of the default recipes and their own personally added recipes. These are listed in alphabetical order by name and clicking on any of the names leads to a page showing the details of that recipe, including serving size, dietary information, ingredients, and method. This allows users to get more information about recipes that they may wish to add to their menu, as it would be difficult to fit all this information on the menu editing pages.

The 'Add Recipe' page allows users to add their own personal recipes to the database to be used in the menu planner. These are entered in a specific format to ensure compatibility with the rest of the application's functionality. Users can specify their own 'preference score' for recipes that they add, which will affect how likely the meal is to feature in menus that the application suggests. This allows for greater personalising of the user experience.

3.1.4 Menus

The 'Add Menu' option will allow users to input a group size, event dates, mealtimes and information on members with specific dietary requirements. On confirmation form the user, this information will be saved in the database and the 'Menu Editor screen will be shown, where users can choose to either manually select meals for their menu or have the application automatically generate a suggested menu for them. There will be a control panel where they can edit their budget and change the prioritisation of the various factors by which a menu is suggested. Once the user is happy with their menu, they can choose to save it in the database, after which they will be redirected through the shopping list and timeplan pages.

The 'View Menu' page will display a list of the user's saved menus, along with options to edit, delete and export them. The 'edit' option will lead them through to the menu editor described in the previous paragraph, and the 'delete' option will remove the selected menu and all associated data from the database. The 'export' option will allow the user to download a PDF file containing the menu plan, time plan, shopping list and equipment list, as well as an appendix containing the full text of all recipes used in the menu.

3.2 Back-end

3.3 Database

The database design for the application is displayed in the diagram in Figure 1. It contains the key data types of **Menus**, **Recipes**, **Ingredients**, **Users**, **Special Members**, **Diets** and **Equipment**, as well as several bridge tables relating these data types to each other.

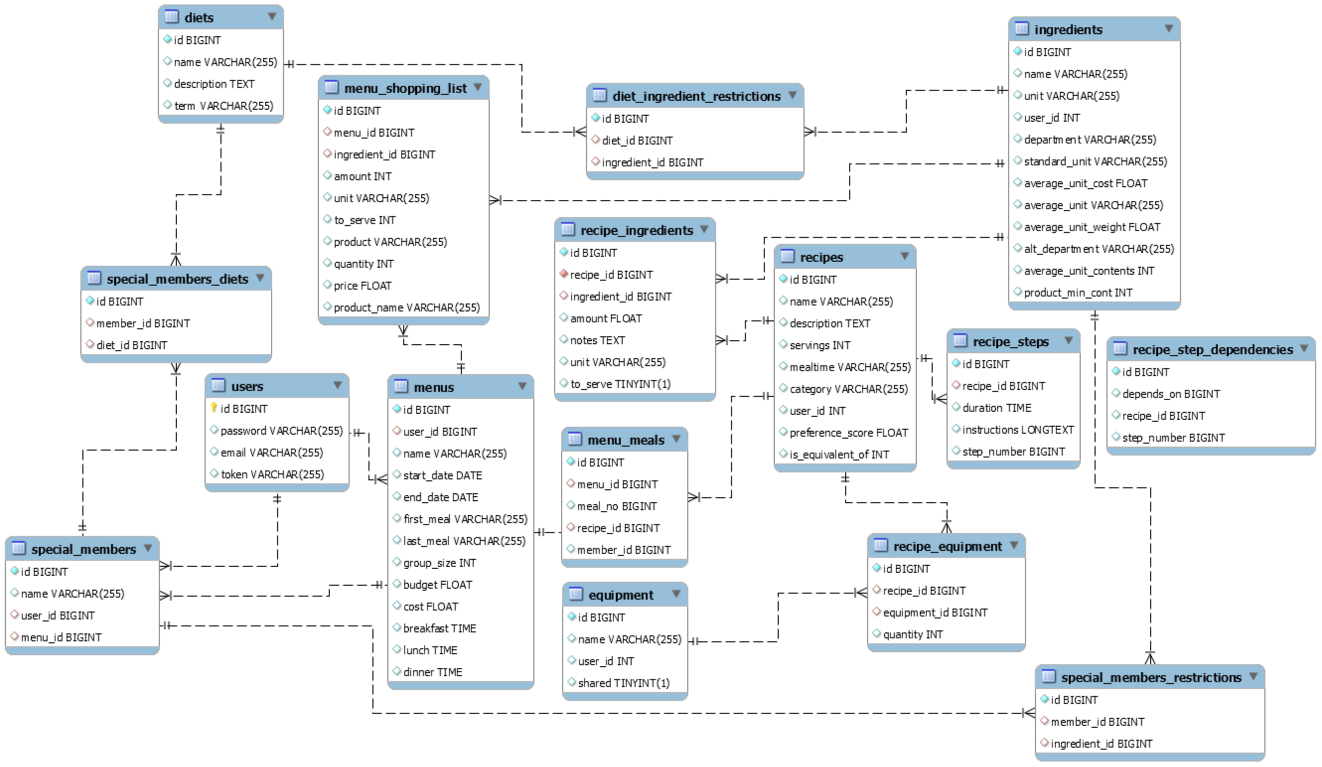


Figure 1: ER diagram of the application's database design

3.3.1 Meal Selection

As mentioned earlier, the meal schedule is organised into a set number m of mealtimes, with each mealtime being split into a number g of groups. Users will be able to manually select meals for their menu using a grid of selection boxes populated with the meals suitable for each group. Alternatively, they will be able to use the application's AI functionality to automatically generate a suitable menu for them. This suggested set of meals will be populated into the manual meal selection boxes, allowing users to change meals in the menu before saving it to the database.

First, the learning algorithm runs for 150 iterations, generating menus and calculating their preference score and cost. Menus are selected via an **epsilon-greedy** approach. This balances exploration of the possible choices against the technique of choosing the choices with the highest current reward to enable the program to explore sufficient choices, but also to converge to an optimal solution relatively quickly. I chose an initial epsilon value of 0.5 with the intent to refine this further. This means that 50% of the time, meals for a menu are selected randomly, and 50% of the time they are selected based on the current best meals, as defined by the meal's current reward. The learning rate, alpha, will be 0.5 initially — this value, from 0 to 1, determines the rate at which new information is incorporated into the algorithm. A learning rate of 0 means that the algorithm does not learn at all, whereas a rate of 1 means that only the newest information is considered.

A heuristic domain-specific approach is used to optimise the concurrency of the menus by having the algorithm choose meals for each group that are already selected in the menu for another group, if suitable for the current group's dietary needs. For example, if a vegan meal had already been selected and a group of vegetarians was also in the meal plan, the algorithm would select the vegan meal again for them to minimise concurrency.

The breakfast selection process is different from the lunch/dinner selection process since people are usually happy to eat the same thing for breakfast each day, which is not the case with the other meals. Breakfasts are thus selected for each 'dietary needs' group by choosing the breakfast meal with the highest preference score in that group, regardless of concurrency. Only one breakfast combination needs to be generated, and this is used for each breakfast slot in the menu.

After each iteration, a reward is given proportional to the cost and preference score of the given combination of meals. The reward is negative if the menu is 'good' and negative if the menu is 'bad'. These rewards accumulate over the course of the iterations, so that when the learning process has finished a menu can be generated based on the best meals in the rewards list. This final menu us then presented to the user, who will have the chance to edit it before saving.

Once the user chooses to save the menu to the database, a shopping list is generated containing the necessary quantity of each ingredient in the menu. This is also saved in the database and can be accessed using the Shopping List page, where the different products available for each ingredient can be viewed.

3.3.2 Shopping List

There are several criteria that will need to be considered when selecting products to match an ingredient. Firstly, suitability for dietary constraints, and secondly, price. For each ingredient used in a menu, the product selection algorithm should first find all the special diets of the members eating the meals that the ingredient is a component of. Each product selected for that ingredient must therefore comply with the dietary requirements of those groups.

Users will be able to select their desired products for each ingredient in a shopping list and see the total price of the list, the quantity needed of each product to match the amount of ingredient needed, and the total price of each entry in the shopping list. This is reasonably straightforward in most cases, involving a division of the total quantity of the ingredient by the unit quantity of the selected product. However, in certain cases the calculation is a little more complicated — for example, sausages are usually listed by unit in recipes (e.g. '4 sausages') but sold by weight. For cases such as these, an "average unit weight" column is present in the database. Since there are only a few of these cases it was possible to go through and manually calculate the average unit weight from the best result for each of these ingredients given by the Tesco API data. A similar column, "average contents quantity", is present for ingredients such as garlic cloves or slices of bread, where the products are sold as a unit containing a certain quantity of the ingredient (for example, garlic is sold by the bulb, with each bulb containing an average of 10 garlic cloves).

3.3.3 Timeplan

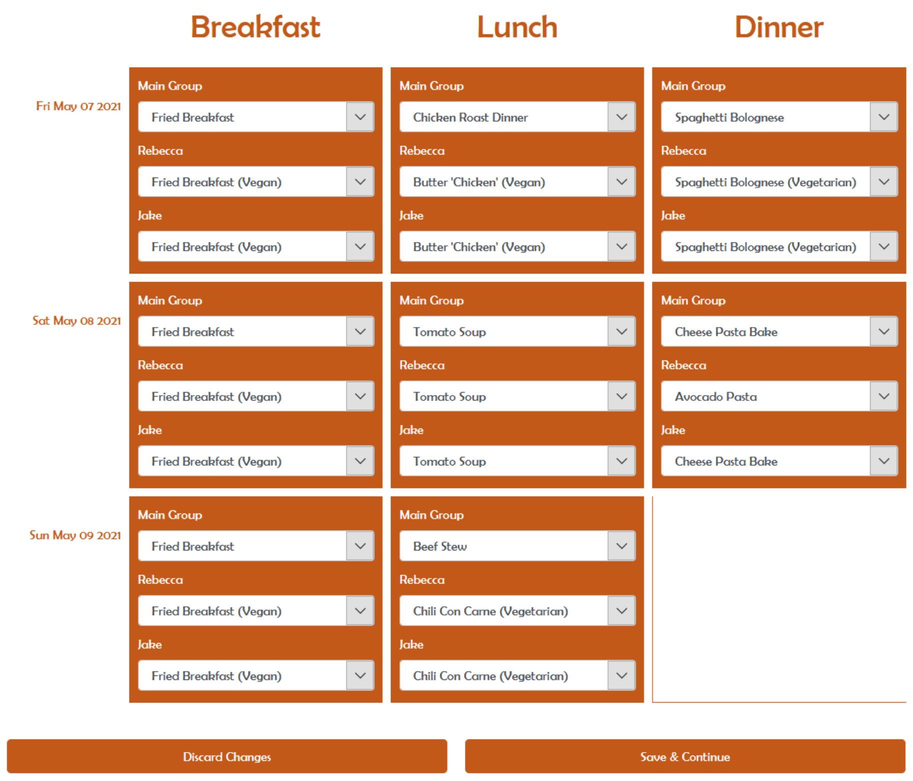
For each mealtime, the software will generate a timeplan. This is a series of instructions that describe the food preparation tasks and their relative timings leading up to the meal serving time. This will be generated from the time steps for each recipe in the menu, which are saved in a table in the database as shown in Figure 1.

4 Development

I followed an agile development process for this project. There are many reasons to use agile software development for a project like this — one such reason being that it breaks down a complicated development process into manageable units. This allows focus to be directed towards making sure high-quality results are delivered at each stage, with a minimum viable product being produced and tested. Errors and bugs are detected early and fixed before the next stage of development begins, which prevents these defects from perpetuating through the code and causing greater problems later in the development process. This saves time and effort throughout the project's entire life cycle.

My development process was organised into several two-week 'sprints', where the goal at the end of each sprint was to have produced a Minimal Viable product, or MVP. This allowed me to grow the application iteratively and focus on each piece of core functionality in turn. There were several main stages to the development, detailed below.

4.1 Basic Application Structure



		Breakfast	Lunch	Dinner
Fri May 07 2021	Main Group	Fried Breakfast	Chicken Roast Dinner	Spaghetti Bolognese
	Rebecca	Fried Breakfast (Vegan)	Butter 'Chicken' (Vegan)	Spaghetti Bolognese (Vegetarian)
	Jake	Fried Breakfast (Vegan)	Butter 'Chicken' (Vegan)	Spaghetti Bolognese (Vegetarian)
Sat May 08 2021	Main Group	Fried Breakfast	Tomato Soup	Cheese Pasta Bake
	Rebecca	Fried Breakfast (Vegan)	Tomato Soup	Avocado Pasta
	Jake	Fried Breakfast (Vegan)	Tomato Soup	Cheese Pasta Bake
Sun May 09 2021	Main Group	Fried Breakfast	Beef Stew	
	Rebecca	Fried Breakfast (Vegan)	Chili Con Carne (Vegetarian)	
	Jake	Fried Breakfast (Vegan)	Chili Con Carne (Vegetarian)	

Figure 2: The basic menu planning interface

In this stage of development my goal was to create the base structure of the application and populate the databases. I set up a new Flask application, hosted locally on my machine, and created a registration/login system, a homepage and functionality for adding and saving new recipes and menus in the required format. I also created the functions which would combine members of a catering group with identical dietary requirements, and populated the database with 14 special diets, 34 recipes and 600 common ingredients to use in initial testing. I decided on the name Tagine for the application — a tagine is the name of a conical earthenware pot used for cooking, and also the name of a dish from Berber which is cooked in a tagine. Due to the culinary connection it seemed like a fitting name. I created a simple logo and chose a clean orange and white colour scheme to give an earthenware/terracotta feel to the application. The orange that I chose as the main theme colour for the application was bright enough to be cheerful and eye-catching yet dark enough to be readable.

I imported and used the Pandas package at this stage, which is a Python software library used for data analysis and development. I mainly used the data frame structure, which is a tabular two-dimensional data structure that allows for labelling of axes (rows and columns). This allowed me to have a matrix with group members with special dietary requirements as the column labels, meals in the database as the row labels, and 0s and 1s as data in the matrix body to represent which members could eat which meals.

this allowed me to easily combine members with identical requirements by simply combining the columns with identical patterns of 0s and 1s.

The MVP produced at this stage was a 'dumb' menu planner — an application that allowed the user to input a budget, group numbers and their dietary needs, and event timings. The user would then be presented with a menu editor in which they could create a meal plan using a series of dropdown boxes, populated with all the meals in the database suitable for each group and mealtime. The design of this is illustrated in Figure 2. The program would then calculate an overall list of ingredients and the estimated cost of the menu (based on an estimated price of products drawn from the Tesco Product Information API).

4.2 Menu Planning Algorithm

The next stage of the development process was to add the AI functionality to the meal planner. This required the development of four functions that would measure the concurrency score, user preference score, cost score and variety score of a menu. These would be used to calculate the reward given for each combination of meals and mealtimes. I also developed the control panel that would allow users to edit the factors influencing menu selection.

I initially intended to have the variety and concurrency of a menu be preferences that would influence the reinforcement learning process of the meal selection algorithm. However, it quickly became apparent that it would be far better to have these factors as hard constraints, and for the system to only consider the preference score and cost of a menu when evaluating its suitability. In the section of my code where menus are generated for the program to learn from, I ensured that only menus that fit the (user defined) concurrency and variety constraints. This allowed the learning process to better focus on the more subjective parts of the menu in its evaluation of suitability.

Each ingredient stored in the database has an estimated unit price associated with it. This average price is calculated by getting the average unit price of the top 5 products for each ingredient via the Tesco Search API in a similar process to the way the shopping list is developed in Section 4.3. The value is calculated and stored in the ingredients table in the database each time the user makes a request to update the price estimates — this is handled through a different section of the application to avoid making calls to the API every time a menu is generated as this would be inconsiderate of Tesco's resources. The user can update these unit price estimations by visiting the Update page, allowing them to keep the data as up to date as they need.

The estimated cost of a menu is calculated using these average unit prices, at which point a cost score is calculated. If the cost is higher than the user's budget then the reward given is -1 — this is a constant, large negative reward as going over the user's budget is a behaviour that should be strongly discouraged. If the cost is below the user's budget, then the reward returned will be 1.

The overall preference score of a menu is calculated as follows:

$$\frac{\sum\left(\frac{preference\ score\ of\ each\ meal}{number\ of\ people\ eating\ that\ meal}\right)}{10 * number\ of\ meals\ in\ menu}$$

This ensures that preference scores are proportional and dividing by 10 keeps the value below 1 to correspond with the cost reward. However, this would always give a positive reward if used directly, since preference scores cannot be negative. This would have meant that preference had a lesser effect than desirable. To counteract this and differentiate between 'bad' and 'good' menus I decided that the reward for the preference component should be equal to the overall preference score of the menu, minus the average preference score for all meals in the database. This meant that menus could be classified as either better or worse than average and a reward or punishment given appropriately.

This reward value still was not quite sufficient — I wanted preference to be roughly equal to cost in terms of their effect on the learning algorithm. The cost reward ranges between -1 and 1, but the preference scores of the meals ranges from 5.34 to 8.89, with an average of 7. This means that the range of possible values for the preference reward would be between -0.234 and 0.189, proportionally a lot smaller than the cost reward range. to counteract this I multiplied this value by 5, giving a range of values between -1.17 and 0.945, which is much closer in proportion to the cost reward range.

The meal preference scores were calculated using the results from an anonymous survey asking respondents to rate meals with a score between 1 and 10. There were 57 responses, and approximately 70 meals included. These preference scores (divided by 10 to keep them between 0 and 1) are used as the initial reward values in the learning algorithm to help improve the preference scores of the output menus.

As described in my design in section 3.3.1, I initially chose an alpha value of 0.5, a learning rate of 0.5 and used 150 iterations in my algorithm for meal selection. After some basic testing I determined that the algorithm was learning 'too fast' — historical data was not being considered enough and the reward totals for each meal were changing too quickly for preference scores to be adequately considered. However, reducing the learning rate required a significant increase in the amount of iterations needed for the learning process which would have come at a significant cost to the speed at which results were generated. I solved this by having my learning rate 'decay' — at the beginning, the learning rate is high so that large leaps are made towards finding a solution. The learning rate decreases with every iteration so that in the later stages of the learning process a solution can be effectively narrowed down with smaller steps. My learning rate was set to be $1 - \frac{z}{num\_iterates}$, with z being the current iterate number and $num\_iterates$ being the total number of iterations, so that it would decrease proportionally over time.

I suspected that the initial epsilon value of 0.5 could also be improved. I was unsure if a greater or smaller amount of randomness was needed, so I ran simulations with an epsilon of 0.25 and then of 0.75 to see which direction the parameter change needed to be. The results for the epsilon value of 0.75 were generally higher in preference score and lower in cost than those with an epsilon of 0.25 and were also less varied as they were grouped much more tightly in the graph that I plotted. This seemed to prevent the reward total for one specific meal from growing too fast and artificially making it seem as though this was the 'best' meal before sufficient alternatives had been considered. Increasing epsilon to 0.75 (75% random choices, 25% greedy choices) allowed a higher probability that 'unchosen' meals could be chosen and tested, allowing for a broader view of the data.

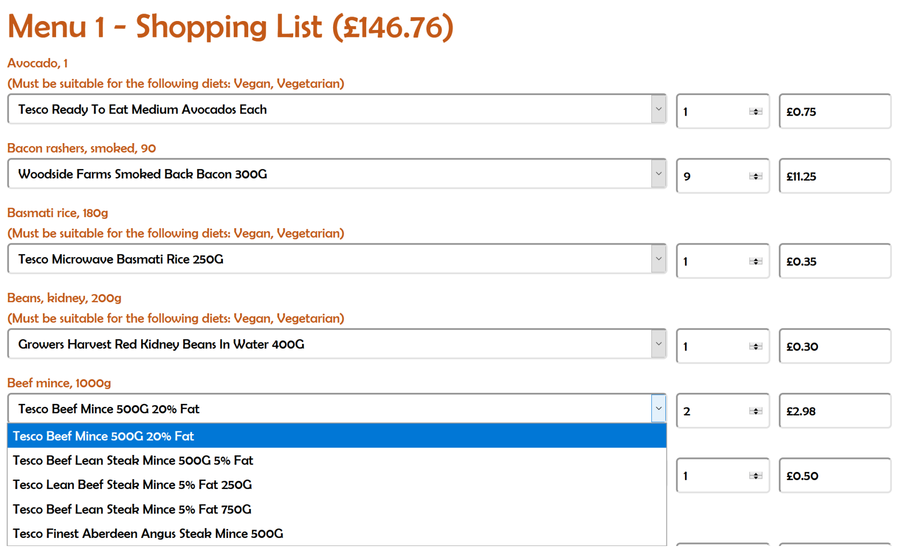
I found 150 iterations to be sufficient in the majority of cases, but to further optimise the application and add a fail-safe against potential unforeseen complicated scenarios I added the caveat that if the generated menu's cost at the end of the iterations is above the budget, the algorithm should continue learning until either a below-budget menu is found, or the process has repeated 10 times. The limit of 10 here is to safeguard against running an infinite loop if the user's budget is simply too low to find a suitable menu for.

4.3 Product Selection

To select relevant products for each ingredient in the shopping list I used the Tesco Search API and the Tesco Product Information API. The Search API allows me to search for products across the Tesco range, refining by department, so generate a list of potential products to be used. The Product API would have allowed me to refine this list by returning information on the dietary content of each product returned in the search, removing products from the shortlist which conflict with the dietary requirements of the members eating the meals in which that ingredient is used. However, sometime after implementing this functionality I lost access to my authorisation for this API due to changes in Tesco's Developer Portal policies (unrelated to my project). This meant that I was unable to include this feature in my final product, although I have left the functionality in the source code for completeness and in case access is restored in the future. Luckily, the product search functionality was unaffected due to the Search API having a different authorisation level to the Product API.

I decided to offer the user a range of products to choose from, rather than try to decide the 'best' product for them — this is due to the fact that individual users may have different preferences and priorities when it comes to selecting products.

The product range is ordered in such a way that a good product will be initially selected for each ingredient ('good' here referring to price and suitability), so that the users that prefer to let the application make the decisions for them can leave the selected options as-is and still have a suitable product shopping list generated. Allowing users to change the selected products also helps to mitigate the potential for human errors originating from the API from causing issues in the shopping list — by this, I mean issues such as products being placed in the wrong category and therefore showing up in unsuitable ranges, and other such errors that I have no control over.



Menu 1 - Shopping List (£146.76)

Avocado, 1
(Must be suitable for the following diets: Vegan, Vegetarian)

Tesco Ready To Eat Medium Avocados Each	1	£0.75

Bacon rashers, smoked, 90

Woodside Farms Smoked Back Bacon 300G	9	£11.25

Basmati rice, 180g
(Must be suitable for the following diets: Vegan, Vegetarian)

Tesco Microwave Basmati Rice 250G	1	£0.35

Beans, kidney, 200g
(Must be suitable for the following diets: Vegan, Vegetarian)

Growers Harvest Red Kidney Beans In Water 400G	1	£0.30

Beef mince, 1000g

Tesco Beef Mince 500G 20% Fat	2	£2.98
	1	£0.50

Tesco Beef Mince 500G 20% Fat
Tesco Beef Lean Steak Mince 500G 5% Fat
Tesco Lean Beef Steak Mince 5% Fat 250G
Tesco Beef Lean Steak Mince 5% Fat 750G
Tesco Finest Aberdeen Angus Steak Mince 500G

Figure 3: The product selection/shopping list interface, showing a drop-down menu of products

With no sorting criteria specified, products are returned from the Tesco API in order of relevance. I wanted the products to be displayed to the user as sorted by unit price but found it more efficient to sort the response from the API by price on the application end and make use of Tesco's relevance sorting algorithm, rather than trying to 'reinvent the wheel' by writing my own. My application therefore gets the 5 most relevant products from the API, removes those that do not correspond to the required dietary constraints, and sorts the list by the '*unitprice*' field.

Occasionally, ingredients only appear in a recipe as a serving suggestion to go with the meal, such as serving tomato soup, with crusty bread. There would be no need to classify this recipe as not being gluten free, for example, when the group members who cannot have gluten could not eat the bread with their soup.

My solution to this was to add an extra 'to_serve' column to the recipe_ingredients table (with a corresponding checkbox in the 'Add Recipe' page). If an ingredient in a recipe is only there to be served with the meal, then only the quantity of ingredient required for those who can eat the ingredient will appear the shopping list. This prevents food waste from buying ingredients that cannot be eaten and expands the range of meals available for group members with dietary requirements.

I added the option to save the chosen products in the application's database so that this information could be downloaded as part of the menu information file. Saving the products allows for greater efficiency in the program, as well as being considerate of the number of calls made to Tesco's APIs.

4.4 Timeplan Generation

In catering, a timeplan is a list of steps to be taken to complete a dish, containing a starting time for each step and a description of the actions to be taken at that time. This can become complicated when cooking multiple meals simultaneously, and so my application aims to reduce the complexity by automatically generating timeplans for each mealtime in the meal schedule.

To create accurate timeplans within my application, the method sections of all the recipes needed to be in a specific format. I originally tried having the steps saved with a start time and end time but found that having a duration and a set of dependencies allowed me to generate the timeplans much more effectively. By modelling the method steps and their dependencies as a directed graph, where the edges represent the steps and the nodes represent points in time, I was able to visualise how the steps and dependencies fitted together and come up with an algorithm inspired by Critical Path Analysis [10] to organise the steps into a timeplan.

Algorithm 1: Algorithm to generate a timeplan

```
Create empty array timeplan_steps;
Function Activity(step, finishtime):
    step start time = finish time - step duration;
    add step and start time to timeplan_steps array;
    remove step from unsorted_steps array;
    set new finish time = step start time;
    if step is dependent on other steps then
        for step in dependencies do
            activity(step, new finish time);
        end
    else
        break;
    end
    return;
for meal in mealtime do
    Create array of unsorted_steps and their
      dependencies;
    set finish time = meal serving time;
    for step in unsorted_timeplan_steps do
        activity(step, finish time);
    end
end
sort timeplan_steps array by step start time
```

The timeplans needed to be built backwards, starting at the time at which the meal was intended to be served. For each mealtime, the algorithm to generate a timeplan was as defined in Algorithm 1. It recursively adds steps to the timeplan, considering their start time, duration, and dependencies. Once all steps for all meals in the mealtime have been added, the timeplan is sorted by step start time. Each timeplan lists the step's start time, instructions, and recipe to which it belongs.

4.5 Creating the Download File

I used the 'pdfkit' package to create a formatted download file containing information from the user's menu. This document is designed to be printed and brought along to event sites where catering would be taking place, so consideration needed to be given to the presentation of the information.

The sections of my download file were as follows: a **cover page**, stating the menu's name and date; the **meal schedule**, listing all the meals for each member of the group at each mealtime with a footnote including details on relevant dietary requirements; a **shopping list**, including the suggested/chosen product and quantities for each ingredient in the meal plan; a minimal **equipment list**; **timeplans** for each mealtime; and the full **list of recipes** for the meals featuring in the menu.

For each section of my download file I created a new HTML template to hold the information. These were mostly based on both the back- and front ends of the original web pages in my application with some changes made as appropriate, such as removing logos and links, simplifying code, and converting form elements to static elements. I found that the pdfkit package does not support the CSS 'grid' property, so I needed to reformat a lot of the HTML templates to use tables instead.

The only section of the download file not directly based on a web page was the equipment list — I chose not to make this as its own section in the application since the only time it would be needed would be to bring to an event site. A minimal equipment list is generated by iterating through the equipment required for each recipe in each mealtime of the menu — the general principle being that if a piece of equipment appears in more than one recipe in a mealtime then multiples of the equipment must be brought (to prevent cross-contamination between different dietary requirements, and to prevent the situation where a piece of equipment is needed by two recipes at the same time) but if a piece of equipment appears across multiple mealtimes then the assumption is that it can be washed in between mealtimes and reused. The quantity needed of each piece of equipment can therefore be defined as the maximum number of recipes in which the equipment is used simultaneously in a single mealtime.

The equipment list in the download file features two columns of checklists to aid the user in packing.

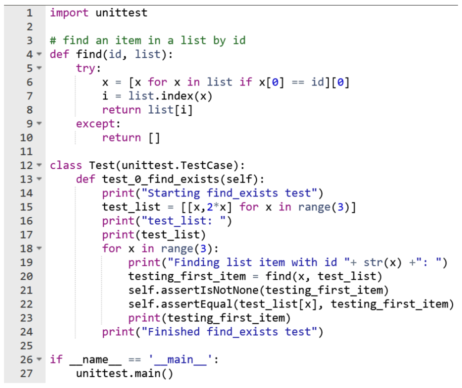
One column is to be used when packing to attend the event, and the other is to be used when gathering one's equipment from the event site in preparation for departure. This is a technique I use myself to avoid losing equipment, and so I have included it as I feel it enhances the user experience and provides greater efficiency during the catering process.

5 Testing

5.1 Unit Testing

Unit testing is the process by which each individual component of a system is evaluated separately. To ensure that my unit tests were done in isolation from the rest of the code I used the Katacoda Python Playground, with the Python 'unittest' package. [11] An example of one of my unit tests is shown in Figure 4.

Throughout my development process I used unit testing to ensure my code performed correctly and reliably. Each time a new component needed to be added I would first write a unit test to use for unit testing. Writing the test before the writing the code to be tested ensured that I had a good understanding of the expected behaviour of my code before implementing it, allowing it to be bound by the specification. Each component was only implemented once it reliably passed all the unit tests I had written for it.



```python
import unittest

# find an item in a list by id
def find(id, list):
    try:
        x = [x for x in list if x[0] == id][0]
        i = list.index(x)
        return list[i]
    except:
        return []

class Test(unittest.TestCase):
    def test_0_find_exists(self):
        print("Starting find_exists test")
        test_list = [[x,2*x] for x in range(3)]
        print("test_list: ")
        print(test_list)
        for x in range(3):
            print("Finding list item with id "+ str(x) +": ")
            testing_first_item = find(x, test_list)
            self.assertIsNotNone(testing_first_item)
            self.assertEqual(test_list[x], testing_first_item)
            print(testing_first_item)
        print("Finished find_exists test")

if __name__ == '__main__':
    unittest.main()
```

Figure 4: An example of a unit test for one of my functions

I had very little experience in unit testing when I began this project and learned a lot more about it during the process. Had I known at the start what I know now, I would have written all my unit tests in a single file in my project directory instead of doing them each individually using a temporary online sandbox. This would have been a lot easier to implement and run, giving me a record of the tests written and allowed me to quickly get an idea of the code coverage, rather than trying to do it manually. I believe that doing the unit tests in the way I chose to do them due to my lack of experience was not a good idea, and I am glad to have learned from this experience as it will allow me to use better testing practices in my future work.

5.2 Integration Testing

Unit testing is the process by which the performance of each feature of a system is evaluated. To unit test my application I created a large spreadsheet and used my development log of all the features I had implemented to list all the components of my software. I decided on a simple functionality test for each feature, detailing the required result for a successful test. Features that could not be evaluated with a simple test in this manner were broken down into smaller sub-features with their own tests to allow for a comprehensive evaluation of the system's components.

This testing process was done during every sprint in my development to keep with the Agile methodology of continuous iterative testing and development, allowing me to produce a working application at the end of each sprint and ensuring that bugs and errors within individual features did not propagate throughout the development process. At the end of development, all integration tests were passed.

5.3 AI Functionality

Since the AI menu planning functionality is the main focus of this project, I wanted to dedicate extra time to evaluating this specifically. I first generated 1000 random menus using the same random menu generation function used in my reinforcement learning process — the menus generated were therefore constrained by variety and concurrency parameters, but with no restrictions on the cost or preference score of the menu. This was done as a comparison to see how much of a difference my program's AI had on the menus generated. I plotted the results of this random generation on a scatter graph comparing cost and preference score, shown in Figure 5.

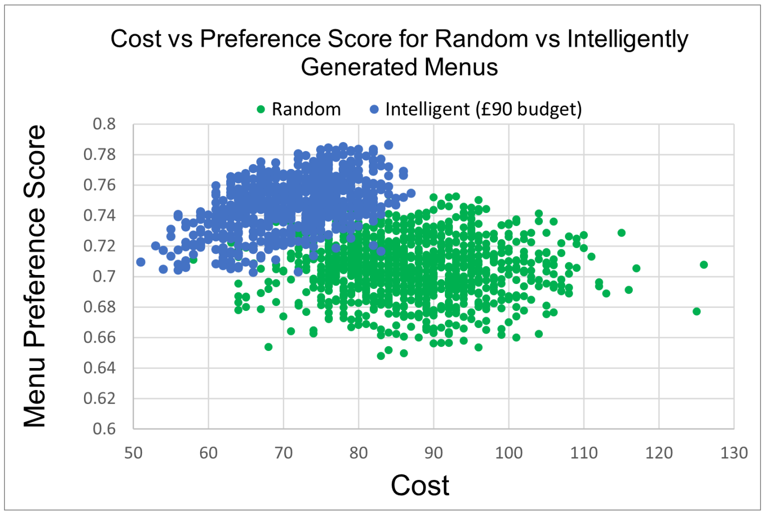


Figure 5: Cost vs preference score for intelligent (blue) and randomly generated (green) menus, with a budget of £90

My next step was to run the system 1000 times, this time using the menu's AI functionality to generate 'good' menus. I used a budget of £120 to start with. I plotted the results from this testing in the same scatter graph as the random data (shown in Figure 5) to compare the two datasets. The data clearly show a significant improvement in both cost and preference score for the menus generated intelligently, compared to the randomly generated menus. All the intelligently generated menus are above the average preference score (0.7) and below the menu's budget.

Running the system again using budgets of £90 and £70 produced similar results, although the as the budget became smaller, more menus were produced at lower preference scores compared with the results at higher budgets. This was expected behaviour, as reducing the budget gives less freedom of choice for selecting high-preference meals.

5.4 System Testing

Having passed unit testing and integration testing successfully, I decided to test the entire system's functionality by planning a real menu for my first catering event of 2022 using two methods — firstly using my usual manual method of menu planning, and the second using this application. This would be a good method of testing my applications suitability for its intended purpose, by comparing it to the existing alternative. I used the same scenario for both planning methods — 7 regular group members, 3 vegetarians, one vegan, and one person with an egg allergy. My budget was £120, and the menu plan was for a weekend, starting on Friday at dinnertime and ending on Sunday at lunchtime.

My process for the manual menu planning started off by making several tabs in an Excel spreadsheet — one for the meal schedule, one to list all the dietary requirements of group members, one for the ingredient quantities, one for the shopping list, one for the list of equipment and one for the timeplan. Choosing meals for each group to be catered for was fairly straightforward, although I had to think carefully about the menu's variety and double check the ingredients for certain recipes to ensure they were suitable for the group to which they had been assigned.

The ingredient quantity calculations were the most time-consuming part of the process, involving scaling up the ingredients in each recipe by the number of people to be consuming it, making a giant list of all these quantities, then combining all the duplicate entries. Excel formulae helped to speed this process up a little, but it still took just over half an hour.

To create the shopping list, I searched a supermarket grocery website for each ingredient name, worked out which of the products available would be the cheapest overall for the quantity I needed, then added the correct amount of the product to an online shopping basket. Once I had finished, it was clear that the ingredients I had selected meant that I was over budget for the event, so I needed to go back to the menu planning stage and choose a cheaper meal. This meant that I had to redo a lot of the previous planning stage and recalculate the ingredient quantities.

The second menu plan I chose ended up being within my budget for the menu, and so all that was left to do was the equipment list and timeplans. Working out a list of equipment to bring only took around 10 minutes, and involved looking at each meal in the menu, writing down the equipment needed for it from memory, and then refining the list to see which items could be used for multiple recipes in a mealtime simultaneously (e.g. knives, peelers, graters) and which would need to be designated for a single recipe (such as frying pans or baking trays).

To create the timeplan, I worked backwards from the serving time of each meal, writing down the steps in reverse order for each recipe. Once the steps were in a sensible order with approximate timeplans listed, I combined all the steps for each mealtime into a big master timeplan, showing all the steps that needed to be completed at each stage in the cooking process.

Overall, manually planning the menu took me approximately two hours, and I was happy with the meal schedule. In the past I have missed out ingredients and equipment due to human error when creating the lists, so double-checking these was important. The next step was to create the menu using my application and compare the processes.

I navigated to the Add Menu page, entered my requirements and clicked 'Save and Continue' to go to the menu generation step. The automatic menu generation took around 15 seconds, but I did not like the first menu it produced so I clicked it again to get a different one before moving on to the shopping list. Selecting my desired products took around 2 minutes, since the list was already auto-populated with the cheapest option and all I needed to do was switch some of the products to my preferred brands. The cost of the menu was below my budget, and the total automatically updated every time I changed a product so I could immediately see the effect of my changes.

When I was finished, I saved my shopping list and navigated to the "Download Menu Information" page to get my PDF file. The entire process took less than 6 minutes, which was already a vast improvement over the manual planning process. Additionally, I did not need to worry about missing out ingredients or equipment since it was all done for me by the application. I added some loading animations to the shopping list, menu editor and download pages to make the waiting experience a little more user friendly.

5.5 User Testing

To test the functionality of my application from a user's perspective, I asked for volunteers from my peers and colleagues to anonymously trial the system by using it to plan realistic menus for their households for a set period. I received 6 volunteers to trial the system, who provided basic anonymised information about their catering experience and the dietary requirements of their household members along with their evaluations of the application. This was done via Google Forms, with one form for volunteers to register interest in testing the system, and one form to give feedback.

Users liked the user interface for its "clean", "simple" and "minimalistic" design, although it was mentioned that it would be helpful to have the functionality to delete menus (this was simple enough to add so I did so immediately on receiving the feedback). The shopping list feature was particularly well received, with one person describing it as "amazing". Users liked the formatting, the total-price calculation, and the suggested cheapest products.

The testers rated certain features of the application with a score from 1 to 10, the results of which are shown in the chart in Figure 6. Overall the application scored well, with average scores being 8 or higher across more than half of the criteria and with all criteria scoring higher than 6. The criteria with the lowest user scores were the loading times, the accuracy, and the ingredient variety. I highlight the main issues.

The loading times were scored worse by users calculating menus for week-long periods, with the main complaint being that menu generation took a relatively long time — all other loading times, such as for shopping list generation, received no complaints. As the number of meals in a menu increases, the time taken to generate a suggested meal plan increases due to the increase in difficulty in finding a meal plan that suits the constraints. On average, every lunch or dinner added to a menu increases the loading time of the menu generation by approximately 4 seconds, assuming two separate groups are to be catered for. This means that a ‘weekend” menu will take approximately 16 seconds to generate, and a week-long menu will take approximately one minute. To improve these loading times further would be difficult, as it would likely involve decreasing the number of iterations in the learning algorithm which would reduce its accuracy. In [12], Balatsko suggests using the ‘vectorise’ function in the NumPy package to speed up Python machine learning programs. I tried this and managed to speed up the week-long generations by approximately 7 seconds which is an improvement, but I would not consider it to be a significant improvement.

The second issue found by the users was that the menu cost was occasionally higher than the budget. My AI functionality testing in Section 5.3 showed that this should not be the case, so I ran some extra tests to see where the error lay. The issue turned out to not be with the AI menu planner, but with a mismatch in how the estimated menu costs are calculated vs how the total price in the shopping list section is calculated. The AI learning process uses an average cost per unit to estimate the cost of its recipes, meaning that each recipe is priced according to the exact amount of ingredient used. The shopping list on the other hand is based on selecting real products for each ingredient, meaning that in most cases there will be a slight unavoidable excess of ingredient purchased, due to packaging sizes. The shopping list is set up to minimise this excess, but there will still inevitably be a mismatch between the two costs. There are two potential solutions to this issue. The first would be to save the contents quantity of the product used to get the average unit price of each ingredient when the average costs are updated and ensure that the cost is estimated based on purchasing whole units only. I attempted this solution and found that this made the estimated total menu cost much higher than the actual cost since products with the cheapest unit prices often have the largest content quantity (since ‘buying in bulk’ is cheapest). This affected the performance of the menu planner since menus that would have been within budget were no longer being considered.

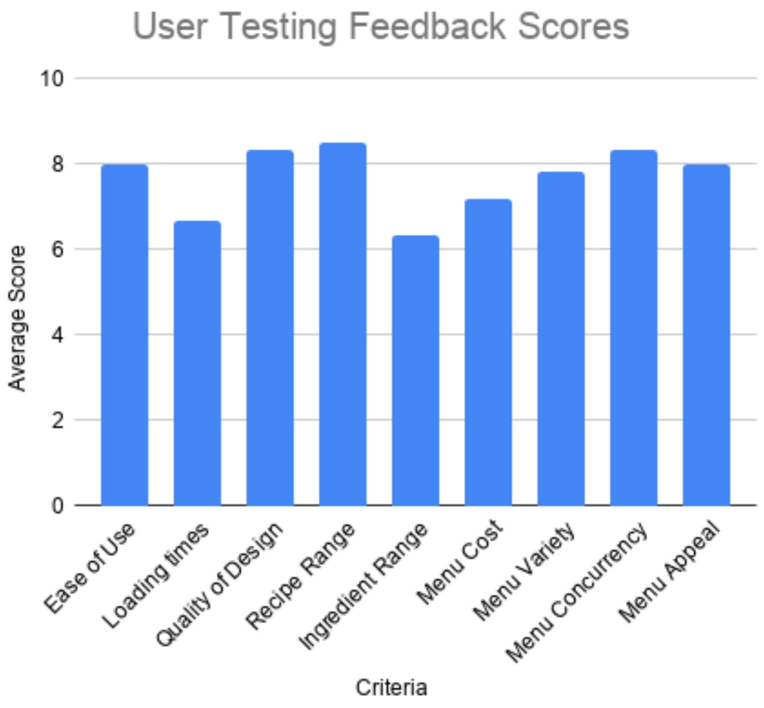


Figure 6: Average feedback scores from the user testing

The other solution would have been to use the Tesco Search API to get a list of products for each ingredient so that the learning algorithm could use the most accurate cost estimate possible. This would be the solution I would implement as it would effectively solve the issue, however it would also mean making approximately 600 calls to the API every time the menu editor page is loaded to account for every ingredient that might appear in a menu. The reason for not implementing this is that when I was granted access to the API, I was specifically asked to be considerate in my use of resources. It was a courtesy on Tesco’s behalf to allow access to the API in the first place and I would not want to take advantage of this. The menu generation functionality still works as expected — menus generated are below budget based on the calculated cost of the menu using all the resources available to it, and this is merely an issue of resources. If I had unlimited access to the API then this would be my proposed solution, and I am satisfied that it would completely solve the issue.

6 Results and Evaluation

6.1 Final Product Description

The final product is a Python-based web application, written using the Flask framework, containing a database of approximately 600 ingredients, 34 recipes (including details on ingredients, instructions and equipment needed), 14 dietary requirements (and their respective restricted ingredients), and 30 pieces of common kitchen equipment. The application allows users to register, login, and create new menu plans based on their budget, event timings, their group's size, and dietary requirements. Users may use the default recipes that are pre-loaded into the application or may create their own recipes by following the suggested format to allow for compatibility with the menu-planning features of the program. Menus can be created manually, or the user can make use of the application's artificial intelligence capabilities and allow it to generate a suitable menu to fit their requirements. Menus generated using the AI functionality will fit the user's budget (in terms of the actual cost of ingredients used) and will have higher average preference scores than menus generated randomly. Once a menu has been generated, users can view and edit a shopping list to select suitable products for each ingredient necessary for their menu. They can view timeplans for each mealtime and download an information file containing all relevant information for their menu. This can be printed off to assist with catering.

6.2 Departures from Original Requirements

Loss of access to the Tesco Product Information API meant that I was unable to implement one of the features I had planned to assist with non-functional requirement NF4 — "The meals chosen for groups with dietary restrictions must comply with those restrictions". Sometimes an ingredient can comply with a dietary requirement in theory, but certain products for that ingredient may contain ingredients that are unsuitable. An example of this would be barbecue sauce — in theory, it is suitable for mustard allergy sufferers, but certain brands of barbecue sauce may use mustard in their product for flavouring.

The intended feature would have been part of the Shopping List section and would have ensured that products suggested for an ingredient did not contain any ingredients that were unsuitable for any of the group members who would be consuming a dish containing that ingredient. In lieu of this, I implemented an alternative feature to highlight ingredients in a menu's shopping list that are part of a meal to be consumed by those with restricted diets. The user would then be advised to manually check the products that they select to ensure their suitability. This seemed to be the cleanest and most appropriate solution.

Aside from this issue, there have been no other departures from the original requirements. All functional and non-functional requirements outlined in Section 2.1 have been implemented fully.

6.3 Fitness for Purpose

The first indicator of my project's fitness for purpose comes from the objectives outlined in Section 1.2, the first of which being to **"Use artificial intelligence to create a menu planner that suggests a suitable meal plan based on user inputs"** — the testing in Section 5.4 shows that this objective has been achieved. If we define a 'suitable' menu to be a menu that fits the variety, concurrency and cost requirements set out in Section 2.3, then all menus produced by the menu generator are suitable menus since every menu produced fits these criteria. Additionally, the AI testing showed that the menus produced by the menu planner had significantly higher preference scores than menus generated randomly, and also showed that the fact that the planner consistently generated menus that were below the user's budget could not be attributed to randomness. It is clear that the menu planner has been successfully implemented in relation to its evaluation criteria.

The second objective was that the application should **"Automatically generate an accurate timeplan, equipment list and shopping list for menus in the application"**. For the questions asking users to rate how useful they found these three features, my test users gave average scores of

8.83 for the timeplan, 6.33 for the equipment list and 9 for the shopping list. All three features were implemented successfully, passing their integration tests and all associated unit tests.

The user testing feedback showed that these features were well received by the test users, with the shopping list and timeplan deemed to be particularly useful. The equipment list scored lower on the usefulness question as the users were catering in their own homes and therefore had no need for an equipment list. Despite this, they still found it to be well implemented and accurate. When I tested the system in an event catering context as opposed to a home catering context, I found it to be comprehensive and accurate, and a valuable addition to the menu planning download file. The automatic generation prevents human error from causing items to be forgotten which is a valid concern in event catering.

The third objective was to **"Create a process to suggest suitable products for each ingredient in a menu"** This objective was achieved via the Shopping List section of the application. The user testing in Section 5.5 highlighted the effectiveness to which this has been implemented, with users scoring the feature an average of 8.67 out of 10 in answer to, "Does the application suggest suitable products for ingredients in the shopping list?". Several users mentioned that the reminders to check product suitability for certain diets were useful. Unfortunately, not all of the desired functionality for the shopping list could be implemented due to the change in access to the API that I covered in Section 4.3 but before that situation arose, I had successfully implemented the functionality wherein the shopping list filtered out products that were unsuitable for people who were consuming them and had dietary requirements. This functionality had passed integration testing successfully. My implementation of an alternative feature was well received and deemed to be useful by the test users, so I believe that overall this was still successful.

6.4 Performance

My project's overall fitness for purpose can also be evaluated via its performance in relation to the functional and non-functional requirements defined in Section 2.1. Functional requirements F1 through to F13 have all been checked via the combination of unit testing and integration testing performed throughout the development of my project, and therefore have all been satisfied, so I will concentrate on the non-functional requirements here, starting with the ones that have definitely been satisfied.

The application makes use of password hashing to ensure password security. Users are unable to access the application without logging in, and a user's status is checked on every page to verify this. Users are also unable to view the menus and recipes of any user other than themselves — this is also verified on every relevant page of the application. Verification is done on the back end of the application to prevent malicious tampering from users, and all queries to the database are done via stored procedures to guard against injection attacks. For these reasons, requirement NF5 ("user account data should be stored securely in the database") has been satisfied.

The decision to change the variety from being a preference to a hard constraint meant that the menus produced by the application were guaranteed a minimum level of variety. This level is controllable by the user. This, combined with the fact that the test users scored the menu variety of the application at 7.83 out of 10 in the feedback survey shows that requirement NF2 ("the menu should contain a reasonable level of variety") is satisfied.

Requirement NF3 states that the user interface should be intuitive and easy to use. Careful consideration was given to the layout and design of the application throughout its development. Users ranked the application at 8 out of 10 in the 'Ease of Use' category and 8.5 in the 'Quality of Design' category — I consider this to be sufficient to mark this requirement as satisfied.

Requirement NF4 states that "the meals chosen for groups with dietary restrictions must comply with those restrictions". Since the only meals suggested for each group are the ones containing no ingredients that group cannot consume, this requirement is satisfied. The restricted ingredients for each diet are saved in the database and combined with any specific allergies associated with group members.

The performance for requirement NF6, "the software must behave predictably and appropriately for its given function", is satisfied for the most part. The application's features have all been implemented correctly and each feature performs appropriately in its own scope. The main detriment to this requirement is that while the menu planner estimates the cost of menus based on the actual ingredients used, the shopping list calculates its total cost based on the full product purposes which is often higher than the cost of the ingredients used due to not being able to purchase the exact quantity of ingredient needed. Unfortunately, this means that occasionally the actual cost is higher than the user's budget. This is not an issue with the AI functionality, but an issue in obtaining accurate menu costs, as the learning algorithm performs well in the context of the information available to it. An ideal solution would be to use the same process used to generate the total cost in the shopping list, but this would involve putting an inconsiderate strain on the resources that were extended to me as a courtesy, and so unfortunately would not be a viable option. Due to this, I would deem this requirement mostly satisfied, with the potential to be fully satisfied if I had unlimited access to this API.

Requirement NF1, "the software should produce its suggested meal schedule within a suitable timeframe", was the requirement with the most room for improvement. The application produces results reasonably quickly for weekend-long events, which are the main event type present in event catering. Week-long events took significantly longer to generate menus for and received lower scores in the user testing as a result, but with an average overall score of 6.67 for the loading times it appears that even the loading time for these events were still within a somewhat acceptable range. No test users produced menus for events lasting more than one week. I believe that this requirement is partially satisfied — loading times are acceptable for weekend events, are acceptable for some users for week-long events, and reduce in acceptability as the length of events grows beyond the one-week point. The application would benefit greatly from better optimisation of the learning algorithm to fully satisfy this requirement.

6.5 Scopes

The project was originally designed with event catering in mind, as outlined in Section 1. The functional testing in Section 5.4 showed that it was suitable for use in this scope, providing an efficient way to plan menus for event catering and a significant improvement on the manual planning process. However, the testing in Section 5.5 also indicated an additional scope that I had not considered when starting this project, showing that the application was also an effective tool to assist with family meal planning. While the timeplan and equipment list may not be particularly applicable to this scope, the testers found the shopping list functionality to be "incredibly useful" and "really helpful", and the ability to easily select meals for the menu plan helped with the decision-making process. The application could also be used for more large-scale catering, such as in military and hospital applications, or for schools wishing to create menus for their canteens. However, since schools usually only provide lunches and therefore do not require the full menu of meals. Adding a feature to allow users to select which mealtimes they would like to include in their menu would allow for the application to quite easily be used in this scope.

6.6 Limitations

One limitation of my application is that there are dietary requirements that it would be difficult to incorporate into a meal plan using this program. For example, members of the Jewish faith often keep a Kosher diet, which has strict rules for the separation of meat and dairy products [13] which extend to the utensils used to prepare food. A separate, designated set of utensils is required for each of meat and dairy. [14] This is not always possible in event catering due to location limitations. Additionally, the Kosher dietary rules dictate that meat and dairy products may not be mixed within the same dish — my application does not currently have the functionality to implement this as a restriction. However, as this project is personally relevant to me, I fully intend to continue working on it after the submission deadline, and this is one of the features that I would be most interested to implement — from a perspective of accessibility, but also as an interesting challenge.

Another potential limitation is the scalability of the application. Within the application, certain aspects have the potential to limit scalability. Firstly, when calculating which meals are suitable for which group members, a data frame object is created — each column represents a group member, with the main group being represented as a single row. Rows are indexed by recipe name, and each cell of a data frame contains a Boolean value communicating whether the column's group member can eat that row's meal. As data frames do not have specific row or column limits, the memory of the system running the application becomes the main limitation to scalability. All data for the application is stored in a MySQL database which will also influence scalability, but this data is also more limited by system memory than by an arbitrary limit. There is a row limit of 4.2 million and an overall database size limit of 256 terabytes [15], which should be sufficient for almost all menu planning applications.

The more likely limitation to scalability is that as the length of the menu (and therefore the number of mealtimes in the menu) increases, so does the time taken to generate the suggested menus using the AI functionality. The wait time is perfectly fine for weekend events, but the application takes approximately a minute to generate simple week-long menus and increasing the event length or using more complex parameters would increase this even further. Due to this, it appears that the application works best for shorter time periods shorter (e.g. 1-2 weeks, preferably 1 week). This is often the way that long-term catering menus are structured anyway, having a repeating 'Week A/Week B' schedule, so this would likely be an acceptable limitation from an end-user point of view. However, this is purely subjective — some users may be perfectly happy to wait the extra few minutes to generate longer menus.

7 Critical Assessment

The primary objective in this project was to create a better system for menu planning that will allow a user to generate a suitable menu based on cost, variety, dietary requirements, and preference. I chose this project because as a caterer myself I am aware of how time-consuming and complicated manual menu planning can be. After a review of literature on automated planning as applicable to menu planning, I chose a reinforcement learning approach to tackle this problem and used Python for implementation.

Python was a good choice to use to develop this project, as it is a straightforward, powerful, and flexible language. It works well for web development and for machine learning, the core technologies of my project, and allows access to a wide range of useful packages. I coded my machine learning algorithm from scratch, but if I were to do this project again, I think I would take a different approach and make use of one of Python's packages for machine learning. I chose to do this project without a package as I wanted the opportunity to explore and thoroughly understand how machine learning worked and see the principles behind it in the most direct way possible, and I believe it was a valuable experience. However, from a functional point of view, purpose-built packages for machine learning have the potential to improve the performance and usefulness of my application, a valuable consideration since I have designed this application as a tool to use in my own real-world scenarios. One of the limitations of my project was its scalability in terms of the time taken to generate menus, especially for longer events, and using packages could be a good start in optimising my application to mitigate this limitation.

At each stage of development, the agile methodology was followed, a working piece of software was produced, and unit/integration testing carried out. As I mentioned in Section 5.1, I would have benefited from a little more experience in unit testing before beginning this project. Writing all the tests to a file within my application so that they are all together would have been the correct way to approach the unit testing, and I have gained valuable experience from this that I will carry forward into my future work.

My development was organised and procedural, which I feel contributed greatly to the overall success of my project. I used a spreadsheet-based Kanban board and development log, which made it easy to keep track of features and deadlines. The timeplan I set out in my original Project Statement and Plan document was realistic and gave me a good starting point to set manageable, effective development goals for each sprint. Due to this I was able to manage my time effectively and avoid any last-minute rush.

The most time consuming and least efficient part of the development process was testing the AI functionality of the menu planner due to the amount of tweaking of parameters and plotting/comparing charts that it required. If I were to do this project again, I would like to find a way to automate this by writing helper functions to simulate and plot the effects of changing every parameter in combination with each other. Not only would this vastly improve the efficiency of this stage of the process, but it would also be truly interesting to be able to view and analyse all that data to see the effects and it would provide useful insight into the mechanics of machine learning.

The application that I developed definitely has room for improvement in two obvious areas, the first of which being the optimisation of the menu generation algorithm to improve the time taken, and the second being to find a more accurate way to estimate menu costs within the learning algorithm so that the price calculated for the final menu is the same as the price generated during the shopping list stage. The first improvement requires further research into the field of reinforcement learning, whereas the second will likely require greater access to resources. I have emailed the Tesco Developer team to enquire about this and am currently awaiting their reply.

Despite the areas I could have approached differently and a couple of areas for improvement, I believe that overall the project was a success. I have created a useful tool that satisfies the objectives set out at the start of my project and solves the issue that motivated me to choose this topic in the first place, gained experience in a range of software development techniques, and learned a great deal about the fascinating field of artificial intelligence.

8 Conclusion & Future Work

The aim of this project was to explore the applicability of automated planning to the context of menu planning, and to improve the efficiency of the menu planning process by creating an automated menu planner that uses artificial intelligence to create a suitable menu, given a set of user-defined constraints. I researched relevant topics in my literature review and planned a sensible, effective approach to the problem, designing a suitable system and bringing it to fruition through well-structured implementation and testing. The final product provides a good solution to my original problem, and overall a good alternative to the task of manual menu planning has been developed.

The testing carried out in section 5.3 shows that the AI functionality works well for its intended purpose and performs significantly better than random generation of menus. By evaluating the menus produced by their preference scores and costs it is clear that the menus produced are of good quality, and this viewpoint is supported by both the system testing in Section 5.4 and the user testing in Section 5.5.

This project was personally relevant to me, and I will be getting plenty of use out of the application now that it is functional. Since it is such a passion project, I fully intend to continue working on it after graduation as I believe it has a lot of potential. Currently, the application is only accessible from the locally hosted production server. To achieve its full potential, a suitable method of deployment is necessary. This would firstly involve gaining full, unlimited access to the Tesco API range, as my limited access was the root cause of several limitations in the project's functionality. This would also allow me to improve the meal cost estimations in used in the learning algorithm. If permission is granted, the next step would be to choose a method of deployment — the application is written in the Flask web framework, which would make it perfectly suited to deployment onto a server. Alternatively, I have come across a few Python packages, such as the Widdershin package [16], which can be used to convert Flask applications into desktop applications. I think that this would be an interesting area to explore.

Additionally, there are several minor features that I would like to add to Tagine, one of which being the Kosher functionality I mentioned in Section 6.6. I would also like to see the full extent to which I can improve the efficiency and performance of the application and will be looking into the best ways in which to do this. I believe I will likely be 'tinkering' with this project for a long time.

I have thoroughly enjoyed working on this project, and through it have gained a useful tool to use in my catering work, in addition to a wealth of experience in software development that I look forward to using in my future career. Overall, I consider this project to have been a success.

References

[1] A. K. Pani and G. P. Bhattacharjee, "Temporal Representation and Reasoning in Artificial Intelligence: A Review," *Mathematical and Computer Modelling: An International Journal*, vol. 34, no. 1-2, p. 55–80, Jul 2001.

[2] M. Das, M. R. Islam, J. R. Doppa, D. Roth, and S. Natarajan, "Active preference elicitation for planning," Feb 2017.

[3] T. M. Mitchell, *Machine Learning*. McGraw-Hill, 1997.

[4] J. A. Hendler, A. Tate, and M. Drummond, "AI Planning: Systems and Techniques," *AI Magazine*, vol. 11, no. 2, pp. 61–78, Jun 1990.

[5] F. Mohr, M. Wever, and E. Hüllermeier, "ML-Plan: Automated Machine Learning via Hierarchical Planning," *Machine Learning*, vol. 107, no. 8, pp. 1495–1515, Jul 2018.

[6] A. Kumar, S. Yagati, M. S. Tomov, S. J. Gershman, and W. Yang, "Reward Generalization and Reward-Based Hierarchy Discovery for Planning," *PLoS Computational Biology*, Apr 2020. [Online]. Available: https://journals.plos.org/ploscompbiol/article?id=10.1371/journal.pcbi.1007594

[7] S. Jiménez, T. De La Rosa, S. Fernández, F. Fernández, and D. Borrajo, "A Review of Machine Learning for Automated Planning," *The Knowledge Engineering Review*, vol. 27, no. 4, p. 433–467, Dec 2012.

[8] V. François-Lavet, P. Henderson, R. Islam, M. G. Bellemare, and J. Pineau, "An Introduction to Deep Reinforcement Learning," *Foundations and Trends in Machine Learning*, vol. 11, Nov 2018.

[9] R. T. Baust, "Study of Computer Procedures for Menu Planning and Recipe Service for DoD Elements." DoD, Jul 1967.

[10] J. E. Kelley and M. R. Walker, "Critical-path planning and scheduling," ser. IRE-AIEE-ACM '59 (Eastern). New York, NY, USA: Association for Computing Machinery, 1959, p. 160–173.

[11] "Python playground." [Online]. Available: https://www.katacoda.com/courses/python/playground

[12] M. Balatsko, "4 easy steps to improve your machine learning code performance," May 2019, accessed on April 20, 2021. [Online]. Available: https://towardsdatascience.com/4-easy-steps-to-improve-your-machine-learning-code-performance-88a0b0eeffa8

[13] A. Cassoobhoy, "What is kosher food?" Jul 2020. [Online]. Available: https://www.webmd.com/food-recipes/kosher-food

[14] R. Schwartz, "Keeping a kosher kitchen." [Online]. Available: https://www.kosher.com/learn/laws-of-meat-and-milk/keeping-a-kosher-kitchen

[15] N. Duncan, "Maximum mysql database size?" Feb 2020, accessed on April 20, 2021. [Online]. Available: http://nickduncan.co.za/maximum-mysql-database-size/

[16] N. Johnstone, "Widdershin/flask-desktop," Sep 2013, accessed on April 20, 2021. [Online]. Available: https://github.com/Widdershin/flask-desktop