PLYMOUTH COLLEGE of ART

Making Futures

Cooking up a Remix:

Investigating the use of Max MSP graphical programming environment to develop a new software design

By Chris Watson, Sheffield Hallam University and Michael Dunn, Doncaster University Centre

Abstract

This paper presents our approach to the development of Food Monitor, an original software program that provides a database of grocery items that can be used to monitor stock levels, suggest recipes, and provide nutritional information to the user. The software is intended to provide a low-cost, flexible and useful tool for the everyday consumer to assist the universal management of grocery-related activities in a residential setting, with a focus on increasing awareness of nutritional information to encourage healthy-eating, reducing food waste, and offering interactive recipe choices. The software was developed in the Max MSP graphical programming environment, which despite primarily being a domain for computer musicians and visual artists nevertheless offered a flexible set of tools to achieve the aforementioned aims and intentions. To this end, here we reflect not only on the development of the Food Monitor project itself, but more-so on the opportunities facilitated by flexible programming environments such as Max MSP to non-expert programmers.

Keywords

Max MSP, software design, remix

Motivation

The UK has observed deterioration in quality of diet over the last five to ten years, whilst levels of obesity and associated health problems have also increased across the same period. National statistics (Health and Social Care Information Centre, 2014) illustrate that the proportion of adults classed as obese has increased by 11.2 per cent (men) and 8.5 per cent (women) over the period 1993 – 2012. Furthermore, household purchases of fruit and vegetables have fallen by 16 per cent and 6.1 per cent respectively since the mid 2000s, with less than 33 per cent of the UK found to be consuming the recommended five or more portions per day (as of 2011), whilst intake of saturated fats and non-milk extrinsic sugars (NMES) were both found to be in excess of the recommended levels. Despite these statistics, over 87 per cent of the population considered their own diets to be quite healthy or very healthy, suggesting a lack of congruence between individuals' perception of 'healthy eating' and the reality of the habits in which they engage.

These figures seem to suggest that UK consumer habits and awareness of nutritional implications of certain food and drinks would benefit from improvement, particularly in the light of recent debates over proposed taxes on alcohol and sugar for similar reasons. Furthermore, official figures (Department for Environment Food and Rural Affairs 2015) show that around 15 million tonnes of food is wasted each year in the UK, the most of all European countries; these statistics also suggest that whilst manufacturers, retailers and the service industries account for a combined 53 per cent of the UK food waste, the remaining 47 per cent is attributed to individual households, the biggest single contributor of all. Therefore, we submit that perhaps the most effective way to reduce the amount of food wasted is by encouraging changes in consumer habits.

Finally, we also identified the potential to design an effective consumer system that would augment the ways in which households are able to interact with food and drink in a typical residential environment; by this we refer to an interface that could be used to query a database of grocery items available in the house, suggest recipe ideas based on available ingredients, and to guide individuals to prepare meals through set instructions. With these points in mind, this project aimed to develop a software system that could respond to these issues by taking advantage of flexible programming tools to design and implement a bespoke program titled Food Monitor that could be implemented into the home consumer environment.

Related work

Having established our aims at the outset, it seemed appropriate to review any existing systems that were available. We found a number of commercial software examples – such as MasterCook, Living Cookbook, and Shop'NCook – that provided effective functionality in the context of the aims outlined in the previous section of this paper, including: recipe databases, nutrition databases, ingredient search, verbal cooking instructions, and shopping list generator, amongst others. Other apps, such as Sugar Smart from the NHS's Change for Life incentive, offer easy-to-use tools with a focus on encouraging healthier eating and increasing awareness of nutritional information. With this in mind, the focus of the project shifted somewhat from the original intention to create a completely innovative tool in response to the initial motivations, to instead explore the potential of using Max MSP as an environment to develop an effective program in the chosen context.

Nevertheless, the opportunity to implement our own original features remained, for example a stock control database system that would automatically update as the user purchased and consumed grocery items, and a best-before-end (BBE) warning system that would inform the user of impending use-by dates as a reminder to use specific food items to avoid waste; these features were not prevalent in the existing software products that were reviewed, and thus were seen as valid points for potential development.

Programming environment

We proposed the development of an original piece of software, the synthesis of which would be undertaken in the graphical programming environment Max MSP from Cycling '74. Since we were not programmers in the traditional sense, with no real experience of using script-based languages such as C or Java, we were keen to examine the potential of graphical programming to provide alternative means of constructing a robust and functional piece of software.

Programs such as Max MSP offer relatively easy access – in comparison with traditional programming languages – to working in a programming environment, making use of graphical objects that are connected in a blank workspace using 'drag and drop' commands.

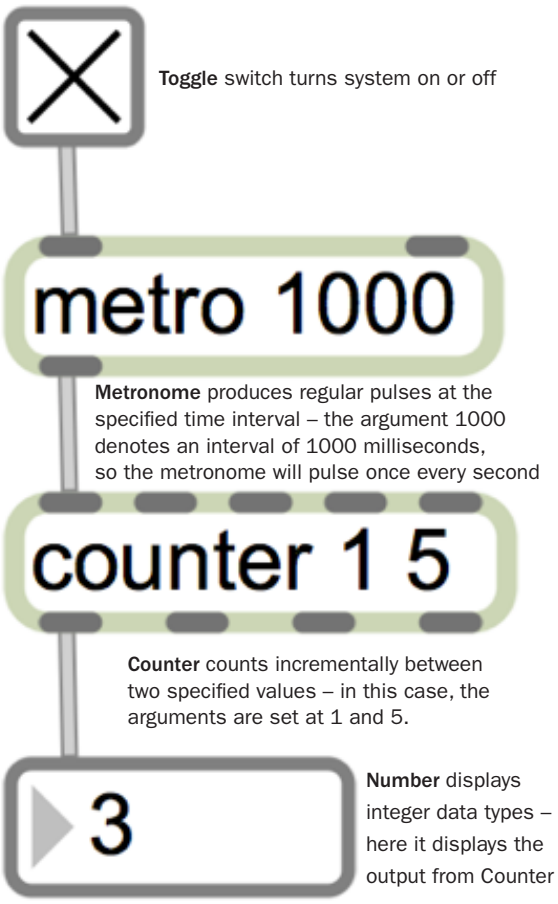

Fig.1 – Example of graphical programming in Max MSP

The programmer joins individual objects together using patch cords to make node-to-node connections, and specifies arguments to denote how each object should function. To illustrate, an annotated example is presented in Fig.1.

This particular programming environment was originally designed for musicians, however gradually over time it has become an equally powerful tool for those working with video, performing artists, interactive installations and so on. A few examples of the application of Max MSP that illustrate its versatility include audio-visual environments for the rehabilitation of children with special needs (Azeredo 2007), exploring the use of dance as a form of social interaction through a rhythm-based robotic system (Michalowski et al. 2007), and D-Jogger that uses a 'step detection algorithm' to select different pieces of music whose tempi sync with the user's walking pace (Moens et al. 2010).

Fundamentally, the software enables the programmer to work with a wide range of data, and this flexibility, along with the relative ease of programming with it, is what we present here as being of particular value as we reflect on the methodological aspects of this particular project.

One other important point to note is that this programming environment is 'open', in the sense that it can be connected from external sources and connected to destinations using common communications protocols such as MIDI, OSC and Serial. Therefore, compatibility with increasingly affordable programmable microcomputers such as Arduino – as used in this project – and Raspberry Pi means that more complex systems can be developed that make use of hardware and software, with further potential to include the use of sensors, interfaces, actuators and integration with other pre-existing software. Furthermore, with increasing communities of practitioners from a range of diverse backgrounds, there is a substantial support network available – primarily online – to question, offer assistance and help troubleshoot.

This infrastructure of systems and support form what Lev Manovich (2013) terms 'new media' in his book Software Takes Command, something that will be explored further from the taxonomy he devises. The software and hardware are important tools that support future innovations to be explored across different disciplines, much in line with the interdisciplinary aims of the Institute of Making. Given the climate of 'big data processing', the visualisation of data sets from different perspectives offers the potential for new connections to be made that enable us to see the bigger picture.

System Design

The intention at the outset was to utilise an Arduino microcontroller to support a sensor capable of feeding barcode data from food and drink products to the bespoke software built with Max MSP. The software would support a database into which the barcodes would be added and indexed, thus building a digital archive of food and drink items purchased by the user. Quantities of each product would be tracked using the barcode scan system, so that when the user purchases or consumes items, the quantities in the database would be amended accordingly.

Each indexed grocery item would also be cross-referenced with an existing database of nutritional information pertaining to a wide range of common grocery items (we used the USDA National Nutrient Database for Standard Reference, see U.S. Department of Agriculture, Agricultural Research Service, 2014), thereby adding nutritional information to the specific user database of grocery items.

Subsequently, as the user consumes combinations of grocery items, the software is able to provide nutritional information for each meal, whilst also maintain a running log per week or per month which could then monitor salt, sugar and fat intake, for example. The database could also be queried to present meal options based on items in stock, with further potential to inform the user where to locate ingredients, for example fridge, freezer, cupboards and so on.

Remixing and Cooking

The projects mentioned earlier in this paper (see Azeredo, 2007; Michalowski et al. 2007; Moens et al. 2010) illustrate the versatile and innovative ways in which the Max MSP software can be used, from movement rehabilitation and music therapy, to artificial intelligence, robotics, and musical recognition, analysis and response systems. There are a plethora of other examples, each illustrating a unique idea realised through, or with the support of, this programming environment.

Whilst the fundamental building blocks or 'objects' within the software are the same, each idea will proceed to integrate these objects in different combinations, in much the same way as a language has a repository of words from which we create different sentences to communicate various ideas; in this sense, we regard Max MSP to offer a certain 'remix' functionality.

Furthermore, the notion of remixing – through sharing, reverse-engineering, and re-purposing 'code' – is a significant part of the wider community of practitioners that work with Max MSP, supported in particular through the Cycling '74 forum. Not only are patches shared, but in many cases other users will offer solutions or ideas as working prototypes to aid in their communication for another user to implement and test within their projects. This free repository of code is constantly being appended, and thus an ever-growing archive of source material is being collated that can be searched, retrieved and implemented by other users developing their next Max MSP projects.

Let us now consider the practice of remix further through Lev Manovich's definition of the term:

'we can define remix in this way: a composition that consists of previously
existing parts assembled, which is edited to create particular aesthetic, semantic, and/or bodily effects' (Manovich 2015:142)

In our scenario, we regard the assembly of existing parts to be the organisation of different objects together into a larger system, which we might think of as the composition. The design of this system will have a particular function, or effect, in mind; in our example the desired functionality is to log grocery items, review nutritional information, suggest recipes and so on. Importantly, Manovich differentiates between the notion of 'selection' and 'remix', with that of remix involving more than simply choosing (i.e. selecting) parts, but to also implement some level of creative application of the elements that have been selected, or something defined as the 'effect produced by the arrangement of parts that are present' (ibid.).

This is equally apparent whether we consider the assembly of individual objects from the program into a new patch or whether we consider the practice of integrating larger chunks of existing code; either way, the defining feature to fulfil the practice of remix would be to implement for a new purpose. The analogy of selecting ingredients and cooking with them is fittingly, quite apt, in the sense that the selection of ingredients from a supermarket may be differentiated from the practice of arranging these ingredients into a completed recipe; in short, we differentiate between the practices of going shopping, and being able to cook.

One other aspect of remixing that we engaged with in this project involved the databases. The primary database stored information on familiar grocery items and referenced the aforementioned USDA database to acquire nutritional information. This was enhanced with other metadata including barcode, price and a corresponding photo, with each grocery item assigned a separate entry in the stock database. An example of one entry would look something like on the next page:

Entry	Barcode	Name	Price	Fat	Sats	Salt	Carb	Sugar
3	47393	Cheese Gouda	1.49	27	18	2.4	2.2	2.2

Fig.2 – Example of stock database entry

As the user scans the item in or out, the system triggers this information from the database, along with a corresponding image of the item. We then proceeded to create other sub-databases that would sort and group the grocery items based on certain criteria; this included whether the item was in stock or not, which recipes it was used in, and recommended recipes for the user to try. This cross-pollination of data sources facilitated a more complex series of operations that really established much of the fundamental architecture of the system, and thus it represents a more fluid remixing of the data since it may be applied in different ways for different purposes.

Furthermore, this fluid system of data assimilation offers augmented functionality through the ability for the user to sort lists by variable. So for example, one might sort the recipe database by price ascending, to bring the most affordable options to the head of the list, or one might sort the recommended recipe database by least sugar to browse low-sugar recipes for which ingredients are in stock. This illustrates a malleable system that enables the user to remix the data for their own purposes, and also establishes the fundamental architecture of the software design.

Abstractions

Something that supports the remix culture further within Max MSP is the capacity to group objects together into modular form; this is referred to as encapsulation into subpatchers. Taking the earlier section of code as an example, let us suppose we would like to compile this series of connected objects as a single process. Suppose we then wanted to reintroduce this function elsewhere in the patch, or to share with others to enable them to remix into their own ideas. The ability to group this function as a single module would enable us to do both these things easily. Essentially, the series of objects are placed into a subpatch, where they continue to implement their commands and processes; however this subpatch could now be treated as a single, modular unit, rendering it more pliable for sharing, reuse and remix.

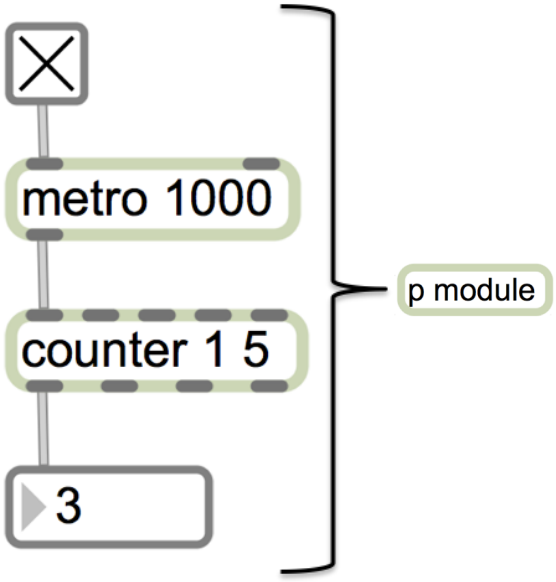

Fig.3 – Encapsulation into Subpatcher: the series of connected objects (left hand side) can be grouped as a single object (right hand side) and treated as a single modular unit. Although we cannot see the individual objects any longer, they continue to 'run' within the subpatcher.

The philosophy of encapsulation can be taken further, with users able to save their subpatchers as new objects in the library, alongside all the other standard objects supplied in the program. It is due to this functionality that a wide array of external objects have been developed and shared amongst the Max MSP community, with some popular and/or significant new objects actually being incorporated into subsequent official releases of Max MSP. There are even examples of users that have developed their own modular software within the Max MSP environment, that has then been released as a standalone system, such as VPT or Jamoma.

Even in the scope of this relatively basic current project, there is potential to remix and reuse parts of our programming for different purposes; for instance, the Food Monitor software makes use of database storage and organisation to enable the user to monitor stock levels, search for healthy recipes, track nutritional information and so on. The databases are reliant on the Coll object drawing data from multiple text files stored locally on the computer, and displaying images of food items and recipes using the Fpic object. However, the code that we have developed could easily be remixed for other applications; for example, to check employee personnel details whilst looking at the corresponding profile images, or to listen to a variety of audio recordings whilst looking at the corresponding spectrograms. In this sense, we might point towards the potential of transposition as an approach to remixing – that is, to consider the application of constructs and frameworks in different contexts.

Problem solving

It is common for the development process in Max MSP to involve a significant amount of time spent problem-solving in order to enable the system and its constituent parts to function as intended. Whilst a certain level of programming fluency does develop over time, significant obstacles will always present themselves which require efficient and robust solutions to be developed. To this end, we might also consider the idea of remix in the context of problem-solving, since solutions can often be re-used or transposed across different contexts. This suggests a reciprocity between reproductive thinking - i.e. use of known strategies - and productive thinking – i.e. application of strategies in a new way, both of which are recognised as part of problem-solving behaviour, for example by the Gestalt school of psychology.
Through Manovich we have established the importance of the application of existing content for a new purpose, and in this sense we have observed a significant amount of remixing and transposition to have taken place in the development of the Food Monitor software as solutions were taken and re-applied in different contexts and for different purposes. Some solutions originated from past experience working on other projects, whilst some solutions were transposed from help files and through analysis of other patches.

One relatively simple example is the use of the delay object, which receives a trigger, but delays its output by a specified amount of time. There are occasions on which one has to await the completion of one event before the next action is able to be carried out, and sometimes this can take time (albeit only fractions of a second). In previous projects, we have made use of the delay object to ensure that the secondary event does not occur before the initial event, by simply specifiying a time delay. In the design of Food Monitor, we encountered a similar scenario, where we required the dec message – i.e. to decrease the counter by a value of one – to await the prior completion of an update message first.

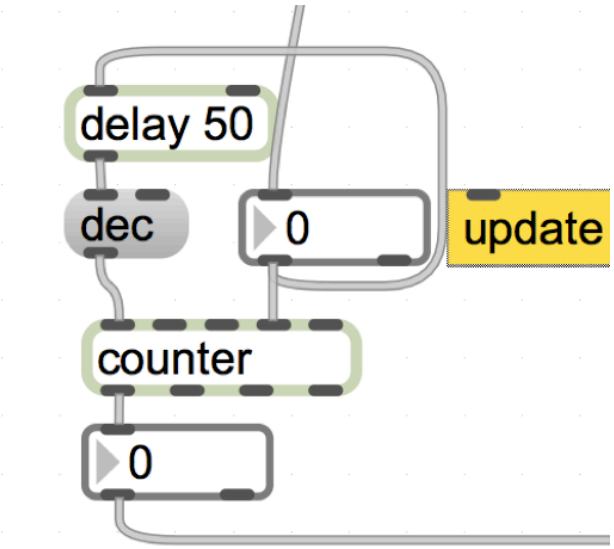

Fig.4 – Use of the delay object to facilitate correct order of event processing – transposition of a previous solution to a new context

In this example, we used the delay object with a time value of 50ms to overcome this problem, thereby transposing a previously-used solution to a new context. However, overall we would surmise that problem-solving in this project has been more reliant on reproductive thinking and trial-and-error approaches, which we believe is due to the relatively well-defined solution state that was established at the outset, i.e. we had clearly defined goals for the project, rather than an open-ended creative brief.

Consequently, solutions tended to be derived from strategies that fit more closely with information processing theory (see Alan Newell and Herb Simon's attempts in the 1950s-60s to model human thinking processes). This paradigm of creative thinking aims to break the scenario down into problem state and solution state, and then using knowledge of pre-programming and past experience, one attempts to apply operators within the problem state in order to partially transform the situation and move closer to the solution state.

Evaluation

The whole system of cross-referencing data from the databases was perhaps the hardest to implement, since it involved implementing tools to analyse, track, count, sort and correlate different data types using a range of variables. As well as being a challenge to accomplish, due to the convoluted series of objects implemented to control the flow of data, we also encountered issues with reliability as more and more bugs started to present themselves. For example, when the software was initially booted, it was able to accurately recall stock values from its previous state, but our attempts to integrate a manual refresh function presented an obstacle since the updated data was applied incorrectly within the databases. This is typical of the issues faced when developing a relatively complex system such as this; however, we regard this as an intrinsic challenge of programming, and as we have touched on earlier in this paper, we have found that skills and fluency working in this context do improve with experience. Nevertheless, the debugging process is a difficult one that requires a holistic and detailed understanding of the system, data flow and message ordering.

This project is still in its adolescent stages, and there are many further features to develop that will support a more refined and ultimately effective system in response to the aims outlined at the outset. For example, whilst we have successfully developed a system to monitor quantifiable stock such as tins, it has been more difficult to decide how to monitor more awkward items such as butter or milk, which the user is more likely to use small amounts of quite often, and recipes might stipulate various quantities which do not fit to simple fractions of a container. Nevertheless, the current functionality points to the potential of the software, and features such as the correlation of nutritional information to totals used by recipes works well and offers a usable tool for users to monitor their intake of salt, fat, sugar and so on, which responds to the initial motivations for this project.

Further developments might also include a system that tracks recipes chosen by the user each week to suggest new recipes that they might enjoy, and more functions to sort recipes in different ways, for example easiest to cook, quickest time to prepare, and so on. Whilst these ideas represent further challenges, from our present experience we feel that they are achievable within Max MSP. Our prior experience developing audio and visual systems incorporating physical computing, sensors and so on offers further scope for transposition of practice into other domains, and we reflect positively on the flexible nature of this particular programming environment and the remix practices identified in this paper that transcend the boundaries that often exist between designers, makers, artists and users.

References

Azeredo, M. (2007) 'Real-time composition of image and sound in the
(re)habilitation of children with special needs: a case study of a child with cerebral palsy', Digital Creativity 18(2): 115-120

Department for Environment Food & Rural Affairs (2015) 'Digest of Waste and Resource Statistics – 2015 Edition', DEFRA.

Health & Social Care Information Centre (2014) 'Statistics on Obesity, Physical Activity and Diet: England 2014', HSCIC.

Manovich, Lev (2013) Software Takes Command, London: Bloomsbury.

Manovich, Lev (2015) 'Remix Strategies in Social Media', in Eduardo Navas, Owen Gallagher & Xtine Burrough (eds.) The Routledge Companion to Remix Studies, New York: Routledge, pp. 135-153.

Michalowski, M.P., Sabanovic, S. & Kozima, H. (2007) 'A Dancing Robot for Rhythmic Social Interaction', Proceedings of Human-Robot Interaction Conference 2007: 89-96. Washington DC, USA, March 9-11 2007.

Moens, B., van Noorden, L. & Leman, M. (2010) 'D-jogger: Syncing music with walking', 7th Sound and Music Computing Conference: 451-456. Universidad Pompeu Fabra.

U.S. Department of Agriculture, Agricultural Research Service (2014) 'USDA National Nutrient Database for Standard Reference, Release 27', [online]. Available from: http://www.ars.usda.gov/ba/bhnrc/ndl [accessed 16 Feb 2016].