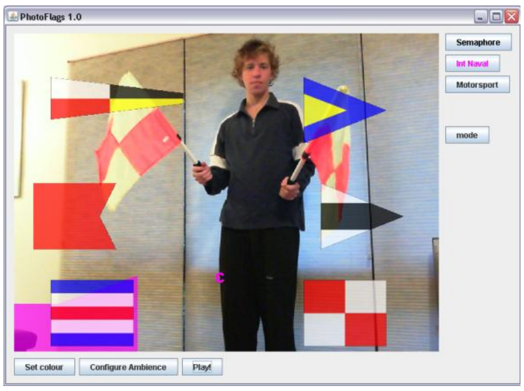
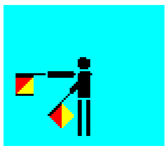
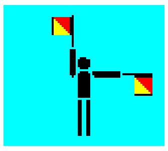
System Report: PhotoFlags



Introduction

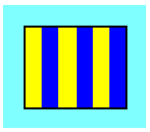
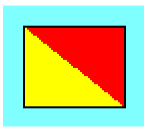
The aim of this software is to assist users learning the various flag communication styles. The specific styles in this program are Semaphore, International Maritime and motorsport. The program is run on the Java platform, including the Java Media Framework and the Java Advanced Imaging packages. The user is required to have a webcam.

Background

The Semaphore communication system was a major form of close communication from ship to ship for centuries. In the Canadian Navy for example, the art of signalling with flags was taught to junior Signalmen until early this century. With modern reliable communication onboard modern ships, the Semaphore system is no longer instructed as its use is rare.

The Semaphore system is also quite popular with Scouts and they would have instructors teaching it to the beginners.

The International Maritime Signal flags are used by NATO and for boating in general. There are various ways of sending messages with these flags; it might be by assigning letters to the flags or by giving messages to the flags. It is also possible to have multiple flags combined to give a more complicated message which is its own code; not just a string of its constituent flag codes. This program supports the instances where one flag is one letter and where one flag is one message.

The motorsport flags in this program are used both for car racing and for motorcycle racing. Their use between various race organisers differs only slightly. The standard used in this program is that of the Fédération Internationale de l'Automobile.

System Design

The system has been designed in a very modular way; the modules involved are the GUI controller, the overlay display, the video processing and of course the questioning.

The design for the user interface was to have the user using their own flag, then configuring the system to recognise that flag. The Semaphore flag system uses the position of the flags to convey a message, so that's obviously what they'd have to do in this system. As for the motorsport and International Maritime flags, they rely more heavily upon the look of the flag itself, so it was decided that these would be tested through multiple choice, with the user having to move their flag over the correct one. Testing the user in this way certainly requires more effort than having them just press a button, but that is simply not as much fun.

The software package is designed with a mix of education and light entertainment; it's not as much fun as Warcraft, but it's certainly more fun than reading an encyclopaedia to learn the signalling methods. What's also important is that the examples are generated instantly; learning through repetition can often be dull because the user is often required to think of their own examples and then test themselves. The psychology behind this should be explained by a more knowledgeable source, but the usual result is that it's simply more tiresome and uninteresting than examples prepared by someone else.

The software is made mainly with the Semaphore system in mind; that is partly because it's very recognisable and also because this program can encapsulate all the necessary knowledge for that system. The motorsport flags have various movements to accompany them, so a user trained only by this software would able to pick the right flag but might look a bit unprofessional in their waving of the flags. Likewise, the information taught by this program in relation to the International Maritime flags is sadly not quite complete. As mentioned earlier, these flags are sometimes shown in a way such that a message can only be made by combining these flags. If the user would like to learn everything there is to know about the International Flag system, then this program is useful more as an easy introduction to the system, rather than a complete guide. The main reason for limiting the software was the timeframe.

Software modules

The classes in this project are as follows:

- *Entrance.java* – displays the opening message, then hands control over to guiControl
- *guiControl.java* – responsible for managing the user interface and for coordinating the interactions between the sensing and the display.
- *FlagCast.java* – responsible for displaying the flags and answering regions, as well as thinking of the questions
- *FlagWatch.java* – responsible for sensing the flags
- *SmallScreen.java* – a simple class merely for displaying a BufferedImage on the screen
- *EyeDropper.java* – uses the colour of pixels under mouse clicks to determine the colour settings for the flag

The FlagCast class creates some shapes which represent the correct regions for holding the flag. If for example, the user is required to hold the flag up in the top-left corner, then the program will consult the pixels who comprise this top-left shape. Some may say that the code could be made more object-oriented by abstracting this collection of shapes into its own class, but this class wouldn't be especially useful. The logical way of improving the class would to have it capable of displaying itself by adding in a paint() method, but then there'd be a paint() method for these shapes and another displaying method for the multiple choice flags.

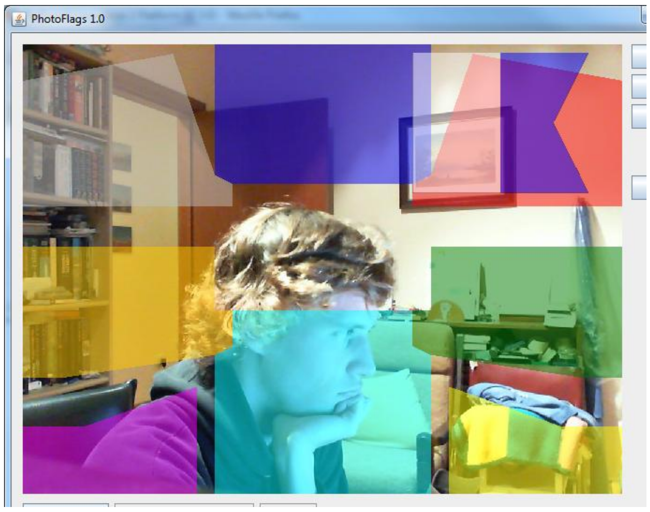
State Transitions

The program has independent game states and configuration states. The configuration states follow a linear path of setting the flag colour, configuring the ambience, then playing the game. The game states can be either Semaphore, Motorsport, International Naval as letters or International Naval as single-flag messages. The game states are identified in guiControl by the character '*game*'.

Since these two states are independent, it's possible for the user to set whatever game they like before the program actually starts playing any of those games. If the user is already playing, then they must finish their current question before moving on to the new game.

Flag sensing

When the FlagCast class is created, it uses a snapshot from the webcam to scale the sensing areas appropriately. The FlagWatch instance '*drapeau*' (French for flag) is then given this FlagCast class so that it can come up with a list of pixels lying within each sensing shape. FlagWatch uses the settings gained from the EyeDropper class to configure the colour channel settings matching that of the flag, as well as the variance in these colour channels.



The sensing shapes can be seen in this image to the right. Take note that the top-right sensing area is the red shape, the flag in this corner is unrelated.

The pixels are stored for the moment in ArrayLists for the x and y pixels in each shape. Once the user hides the flags and presses the ambience button, the FlagWatch class analyses a new image to look for areas similar enough in colour to be identified as the flag. These pixels are then deleted from the ArrayLists so as to avoid the program later mistaking them for the flag. Here it is obviously assumed that the user won't be moving their webcam around while using the program.

Once the ambient occlusion has completed, the points are moved into int[][] arrays to allow the pixels to be traversed faster than if they were still kept in ArrayList<Integer> arrays.

When sensing whether or not the user's answer is correct, the FlagCast reports the correct answering region to guiControl, which in turn requests for FlagWatch to check those appropriate pixels. Remember that FlagCast is the one thinking of the questions. The guiControl class keeps track of the time it's taking for the user to answer the question. If the user takes too long, then it requests the FlagCast class to highlight the correct answering region. The guiControl class also plays a disparaging sound.

It should be noted that the guiControl class isn't actually measuring the *time* for the user to answer the question; it's counting the number of iterations through the thread. Due to various time allocations, the user is given differing periods of time to answer the question. This could've been avoided by having an actual timer there, but when this inconsistency was noticed, it was decided that it keeps the program more interesting; having a uniform time threshold would simply be more boring.

Question generation

The way in which questions are asked remains the same for Semaphore and motorsport for every question. For the Semaphore system, the program always shows a string, then has the user make the appropriate gestures for each letter. For the motorsport flag system, the user is always offered four choices and must pick the one matching up to the description given. The International Maritime flags offer more variety in that the user can choose between being given the letter or the 1-flag message.

When thinking of flags to offer the user in the case of multiple selections, each flag is chosen randomly out of the collection with no consideration given to any factors. This results in the case that the correct answer is sometimes shown as multiple choices on the board. This was considered as undesirable in the initial design and it was deemed that the most effective way to deal with it without sacrificing runtime was to force each flag to be chosen in a spot after the previous flag. This way, the same flag could never be chosen twice.

This design however presented unforeseen consequences; the amount of choice for the last flags was significantly lower than for the first flags, meaning that it was quite common to see the same flags being chosen in spots 5 & 6 for multiple rounds. Therefore, flags that happened to be stored towards the end of the collection were being shown again and again. This is very similar to the paradox of flush values in poker; a flush is more valuable if it's highest card is higher, even though it is much more unlikely to have a flush whose highest card is only a 6 or 7.

This repetition meant the program was changed to its current state of making purely random choices. The simplest way of avoiding this would be to check all the current flags after making every choice, then coming up with a new guess if there was any repetition. This would be a very time-consuming process and was deemed to be too much to weigh out the slight hassle of having the correct flag presented multiple times. Also it's mathematically possible that the program would continuously assign clashes and would be stuck in an infinite loop. You'd then have to count the number of trials and default to some sort of error condition like flag1 = 1, flag2 = 2, ..., but by then there'd be so many checks and assignments that just choosing a flag would almost be a whole program in itself.

Presenting multiple correct answers to the user means that even if they're a genius, they must get an answer wrong occasionally and this keeps the program challenging and exciting. If the user was always correct, they would be quickly bored.

Mirror design

One main concern with the current program is how users naturally want to interact with it like a mirror, but it doesn't behave like a mirror. As mentioned earlier, it was decided that Semaphore was the main flag system addressed by the program, so in order to implement this correctly, the left side of the input image had to be considered as the right hand.

An initial idea for the program was to include a mode where an avatar would send a Semaphore message to the user and the user would either reply or transcribe this message. If the user was used to seeing their flags in a mirror, they would naturally associate their perception of the flags as "I", rather than their hand movements, since vision is naturally a lot easier to remember. When the user saw the avatar moving their flags left for "I", it would appear on the right of the screen and the user would be confused. Even if there was no avatar in the program, when the user went out on the high seas and saw someone else sending a Semaphore message back, it would appear on the opposite side and be very confusing if the user had learnt the flags while looking in a mirror.

This future difficulty in learning meant the decision was made not to invert the camera and hence, to have the users awkwardly adjusting to the program on the first few occasions they use it.

Future Improvements

The program really excels in that it has been designed with the future strongly in mind. The current layout of the program means that upgrading the program is relatively straight-forward. A simplistic idea for improving the program would of course be to expand the amount of schemes available (eg Football flags). This however may be irrelevant to the types of users running this software.

The decision not to display a mirrored image was explained for the Semaphore flags, but some users might prefer for the image to be mirrored when running in the multi-choice modes of the program. Therefore, it would make sense to have this as an option to users.

As mentioned earlier, there are more complicated ways of forming messages with the International Maritime flags, so in the future, it might be beneficial to offer modes where the user was required to choose two flags in order to form a message. It may also be good to see strings of commands put together in the same way that the Semaphore characters are tested.

After using the program, it would be evident that the character strings for the Semaphore messages are being generated by putting together a set of random characters. It may be preferable if the user could import a text file of words or short sentences, then have the program pick out examples from that file. By forming questions this way, it would probably feel a lot more natural to users.

Conclusions

The finished program encapsulates a range of software components to form a system with practical applications. The system is relatively robust and its state transitions are all stable. It has been designed with the user's enjoyment in mind, as well as the effectiveness of the teaching.

The program covers the most popular flag communication styles and also remains easily scalable for further systems. Users can enjoy the program in its current form to gain a good grasp of these useful flag communication styles.

References

International Maritime flags were sourced from Jim Croft from the Australian National Botanic Gardens: http://www.anbg.gov.au/flags/signal-flags.html

Image of Filipino soldiers sourced from LIFE: http://www.life.com/image/53373170

Sound files in the program were sourced from Worms3D, made by Team17 Software Limited and SEGA Corporation.