MIMED – PROPOSAL FOR A PROGRAMMING LANGUAGE AT THE MEETING POINT BETWEEN CHOREOGRAPHY, MUSIC AND SOFTWARE DEVELOPMENT.

Alexis Kirke[1], Olivia Gentile[2], Federico Visi[1], Eduardo R. Miranda[1]

[1]*Interdisciplinary Centre for Computer Music Research, School of Humanities, Music and Performing Arts, University of Plymouth, Drake Circus, Plymouth, PL4 8AA, UK*

[2]*Barbican Theatre, Castle Street, Plymouth, PL1 2NJ, UK*

Correspondence should be addressed to: alexis.kirke@plymouth.ac.uk

Abstract: **A new approach to programming is introduced involving a gesture and movement-based system combined with musical feedback. The concept is introduced through a proposal for the programming language MIMED (Musically-backed Input Movements for Expressive Development). In MIMED the programmer uses gestures to code, and as the program code is entered the computer performs music in real-time to highlight the structure of the programme. MIMED data is video data, and the language can be used as a tool for teaching programming for children, and as a form of programmable video editing. It can also be considered as a prototype for a more immersive form of programming that utilizes body rhythm and flow to create a different subjective experience of coding; almost a dance with the computer. This paper begins with an over of the motivation for MIMED, a survey of related work, and a description of the structure and symbols of the language and the soundtrack elements. This is done using examples of programming in MIMED, and is followed by a discussion of next steps in developing the language.**

1. INTRODUCTION

"Flow" in computer programming has long been known to be a key increaser of productivity [1]. Also many developers listen to music whilst coding. This has been shown to increase productivity [2]. There have also been tests on the effectiveness of debugging programs by sonification [3] and creating audio-enabled development interfaces [4]. There are now low cost high accuracy motion detectors for computer input such as Leap Motion, and there are multiple attempts to define core gesture sets for such a gesture UI [5]. It is also commonly known that music encourages and eases motion when it is synchronized to its rhythms. This paper proposes a programming language MIMED (Musically-backed Input Movements for Expressive Development) whose instruction set is made up of gesture inputs via motion detection. The development environment incorporates a generative soundtrack based on gestures detected in real-time, supporting the user in gesture rhythm and programming Flow. The music also represents information about code syntactical and structural elements. Visual commands equivalent to BASIC commands such as Print, Repeat, Input and conditional structures are assigned their nearest American Sign Language (ASL) equivalent.

2. RELATED WORK

There is a rich history exists of computer languages designed for teaching children the basics of programming. LOGO [6] was an early example, which provided a simple way for children to visualize their programs through patterns drawn on screen or by a "turtle" robot with a pen drawing on paper. Some teachers have found it advantageous to use music functions in LOGO rather than graphical functions [7]. A language for writing music and teaching inspired by LOGO actually exists called LogoRhythms [8]. However the language is input as text. The language was developed so as to teach non-programming-literate musicians to write scripts. Although tools such as MAX/MSP already provide non-programmers with the ability to build algorithms, and educational languages such as Scratch [9] a way of teaching it, their graphical approach lacks certain features that a scripting language such as Java or C provide.

As well as providing accessibility across age and skill levels, sound has been used to give accessibility to those with visual issues. Emacspeak [10] for example makes use of different voices/pitches to indicate parts of syntax (keywords, comments, etc). There are more advanced systems which sonify the Development Environment for blind users [11] and those which use music to highlight errors in code for blind and sighted users [12]. There is also an entirely tone-based programming language in development whose musical structure mirrors its code structure [13]. The field actually called gesture-based programming [14] is focused on robotics. The user shows a robot the movements they want it to imitate. Actual programming languages based around gestures do not seem so common. The closest development is a 3D gesture system for object oriented programming where hands switch between being a form of 3D mouse, and being able to manipulate and link objects and threads [15]. Computer control by gesture is a widely research area with multiple surveys [16]. As is sign language recognition by computer, including real-time sign-to-text translators with Kinect [17].

3. MIMED STRUCTURE

Inputs to MIMED can be through a video camera and / or a gesture sensor system. MIMED outputs come through a computer screen (or VR/AR headset) and speakers or headphones. During programming and, if desired, execution, MIMED plays a generative soundtrack. Rather than giving a formal definition of MIMED, we will first give an example MIMED programming session. Because of its multimedia nature, we have put some videos up showing the links between music and sound, available at http://cmr.soc.plymouth.ac.uk/alexiskirke/mimed.htm

The program to be entered is:

```
Repeat 3 times
        Show "Smile"
        Repeat 2 times
                Show "Waving"
        End
End
```

The Show command works in a similar way to printf in C, or print in BASIC. The Repeat X command repeats the code between itself and its relevant End, X times. So the above program will output:

```
Smile
Waving
Waving
Smile
Waving
Waving
Smile
Waving
Waving
```

Note however that MIMED is designed as a visual-based language. So rather than entering the text "Smile" and "Waving" the object is a video clip of the user smiling or waving. This will be clarified below. The user activates the environment by a hand signal for ASL "Begin" shown in the left hand side of Figure 1.

Figure 1: ASL Signs for Begin (left) and Repeat (right)

After Begin is signed, the interface will play a drum beat. This is either an urban drum loop, or a four-to-the-floor electronic bass drum. For this discussion the electronic dance music version will be used, i.e. the electronic bass drum. Figure 2 at the end of the paper shows how the sounds build up. Bar 1 shows the sounds after the Begin sign. Line one of the above programme is a Repeat command. This is shown on the right hand side of Figure 1. The Repeat command causes a single bass-line note C to repeat on the first beat of each bar as in bar 2 of Figure 2. This tells the user the system is waiting for the Repeat count. The user then holds up their fingers for the count of 3, as shown in the left of Figure 3.

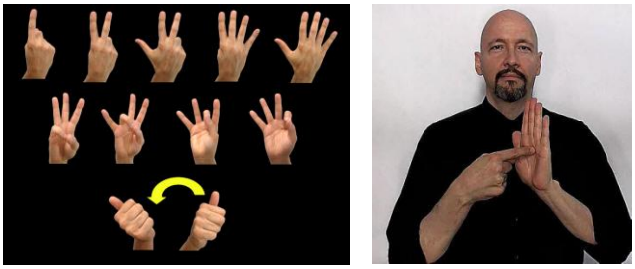


Figure 3: ASL Signs for 1-10 (left) and Show (right)

After the user signals "three", the bass line will change to three offbeat C notes per bar, as shown in Bar 3 of Figure 2. This tells the user that everything being signed now will be repeated 3 times when the program is executed. The show command uses the ASL sign for "show" which is demonstrated in the right hand side of Figure 3. When this command is detected, the system plays an continuous electronic "pad" or string-like sound, seen in bar 4 of Figure2. This is played on pitch E to tell the user that this Show command is one indent in – i.e. it is within a control structure. This process of moving up an interval happens at each indent in a control structure. The user can then make some movements and these are recorded as the data that will be displayed during execution of the Show command. Once they lower their hands, the Show command entry deactivates and the pad stops playing. So in this case they smile.

The next Repeat command on line three of the program is signalled by the user with the ASL Repeat sign from Figure 1. The bass-line changes to that seen in Bar 5 of Figure 2 – a single bass note at the start of each bar. The pitch is now E, because the Repeat is a nested Repeat. The user signals a repeat count of 2 using the ASL for "two" from the left hand side of Figure 3. This causes the dynamic bass-line to switch to that in bar 6 of Figure 2. It is two off-beat bass notes at pitch E. This tells the user that the commands now being entered will be repeat twice during execution. The outer Repeat loop bass notes continue at pitch C.

Another Show command is signalled, as in the right hand side of Figure 3. The pad sound starts again, this time at pitch G – as shown in bar 6 of Figure 2. The rise in pitch once again indicates that the command is occurring at a yet higher level of indent. The user makes a waving motion and then stops moving. The system switches off the pad and stores the wave clip. The End command in line 5 of the above program pairs with the Repeat 2 command in line 3. End is signalled using the ASL symbol shown on the left of Figure 4. This will cause the "outer" bass-line at pitch E to stop, as happens in bar 9 of Figure 2. Thus the user is aware of which Repeat structure has completed entry. The sign for End in Figure 4 is used again, and all bass-lines will stop, telling the user they have ended the outer Repeat structure. This leaves only the bass-drum playing (as in bar 1 of Figure 2). The user completes program entry using the Finish sign given on the right of Figure 4.

Figure 4: ASL Signs for End (left) and Finish (right)

At this point the drum beat will stop and programming is completed. To execute the program the ASL sign for Use is given, as in Figure 5. The resulting execution will be an edited together video of a smile and two waves repeated three times. The video will have a soundtrack that indicates which parts of the code are being executed. So during the inner repeat both bass lines will occur, during the smile an E pitch pad will be heard, and during the wave a G pitch pad, and so forth.

Figure 5: ASL Sign to execute the programme

A more complete set of MIMED commands is given in Table 1 at the end of this paper. This allows the creation of programs such as the following:

```
Get X
Get T
Show X
Repeat 2
        Get O
        Show T + X
        If X = O
                Show O + T
        End
End
```

The variable names X and T are created by putting the arms into a diagonal cross shape, and a T-shape respectively. The variable name O is made by making an O-shape with the hands using thumb and forefingers. To save taking up space with more photographs, the code above will be transformed into verbal sign code as follows (see Table 1 as well):

1. ASL "get", then make X shape with arms
2. ASL "get", then make T shape with arms
3. ASL "show", then make X shape
4/5. ASL "repeat", then ASL for "two"
6. ASL "get", then make O shape with hands
7. ASL "show", then make T shape with arms
8. ASL "add", then make X shape with arms

10. ASL "if", then make X shape with arms
11. ASL "equals", then make O shape with hands
12. ASL "show", then make O shape with hands
13. ASL "add", then make T shape with hands
15. ASL "end"
ASL "end"

Figure 6 at the end of the paper shows the music that would accompany this, based on Table 1. Next to each sign above is a number which indicates the bar when the music is triggered.

3. MIMED DEVELOPMENT

MIMED design has followed an unorthodox path. The initial design work was done without a language kernel or visual gesture recognition. Since then prototype visual recognition has been implemented for a subset of the signs, using low resolution imaging and an AdaBoost algorithm [18]. The AdaBoost algorithm is connected to a music composing tool which contains a series of loops broken down into the sonic elements in Table 1. So for each sign detected, an appropriate loop is triggered.

However initial work was mostly, and is still being done, in collaboration with dancers, led by the second author of this paper. This allowed the basic language structure to be developed through work shopping with movement artists and a musician. One complication that emerged was how to deal with multiple nestings such as the program below, without taxing human memory too much:

```
GET X
REPEAT 2
        GET T
        REPEAT 3
                IF X = T
                        DISPLAY T + X
                END
        END
END
```

A methodology was developed called the Dancer Stack Protocol. In this case the dancers were numbered 1 to 4. The programmer would begin by signing to dancer 1, then at the first indent at line 3 above they would begin signing to dancer 2, then as each indent occurred they would move on to the next dancer, finishing with dancer 4 at line 6. This divides the program up:

```
1 .D1   GET X
2. D1   REPEAT 2
3. D2           GET T
4. D2           REPEAT 3
5. D3                   IF X = T
6. D4                           DISPLAY T + X
7. D3                   END
8. D2           END
9. D1   END
```

Programme execution then begins with dancer 1. When dancer 1 executes the Repeat 2 statement their job is to signal to dancer 2 *twice*. So they signal to dancer two a first time, and dancer 2 executes their own code. Then when dancer 2 has finished their code they signal back to dancer 1. Next dancer 1 will signal to dancer 2 a second time, and dancer 2 will repeat their code. When dancer 2 signals back to dancer 1 they are finished, dancer 1 notes that two Repeats have been done, and so doesn't trigger dancer 2 again. Note that because dancer 2 does a similar thing to dancer 3, and dancer 3 to dancer 4 – it can be seen that the whole program executes across a "dancer stack". This allows each dancer to only have to recall one decision condition or repeat count. This process is analogous to the recursive execution of programme code. It created a forum in which active testing and experimenting could be done on the structure of the language.

As well as the Electronic Dance Music version of the MIMED shown in Table 1, an Urban music version of MIMED has been defined, and in fact utilized in a public demonstration using the Dancer Stack [19]. A similar process could be carried out to define a Jazz music version, and a classical orchestral version.

MIMED has an interesting feature that emerges from its sound-tracked property. It can be introduced as a programming language to children. Or it can be introduced as a performance system. So looking at the signs in Table 1, children can be told that the "Begin" sign stands for "drums" rather than "Begin", and that "Repeat 3" means "Bass line with 3 notes", and that the Display sign means "pad". The language can be introduced as a constrained musical game, and then slowly the students mind can be turned to the true meaning of the signs. We are not aware of another educational programming language that can be introduced this way.

One final feature of MIMED that will be mentioned is its ability for computation. With its current language set, it is possible to add numbers. For example is X is defined as a clip of four hand waves, and Y is defined as a clip of three hand waves, then the command "Show X+Y" will edit together the two clips and display seven hand waves (i.e. 3+4). To multiple X by Y, the following program is used. Note that this code snippet introduces two unmentioned feature of MIMED, that the parameter of the Repeat command can be set to a variable value, and that variables are defined on first assignment:

```
GET X
GET Y
REPEAT Y
        Z = Z + X
END
```

In this case if the user executes the programme and enters a clip of four hand waves for X, and the sign for three (see Figure 3) for Y, then at the end of the code running, there will be twelve hand waves (3*4) edited together in Z. There is currently no way of subtracting and dividing in MIMED.

4. FUTURE WORK

Having proposed, work-shopped and thus defined the basics of the language the next step is to extend the sign language visual recognition system to increase its accuracy and vocabulary. It is not clear how robust such a system is to the user moving the rest of their body. It was seen in some of the workshops that a MIMED programmer tends to move their body to the rhythm of the soundtrack. This a highly desirable trait, as it leads to a sense of flow with the system, but it complicates gesture recognition. An additional complication is deciding precisely how user-defined objects start and begin. The simplest approach is to consider arms at side to be a delineator. So if the user makes an X sign with their arms and then lowers their arms to their side, the X sign clip is all that is stored. However, the arms to the side reduces the ability to move the body more freely. This raises the issue of whether an explicit gesture is required to highlight user-defined clip start and ends.

The recognition of signs is also complicated by the fact the user can define their own signs. For example they can assign X (cross arms) to another user defined video, say the right hand waving twice. This means that when X is utilized in future during programming, MIMED must recognize it as being the same X defined earlier. In theory the symbol for a variable does not have to be static visual patterns but can be a dynamic movement pattern. However in practice this may be too difficult to achieve reliably.

Other elements to be addressed include this being a programming language designed around sign language for the deaf, but having at its core the use of sound. To address this we plan to translate MIMED to the vibration mode. In this case the user could have a

vibration attachment– for example – a mobile phone in their pocket. The attachment then generates layered vibration rhythms to indicate the program structure, in a similar way that sound does in the EDM and Urban versions of MIMED.

In addition to the completion of the gesture detection work, the MIMED interpreter core needs to be fully developed. This is – at its heart – a form of conditional algorithmic video editing software. Such macros are in fact available for video editing using Apple Script. There are also various video manipulation APIs available in languages such as Python. However the MIMED tool can be much simpler.

It is also planned to investigate the applications of affective computing within MIMED. Given the MIMED input system is a visual one, facial mood recognition can be incorporated. This is a highly researched and practical area with current commercial systems [20]. This makes feasible the creation of an affective (emotional) variable type, whose value indicates emotional positivity or negativity. Such values can be computed with, used in conditions, compared, and so forth. This would make an extended version of MIMED a natural test-bed for the concept of affective programming languages.

A final question for future work: can MIMED be utilized outside of education? The BASIC programming language was designed as an educational language, but became a fully-fledged commercial language with Visual Basic and the various forms of VBA. Can the core concepts of MIMED make a similar transition? To address this it may be necessary to seek out more robust and fully formed sign language recognition systems to integrate with the interpreter and sound production system.

5. CONCLUSIONS

We have proposed and introduced a programming language MIMED whose instruction set is made up of gesture inputs via motion detection. The proposed development environment incorporates a generative soundtrack based on gestures detected in real-time, aiding the user in gesture rhythm and programming Flow. The music also represents information about code syntactical and structural elements. Visual commands equivalent to the BASIC commands Print, Repeat, Input and conditional structures are assigned their nearest ASL equivalent.

We have discussed the motivation for MIMED, provided a brief survey of related work, and a description of the structure and symbols of the language and the soundtrack elements. This included some examples of programming in MIMED, and a discussion of next steps in developing the language.

MIMED data is video data, and the language can be used as a form of programmable video editing, and as a tool for teaching programming for children. It can also be considered as a prototype for a more immersive form of programming that utilizes body rhythm and flow to create a different subjective experience of coding; almost a dance with the computer.

REFERENCES

[1] M. Csikszentmihalyi: *Flow and the Psychology of Discovery and Invention*. Harper Perennial, New York, 1997.

[2] T. Lesiuk: *The effect of music listening on work performance*. In *Psychology of Music*, volume 33(2):173-191, 2005.

[3] P. Vickers and J. Alty: *Siren Songs and Swan Songs: Debugging with Music*. In *Communications of the ACM*, volume 46(7):86-92, 2003.

[4] A. Stefik, C. Hundhausen , D. Smith: *On the design of an educational infrastructure for the blind and visually impaired in computer science*. In *Proceedings of the 42nd ACM technical symposium on Computer science education*, pages 571-576. ACM, New York, NY, USA, 2011.

[5] J. Zigelbaum, A. Browning, D. Leithinger, O. Bau and H. Ishii: *g-stalt: a chirocentric, spatiotemporal, and telekinetic gestural interface*. In *Proceedings of the fourth international conference on Tangible, embedded, and embodied interaction*, pages 261-264. ACM, New York, NY, USA, 2010

[6] B. Harve: *Computer Science Logo Style*, MIT Press, USA, 1998.

[7] M. Guzdial: *Teaching Programming with Music: An Approach to Teaching Young Students About Logo*, Logo Foundation, USA, 1991.

[8] A. Hechmer, A. Tindale and G. Tzanetakis: *LogoRhythms: Introductory Audio Programming for Computer Musicians in a Functional Language Paradigm*. In *Proceedings of 36th ASEE/IEEE Frontiers in Education Conference*, IEEE, San Diego, CA, 2006.

[9] J. Maloney, M. Resnick, N. Rusk, B. Silverman and E. Eastmond: *The Scratch Programming Language and Environment*. In *ACM Journal of Transactions on Computing*. volume 10(4):article 16, ACM, New York, NY, USA, 2010.

[10] T. Raman: *Emacspeak - A Speech Interface*. In *Proceedings of 1996 Computer-Human Interaction Conference*, ACM, New York, NY, USA, 1996.

[11] A. Stefik, A. Haywood, S. Mansoor, B. Dunda and D. Garcia: *SODBeans*. In *Proceedings of the 17th international Conference on Program Comprehension*. IEEE Computer Society, Vancouver, B.C., Canada, 2009.

[12] P. Vickers and J. Alty: *Siren Songs and Swan Songs: Debugging with Music*. In *Communications of the ACM*, volume 46(7):86-92, 2003.

[13] A. Kirke and E. Miranda: *Unconventional Computation and Teaching: Proposal for MUSIC, a Tone-Based Scripting Language for Accessibility, Computation and Education*. In *International Journal of Unconventional Computation*, volume 10(3):237-249, 2014.

[14] R. Voyles: *Gesture-based programming: a preliminary demonstration*. In *Proceedings of 1999 IEEE International Conference on Robotics and Automation*, volume 1:708-713, IEEE, Detroit, MI, 1999.

[15] R. Herrera-Acuña, V. Argyriou and S. Velastin: *Toward a 3D Hand Gesture Multi-threaded Programming Environment*. In *Advances in Visual Informatics Lecture Notes in Computer Science*, volume 8237:1-12, Springer-Verlag, 2013.

[16] H. Hasan and S. Abdul-Kareem: *Human–computer interaction using vision-based hand gesture recognition systems: a survey*. In *Neural Computing and Applications*, volume 25(2):251-261, 2014.

[17] Z. Zafrulla, H. Brashear, T. Starner, H. Hamilton and P. Presti: *American sign language recognition with the Kinect*. In *Proceedings of the 13th international conference on multimodal interfaces*, pages 279-286, ACM, New York, NY, USA, 2011.

[18] Y. Freund and R. E.Schapire: *A decision-theoretic generalization of on-line learning and an application to boosting*. In *Journal of Computer and System Sciences*, volume 55(1):119-139, 1997.

[19] A. Kirke: *Programming computers with a 'wiggle, wiggle, wiggle'*, presentation, TEDx Plymouth University, Plymouth University, 2014

[20] G. Sandbacha, S. Zafeirioua, M. Pantica and L. Yinc: *Static and dynamic 3D facial expression recognition: A comprehensive survey*. In *Image and Vision Computing*, volume 30(10): 683-697, 2012.

Figure 2: Build-up of generative music in the electronic dance music version of MIMED.

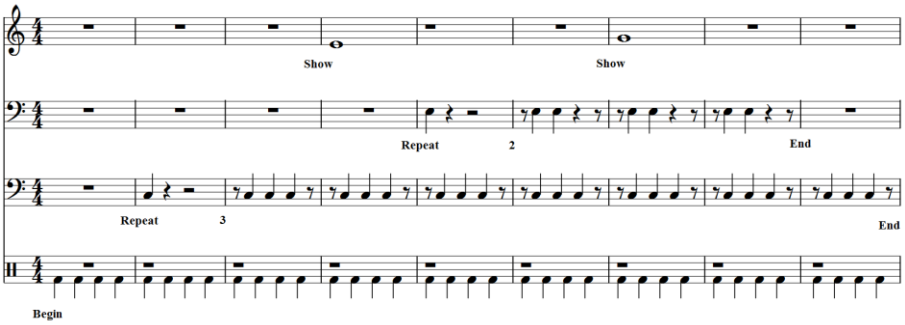


Figure 6: Generative music for the more complex program utilizing broader range of commands

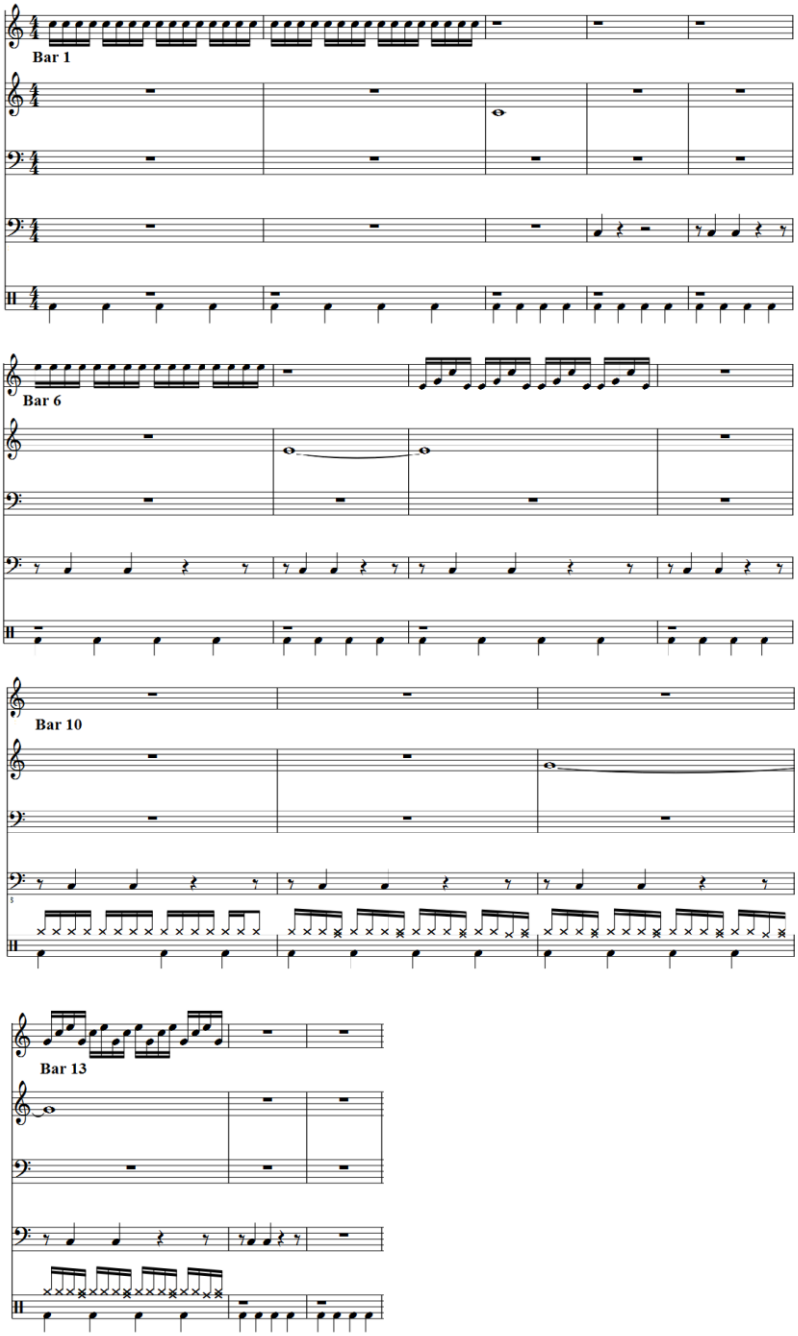
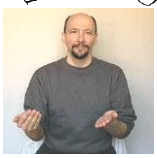
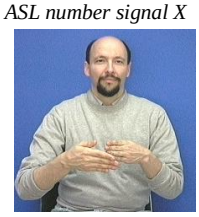
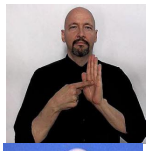


Table 1: Command list for MIMED

Sign	Command	Function	Sound-track in EDM Mode
 	Begin	Start program entry	Drum beat start
	Finish	End program entry	All stop
  *then an ASL number signal X*	Repeat X	Repeat all code X times. Code block end marked with End.	Single bass note at start of each bar, switch to offbeat bass notes whose number is the number of repeats.
 	End	End contiguous code block – If or Repeat	Stop last element of sound-track created by that code black start.
	Show	Display a piece of view.	Continuous pad sound
	Use	Execute the stored code.	During execution, the appropriate soundtrack for those commands currently running is played.
 	Get	Gets video during execution and stores it in a variable; c.f. Input from BASIC, or cin in C++	Gated pad.
 	If	Test a condition and if true then execute the block of code. Code block ended by End.	Closed high hat pattern until end of code block
 	Equals	Compares video clips, or assigns one video clip to be represented by another.	Open high hat pattern until end of code block
 	Plus	Combines two video clips	Fast electronic arpeggio