The Software Model of the "Speedy'O'Brain" Neuro-Feedback Videogame

Luigi Bianchi

Abstract— In this manuscript, some design aspects of the Speedy'O'Brain neurofeedback videogame have been outlined. It is built on top of BF++, a framework for building bio-feedback and Brain-Computer Interfaces. It makes large use of object-oriented programming techniques and the adoption of a well-defined functional model is the key for making it easily adaptable and customizable, as herein described. The chosen programming language, C++, the compatibility with a widely used software platform and its free availability, makes the proposed solution a valuable starting point for building neurofeedback applications.

***Keywords*—** Neuro-feedback, EEG, NPXLab, BF++, videogame, rehabilitation, regression, model, software.

I. Introduction

IN recent years, neurofeedback (NFB) gained a lot of interest because it has been demonstrated that it could support not only medical doctors for diagnostic purposes but also therapists in the rehabilitation process of people affected by several neurological disorders. These include Attention Deficit Hyperactivity Disorder (ADHD) [1, 2], epilepsy [2, 3], schizophrenia [4], cerebral palsy [5], stroke [6 - 8] and traumatic brain injuries [9]. Recently, even non-clinical applications have been proposed, aimed for example at improving motor skills in athletes [10].

The principle on which NFB is based on is that people can be trained to self-regulate some brain signals and events that are collected and processed in quasi-real-time through a system that provides visual or acoustic feedback of their ongoing neural activity. In other words, brain signals are converted in some form of feedback with a well-defined law that relates the neural signals with the feedback and that is specific and characterizes each NFB application. Note that the presence of the feedback modifies brain signals and this implies that most of the offline data analyses cannot be performed because signals have been modified by the neurofeedback mechanism.

There are several different ways to perform this task, and several different brain signals can be used. Among them, those based on noninvasive methods such as electroencephalography (EEG) and functional Near Infra-Red Spectroscopy (fNIRS) are actually preferred and the most widely used, also because of their relatively low costs.

Magnetoencephalography (MEG) has been also successfully used as a brain signals recording method, but it represents a too expensive solution and it is actually used only in a limited number of research laboratories.

In a previous work [11] I have illustrated the main features of "Speedy'O'Brain", a NFB videogame based on the analysis of EEG signal that has also been successfully used in a noisy environment such as that of an exhibition (Maker Faire 2018, Rome, Italy) attended by more than 100,000 visitors.

In the videogame, six puppets appear on a PC screen moving from the left to the right in a sort of race on a virtual horizontal line [Fig. 1]. Five of them are controlled (pseudo-randomly) by the PC while the sixth one, which is chosen by the user, by the user's EEG activity. The winner is the puppet that is at the rightmost position when a predefined amount of time is elapsed. This sort of pseudo race occurs for several times and after a calibration phase, which is necessary to train a simple linear regressor built from EEG spectral data computed over 1 second EEG windows.

The Speedy'O'Brain videogame supports also the NPXLab platform, that can be used to analyze the data [12].

The NPXLab Suite is a collection of tools and software modules - NPXLab was the first one to be released - developed to analyze EEG and MEG signals, even if it can be used for EKG, EMG, fNIRS and virtually any kind of sampled signal.

Among the provided processing capabilities, there are the Independent Component Analysis, the Common Spatial Patterns, several time domain filters, either IIR or FIR, and many linear and non-linear classifiers, which have been widely used in several Brain-Computer Interface systems.

Its name derives from the native NPX file format, which was defined to support Neurophysiological signal in XML [13].

After the first public presentation of the videogame, some research groups requested to obtaining the videogame, usually requiring also customization. Some users asked to modify the feedback modality, in order to have a more neutral kind of stimulation, some others to support different EEG devices, some others to cheat in order to avoid any kind of frustration of the users in case of frequent races loss.

II. Problem Formulation

To satisfy all the new requirements, a re-engineering of the videogame is necessary.

L. Bianchi is with the Civil Engineering and Computer Science Department of the "Tor Vergata" University of Rom, via del Politecnico, 1, 00133, Rome, Italy (phone: +39-06-7259-7461; e-mail: luigi.bianchi@ uniroma2.it).

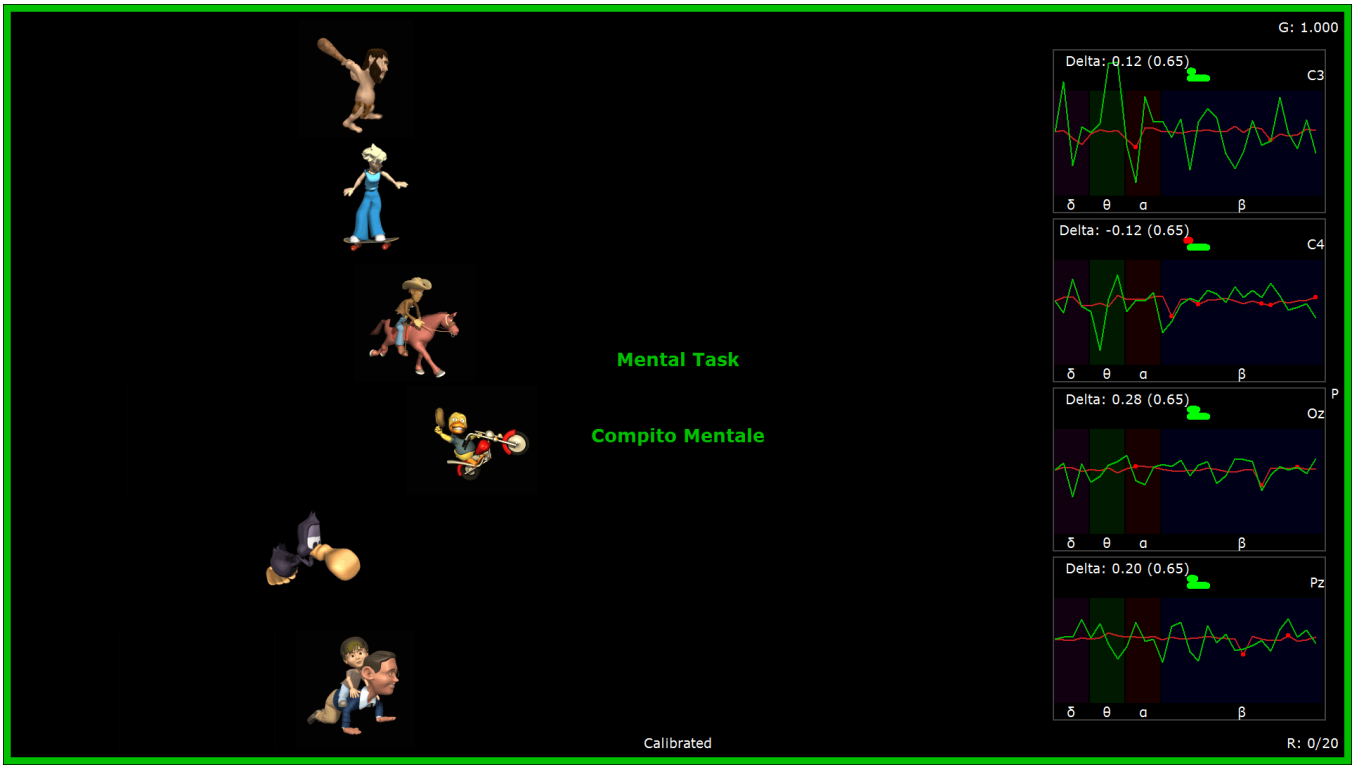

Fig. 1 - A screenshot of the videogame during the Racing Modality. The green surrounding frame indicates that the image is captured during the performance epoch. On the right part of the screen, features and spectra relative to the actual EEG activity are shown.

This falls into two main areas:

1. To support different platforms responsible to acquire brain signals (e.g. different EEG systems vendors or other signals such as MEG or fNIRS);
2. To support different and innovative NFB applications.

The first one was easily achieved because even the first release of the videogame was developed while keeping separated the acquisition device with the NFB videogame. In fact, two computers are necessary to run the NFB videogame, one for the EEG acquisition that was also responsible to transmit the acquired signals to the second computer through a TCP-IP connection and one for the regressor and the feedback. For this reason, the communication protocol needed only few minor changes to take into account a virtually unlimited number of different acquisition devices that have to just implement the transmission protocol to the second PC responsible to process and drive the feedback.

It should be noted that in this class of applications a hard real-time system is not necessary so that the non real-time TCP-IP protocol can be used. It is technically a brute force approach, which is however largely supported by almost every PC: estimated delays introduced by the various acquisition steps are of the order of milliseconds, as compared to the analyzed epochs whose duration is of the order of several seconds. In any case, even a LattePanda, which is a low-cost single board PC, running Windows 10, and equipped with an Intel Cherry Trail Z8350 Quad Core 1.8GHz CPU and with 4GB RAM was able to drive the regressor and the videogame. On the contrary, the supporting of several different NFB applications was more invasive and required some relevant changes. Even if the support to the NPXLab platform for the offline analyses was granted, it should be noted that the feedback relies on the ability to modulate and modify brain signals. This implies that to test different feedback modalities or regressors it is necessary to re-acquire the data: offline analyses cannot be performed to test other regressors because it is assumed that acquired signals are modified by the presence of the regressor used during the acquisition, whose effects cannot be easily removed from the acquired data. For this reason, to test a different regressor, feedback rule or modality, it is necessary to collect new data.

This requires time as well as the implementation of a new NFB system, which usually requires even more time.

III. Problem Solution

The BF++ platform [14] was used to implement the neurofeedback platform. It is a software framework written in C++ programming language to implement and analyze bio-feedback and BCI systems.

The idea behind the specific implementation is to capture all the main common aspects of a NFB application and build a hierarchy of object-oriented classes, which are progressively specialized to cover well-defined aspects of a specific implementation.

This should maximize source code reuse and dramatically reduce the time spent to customize or implement a new NFB application.

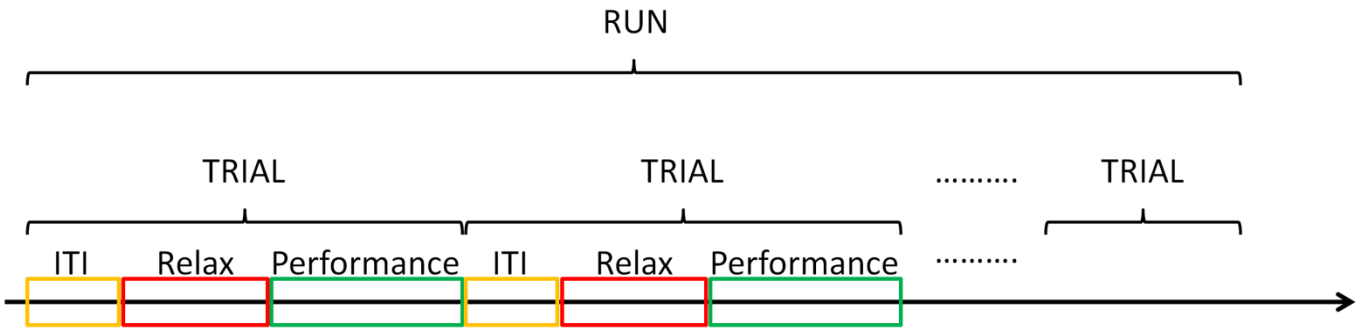

Fig. 2 - A Run is formed by a collection of trials and each trial is formed by a sequence of epochs. They are ITI, Relax and Performance in the example.

In short, there are usually four different working modalities: Exploring, Calibrating, Testing and Racing, each of them completed in a Run and composed of a set of Trials (Fig. 2). Each trial is then usually divided into three main epochs (this number, however, may be easily changed):

a) Inter Trial Interval (ITI), during which no data is processed and the user is free to move and stretch;

b) Relax, during which the user has to stay relaxed. This epoch is usually devoted to collect some reference data;

c) Performance, during which the user has to perform a mental task and whose data are usually compared to those of the previous relax epoch.

While the meaning of the epochs is the same for each operating modalities, the application handles brain signals in different ways, depending on the operating modalities, which usually are:

a) Exploring: a modality used to allow users to familiarize with the mental task. Users can try to perform several different mental tasks in order to find the one that is the most comfortable for them. Data from the relax epoch are compared to those of the performance and a *t*-test, on their respective spectral data is computed on each spectral sample and for each sensor to determine if the chosen mental task can be distinguished from the relax. Operators can also verify if the more responsive brain areas and frequencies are physiologically compatible with the mental task chosen by the user. For example, motor cortex should be selected while performing motor imagery. In this operating modality no feedback is provided to the user.

b) Calibrating: during this epoch, a regressor is trained in order to establish a correlation between the spectral data collected and computed during the relax and the performance epochs. For example, a multiple linear regressor can be trained by assigning a prediction value of 0 for the relax epoch and a value of 1 for the performance epoch. No feedback is provided to the user.

c) Testing: before the races, it is possible to see if the user is able to control the system. Feedback is provided to the user.

d) Racing: data are acquired during the Relax and the Performance epochs and the regressor law computed during the calibration phase is applied to one of the racers. The more a user can replicate the mental task executed during the performance epoch in the calibration phase, the faster the racer. Several races are performed in a run.

It should be noted that the number of operating modalities and epochs could be changed very easily and a programmer can set the desired values for each of them.

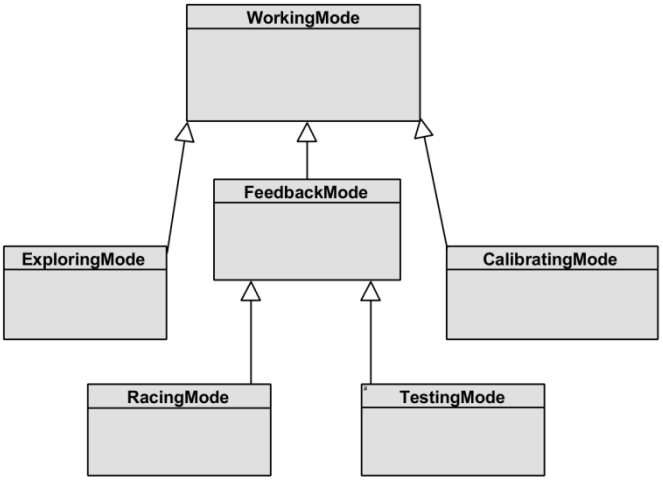

Fig. 3 The Operating Modalities Hierarchy.

The ancestor WorkingMode class holds two main objects, a Regressor, which is responsible to convert brain signals in a numeric value, and a UserInterface, which is responsible for converting the numeric value into a feedback to the user. There are two main classes derived from UserInterface: GUI (Graphic User Interface), for visual feedback and AUI (Acoustic User Interface) for acoustic one. They both share the same C++ interface.

Listing 1 shows the main methods of the WorkingMode class.

```
class WorkingMode
{
   public:
      WorkingMode (const char* name);
      virtual ~ WorkingMode ();

      const std::string& GetName() const;

      virtual int Setup(UserInterface* p_ui);
      virtual void Reset();

      virtual int SetSR(uint32 sr_);
```

```
        uint32 GetSR() const;

        virtual int PrepareTrial();

        virtual bool IsEndOfTrial() const;

        uint32 GetMaxTrialDurationSmpl();
        virtual bool IsTrialCompleted() const;
        virtual bool IsRunCompleted() const;

        virtual void ResetCount();

        int GetEpochIndex(int smpl) const;
        int GetEpochsCount() const;
        int AddEpoch(const MyEpoch& ep);

        const MyEpoch& Epoch(int i) const;

        uint32 GetBeginEpochSmpl(int epoch_index);

        int GetCurrEpochIndex() const;

        virtual void SetUserInterface(UserInterface* u);
        virtual UserInterface* GetUI();

        static void SetRegressor(Regressor* ptr_pred);
        virtual Regressor* GetRegressor();

        virtual void ResetUI();

        virtual void OnExitMode();
        virtual void OnEnterMode();

    protected:
        virtual void OnRunEnd();

        virtual void OnTrialBegin();
        virtual void OnTrialEnd();

        virtual void OnEpochBegin(int epoch);
        virtual void OnEpoch(int epoch);
        virtual void OnEpochEnd(int epoch);

…..

    public:

        static uint32 s_EnterModeDelaySmpl;
        static uint32 s_EnterModeElapsedSmpl;

    protected:
        uint32 m_nTotalTrials;
        uint32 m_nCurrTrial;

        std::vector<MyEpoch> m_vEpochs;
        std::string m_strName;

        static Regressor * s_pRegressor;

        UserInterface *m_pUI;

        uint32 m_SR;
};
```

Listing 1 – The WorkingMode C++ class declaration.

A clock object is also responsible of triggering all the timing events (e.g. OnEpochBegin, invoked at the beginning of an epoch), and to notify them to the various objects such as the UserInterface and the Regressor, that can execute the desired actions.

This triggering feature is provided thanks to the EpochScheduler class, which acts as a concertmaster as it holds a clock and determines when an epoch is elapsed and then a new one should begin as well as if a trial is completed and so on.

Actually, there are two main derived classes to handle different situations:

1) The NFB application determines the events, usually by updating a clock every time a data packet is received. This is the usual case in real-time applications;

2) The NFB application receives the clock info from an external process, as in the case of data playback, that is when the videogame is driven from an already acquired file to replicate the online behavior. In this case, events stored in a file (e.g. “ITI Begin”) are used to determine and trigger the proper epochs timing and events.

In this way, by simply changing the epoch scheduler one can transform, with just one line of code, an online NFB application into a playback one.

It is then clear that the main strategy is to simply derive a class from an ancestor one and customize the behavior of the NFB application by overriding functions whenever necessary, with a minimal effort.

This is also true for the implementation of the various regressors: at the time of this writing, four main different regressors families (multiple linear regressor, artificial neural network, stepwise linear discriminant analysis and Lasso shooting) have been implemented and a user can choose which one to use. All of them are actually fed by spectral data, computed through classical FFTs procedures.

In any case, a new regressor can be easily implemented by overriding a few member functions.

Listing 2 illustrates the main functions of the FFTRegressor class, from which the 4 aforementioned regressors derive. In the listings, Mat is a Matrix class and MyFeature is a class that represents a feature holding information regarding its regression weight, and the channel and frequency to which it refers.

```
class MyFFTRegressor : public MyRegressor
{
public:
    MyFFTRegressor(const char* sz_name);
    virtual ~MyFFTRegressor();

    virtual int Train (Mat& bl, Mat& perf, double res);
    virtual double GetScore(const RowVector& rw);
    virtual int SetSR(uint32 sr);

    int SetFFTLen_ms(uint32 length);
    uint32 GetFFTLen_ms() const;

    virtual void OnEpoch(int epoch);
    virtual void OnEpochEnd(int epoch);

    virtual int Reset();

    virtual bool IsTrained();
    virtual size_t GetTrainedInfo(ostringstream& oss);

    double GetDelta(const WorkingMode * mwm);

    const list<MyFeature>& Features() const;
protected:
    Mat m_vSpeHistCalibBL;    //
    Mat m_vSpeHistCalibPerf; //
    Mat m_p_values;

    fftw_complex* m_pFFTW_In;
    fftw_complex* m_pFFTW_Out;
    fftw_plan m_FFTWPlanFwd;

    std::list<MyFeature> m_lstFeat;
};
```

Listing 2 – The FFTRegressor C++ class declaration.

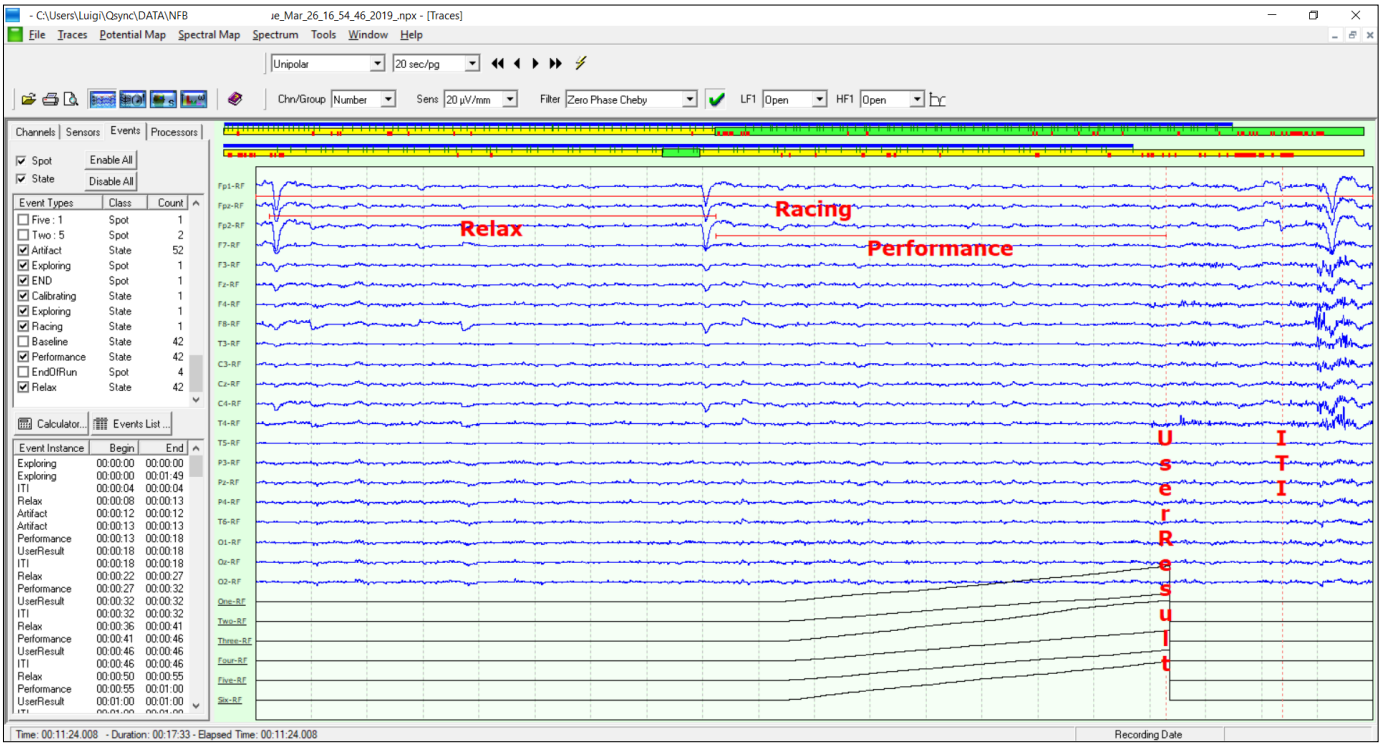

Fig. 4 – A file generated with the Speedy'O'Brain software re-opened with the NPXLab tool for the off-line reviewing and analysis. Relax and Performance epochs are clearly identifiable. The last six rows represent the positions of the six racers.

To give an idea of the oversimplification of the programming task, it should be considered that to implement the RacingMode class it has been necessary to write less than 300 lines of code, most of them devoted to the handling of the feedback, overriding just 11 functions whose names are very self-explicative such as:

1) PrepareTrial(), which is triggered in order to prepare a new trial and perform som randomization such as the speed of the runners controlled bey the computer in the case of the Speedy'o'Brain videogame.

2) OnEpochBegin(), used to reset data buffers, timers, counters and update the user interface;

3) OnEpochEnd(), used to trigger events to regressors and user interface;

4) OnRunEnd(), used to perform final analyses, such as training classifiers in CalibratingMode or present final results to users when in RacingMode;

Therefore, if one has to show the performances at the end of a session he has just to override the OnRunEnd function: the internal model automatically triggers all the events and calls the proper functions. Even the various regressors were implemented by overriding just 4 member virtual functions. As an example, the multiple linear regressor was implemented with less than 150 lines of code.

In practice, most neurofeedback applications share the same working modalities, so that usually there is no need to implement a new one: usually, only the user interface needs to be implemented, which is a very simple task.

Listing 3 illustrates the interface of the MyRacingGUI class, which is devoted to the visualization of the various phases of the race and responsible to animate, move and show the results of the various puppets at the end of each race and at the end of a session, formed by a set of races whose results are summed and notified to the users at the end of a run.

```
class MyRacingGUI : public MyGUI
{
public:
  MyRacingGUI();
  virtual ~MyRacingGUI();

  virtual void OnRunEnd(WorkingMode* m);

  virtual void OnEpochBegin(WorkingMode* m, int epoch);
  virtual void OnEpoch(WorkingMode* m, int epoch);
  virtual void OnEpochEnd(WorkingMode* m, int epoch);

  virtual void OnExitMode(WorkingMode * m);

};
```

Listing 3 – The MyRacingGUI C++ class declaration.

The meaning and functionalities of the member functions are quite self-explicative: in practice, one has to define what happens when the various epochs begin, end and during them. These functions are automatically invoked by the scheduler so that a programmer has just to override them to customize the behavior of the NFB application.

Finally, I should be mentioned that a data file in NPX Format is also automatically generated with all the events (e.g. "ITI", "EndOfRun", "Calibrated", etc…) and feedback entities (e.g. racer position) stored into it.

In addition, calibration parameters and regressors laws are

also stored in a NPX file. All this information can be further processed and reviewed with all the tools of the NPXLab Suite. Finally, it is possible to "playback" a stored file as if it were processed in real-time to help programmers to debug their own implementations. A screenshot of NPXLab with a file generated with Speedy'O'Brain loaded is shown in Fig. 4.

IV. Conclusion

Speedy'O'Brain is a neurofeedback videogame that has gained a lot of interest after its first large audience public presentation. Because lots of researchers and users were interested in adapting it with minimal changes according to their needs, it has been re-engineered according to a well-defined BCI model [14] that has already been used in several BCI implementations [15, 16, 17, 18]. The model has also been considered as a starting point in a proposal for BCI standardization [19].

The fact that it automatically integrates with all the dozens of tools of the free and publicly available NPXLab Suite make it an easy a ready to use platform for implementing a wide range of neurofeedback applications.

Future works will include multiplayer support and the release of more regressors together with the compatibility with a larger set of hardware brain signals acquisition devices.

V. AWARDS

Speedy'O'Brain was awarded at the Maker Faire 2018, European Edition, held in Rome, Italy 13-15 October 2018 with the Maker of Merit Blue Ribbon.

VI. ACKNOWLEDGMENT

The author would like to thank Micromed S.p.A. and EBNeuro S.p.A. manufacturer for the assistance and the support they provided to realize the communication protocols. A special thank goes to dr. Federica Cinelli for her invaluable support and for making it possible to acquire good quality EEG signals in a very hostile environment, such as that of an exhibition.

References

[1] Johnstone SJ, Roodenrys SJ, Johnson K, Bonfield R, Bennett SJ. "Game-based combined cognitive and neurofeedback training using Focus Pocus reduces symptom severity in children with diagnosed AD/HD and subclinical AD/HD". Int J Psychophysiol. 2017 Jun;116:32-44.

[2] Bakhtadze S, Beridze M, Geladze N, Khachapuridze N, Bornstein N. "Effect of EEG Biofeedback on Cognitive Flexibility in Children with Attention Deficit Hyperactivity Disorder With and Without Epilepsy." Appl Psychophysiol Biofeedback. 2016 Mar;41(1):71-9.

[3] Marzbani H, Marateb HR, Mansourian M, "Neurofeedback: A Comprehensive Review on System Design, Methodology and Clinical Applications", Basic Clin Neurosci. 2016 Apr;7(2):143-58

[4] Wenya Nan, Feng Wan, Lanshin Chang, Sio Hang Pun, Mang I. Vai, Agostinho Rosa. "An Exploratory Study of Intensive Neurofeedback Training for Schizophrenia", June 2017Behavioural neurology 2017(2):1-6

[5] Alves-Pinto A, Turova V, Blumenstein T, Hantuschke C, Lampe R. "Implicit Learning of a Finger Motor Sequence by Patients with Cerebral Palsy After Neurofeedback.", Appl Psychophysiol Biofeedback. 2017 Mar;42(1):27-37

[6] Reichert JL, Kober SE, Schweiger D, Grieshofer P, Neuper C, Wood G. "Shutting Down Sensorimotor Interferences after Stroke: A Proof-of-Principle SMR Neurofeedback Study", Front Hum Neurosci. 2016 Jul 15;10:348.

[7] Renton T, Tibbles A, Topolovec-Vranic J. "Neurofeedback as a form of cognitive rehabilitation therapy following stroke: A systematic review.", PLoS One. 2017 May 16;12(5):e0177290.

[8] Kober SE, Schweiger D, Witte M, Reichert JL, Grieshofer P, Neuper C, Wood G. "Specific effects of EEG based neurofeedback training on memory functions in post-stroke victims.", J Neuroeng Rehabil. 2015 Dec 1;12:107.

[9] Bennett CN, Gupta RK, Prabhakar P, Christopher R, Sampath S, Thennarasu K, Rajeswaran J "Clinical and Biochemical Outcomes Following EEG Neurofeedback Training in Traumatic Brain Injury in the Context of Spontaneous Recovery". Clin EEG Neurosci. 2018 Nov;49(6):433-440.

[10] Jeunet C, Glize B, McGonigal A, Batail JM, Micoulaud-Franchi JA. "Using EEG-based brain computer interface and neurofeedback targeting sensorimotor rhythms to improve motor skills: Theoretical background, applications and prospects." Neurophysiol Clin. 2018 Nov 7. pii: S0987-7053(18)30259-4.

[11] Bianchi L (2018), "Speedy'O'Brain: a Neuro-Feedback Videogame driven by Electroencephalographic Signals", International Journal of Biology and Biomedical Engineering, Vol. 12, pp 229-234.

[12] Bianchi L, "The NPXLab Suite 2018: a Free Features Rich Set of Tools for the Analysis of Neuro-Electric Signals", WSEAS Transactions on Systems and Control, ISSN / E-ISSN: 1991-8763 / 2224-2856, Volume 13, 2018, Art. #18, pp. 145-152

[13] Bianchi L, Quitadamo LR, Marciani MG, Maraviglia B, Abbafati M, Garreffa G. (2007). "How the NPX data format handles EEG data acquired simultaneously with fMRI". Magnetic Resonance Imaging. vol. 25(6), pp. 1011-1014 ISSN: 0730-725X.

[14] Bianchi L, Babiloni F, Cincotti F, Salinari S, Marciani MG. (2003). "Introducing BF++: A C++ framework for cognitive bio-feedback systems design". Methods of Information In Medicine. vol. 42, pp. 104-110 ISSN: 0026-1270.

[15] Bianchi L, Ferrante R, Quitadamo LR. (2018). "Analog-like control is possible in SSVEP based brain-computer interfaces". 2018 IEEE Life Sciences Conference, Montreal, Canada; 28 – 30 October 2018, Article number 8572094, Pages 235-238.

[16] Lugo ZR, Quitadamo LR, Bianchi L, Pellas F, Veser S, Lesenfants D, Real RGL, Herbert C, Guger C, Kotchoubey B, Mattia D, Kübler A, Laureys S, Noirhomme Q. "Cognitive processing in non-communicative patients: What can event-related potentials tell us?". (2016). Front. Hum. Neurosci, Vol. 10, 14 November 2016, Article number 569.

[17] Bianchi L, Sami S, Hillebrand A, Fawcett IP, Quitadamo LR, Seri S, " Which Physiological Components are More Suitable for Visual ERP Based Brain--Computer Interface? A Preliminary MEG/EEG Study", (2010), Brain Topography, Volume 23, Issue 2, pp 180–185.

[18] Bianchi L, Quitadamo LR, Abbafati M, Marciani MG, Saggio G. "Introducing NPXLab 2010: A tool for the analysis and optimization of P300 based brain-computer interfaces", (2009), 2nd International Symposium on Applied Sciences in Biomedical and Communication Technologies (ISABEL), 24-27 Nov. 2009, Bratislava, Slovakia.

[19] Bianchi l. (2018), "Brain-computer interface systems: Why a standard model is essential. On BCI standards". 2018 IEEE Life Sciences Conference, Montreal, Canada; 28 – 30 October 2018, Article number 8572142, pp. 134-139.

Luigi Bianchi was born in Milano, Italy, in 1965. He was graduated in Electronic Engineering at the University of Rome "La Sapienza", with a thesis on the analysis of dipolar sources localization techniques from EEG recorded activity in man. In 2005 he got the Ph.D. in "Neurophysiology: Neural Basis of Superior Cognitive Functions" (University of Rome "La Sapienza") with a thesis entitled: "Brain-Computer Interface Systems: Design, Algorithms and Optimization". From 1993 to 1994 he worked at the University of Pierre e Marie Curie (Paris, France). From 1994 to 1995 he worked at the University of Rome "La Sapienza" in the Human Physiology Laboratory cooperating with the Institute of Neuroscience and Bio-Images of the CNR (National Research Centre) of Milan (Italy). From 1996 to 2011 he worked in the Nerophysiopathology Laboratory of the University of Rome "Tor Vergata". From 1998 to 2006 he was a consultant of the Research and Development

Division of the EBNeuro S.p.A., one of the largest European manufacturers and distributors of Neurophysiology instruments, for which he developed, among the others:
1) The first commercial European Digital Video EEG system;
2) The first release of the Galileo EEG system for Windows;
3) The Spectral Analysis Package, which includes 3-D spectral maps, spectral coherence computation and many other facilities;
Since 2011 he is Assistant Professor at the University of Rome "Tor Vergata" (Civil Engineering and Computer Science Engineering Dept.), where he actually works. He is the author of more than 100 peer-reviewed scientific publications (more than 4500 citations) and he is in the editorial board of several scientific journals. His h-index is 34 (Source: Google Scholar, November 2018). He also participated in several EU research project. His main interests are real-time and offline signal processing, Brain-Computer Interfaces, Human-Computer Interaction and assistive technologies and released many free tools which can be downloaded from his web site http://www.brainterface.com. Prof. Luigi Bianchi is a member of the IEEE society and of the BCI society. In 2002 he was awarded at the II° Brain-Computer Interface Workshop (Albany, NY) and in 2018 he was awarded at the MakerFaire 2018, European Edition.