Development of A Biofeedback Enhanced Multimedia Game

Huaiwei Wu, Chenguang Yang*, Ruowei Wang and Chun-Yi Su

***Abstract*— As a frontier technology, biofeedback has been widely used for psychological treatment, control and many other human-computer interaction applications. In this paper, a multimedia game based on biofeedback technology is developed, including two main parts, the design of a brain computer interface (BCI) and the design of a multimedia game. In the BCI design part, eletroencephlogram (EEG) signals are collected and sent to a processing module for classification. All EEG signals will be classified into two categories: movement or relaxation. Subsequently these classified results will be sent to the game process to control the action of a virtual car in real time. In the second part, the design of a multimedia game via QTCreator is presented. We mainly use QTCreator's GraphicsView Frame to develop the game. At last we will introduce how to combine MATLAB, SCAN4.5 and QTCreator together to play the game with biofeedback.**

I. Introduction

With the rapid modernization process, our pace of life speeds up obviously. Which has sparked serious psychological problems. According to the data provided by the Ministry of Health, the rate of severe mental illness has risen from 2.7% in 1950s to 13.47% nowadays, and the number of mild patients are countless. But the reaction of our society is not enough at all. We still lack professional treatment for mental health as well as some adjunctive therapies which the patients can easily practice at home. One of the most effective ways for adjunctive therapy is to use the technology of biofeedback [1]. It is helpful for remote health monitoring [2], stress reduction [3], pains reduction, regulating heart rate, treating hypertension and so on. But the treatment is usually boring [4], asking the patients to repeat some boring actions so that they will lose their interest very soon and therefore undermine the effect to a certain extent in the treatment. For improvement, we can combine this technology with interesting things to attract users' attention to bring better effect on the treatment.

Among the biofeedback technologies, brain computer interface (BCI) is one of the most popular and mature methods [5] [6]. It can reflect our emotions most directly [7]. Besides, BCI provides us with a direct communication pathway between our brain and the peripheral. Through BCI, we do not need to express our opinions by our hands or legs firstly. The command from our brain can be executed directly to play games, drive a wheelchair [8] and control a robot without the participation of our hands or legs; the disabled persons can repossess their "legs" or "hands" to stand up, walk and manipulate complex equipments [9]. For the reasons above, we will employ BCI in our study to the biofeedback video games design.

Computer games are very popular among the masses especially the young people in our scoeity [10]. In games, we can relax, have fun and even make friends. So we can combine the biofeedback technologies with some funny games to improve its effect. For game development, there are many softwares available. For example, MATLAB, Visual Studio (VS) and QTCreator. Among them, MATLAB has a powerful computing ability and VS has a large number of program libraries. But for interface development, QTCreator is the best choice. Firstly, it has a powerful framework of graphics view which will help us to effectively deal with complex relationship among the graphic items in the game. Secondly, it has an unique ability on cross-platform so that we can easily extend the games to other platforms in the future. Therefore, QTCreator will be the game development software in this paper.

The structure of the biofeedback enhanced multimedia game is illustrated in Fig.1. At first, electroencephalograph (EEG) signals are collected by Neuroscan device and sent to SCAN4.5 which is the matching software for the Neuroscan. SCAN4.5 provides a TCP port where MATLAB can get EEG data in real time if we have completed a MATLAB TCP client. In MATLAB, we can process the raw EEG data, extract the features and then classify them by using its powerful computing capability. After that we send the classified results to QTCreator via UDP sockets. The game process designed in QTCreator will respond to these results.

In the following sections, we will present the system as illustrated in Fig.1 in turns.

II. The Design of BCI

A. ERS/ERD Phenomenon and Signal Collection

It seems that our our brain works in a complex manner, but there are still many clues to infer our mind. Indeed, the human brain is made up of billions of nerve cells. They always communicate with each other which are indicated on the scalp by changing potentials. Different intentions will cause different potential distributions on the scalp. For example, when we want to move some part of our body, the energy level of EEG signals on the specific position of the

This work was partially supported by National Nature Science Foundation (NSFC) under Grant 61473120, Guangdong Provincial Natural Science Foundation 2014A030313266 and International Science and Technology Collaboration Grant 2015A050502017, and the Fundamental Research Funds for the Central Universities under Grant 2015ZM065.

The authors are with the MOE Key Lab of Autonomous System and Network Control, School of Automation Science and Engineering, South China University of Technology, Guangzhou, 510641 China. C. Yang is also with Zienkiewicz Centre for Computational Engineering, Swansea University, UK. C.-Y. Su is on leave from Department of Mechanical and Industrial Engineering, Concordia University, Canada

*Corresponding author. Email: cyang@ieee.org

978-1-4673-9714-8/16/$31.00

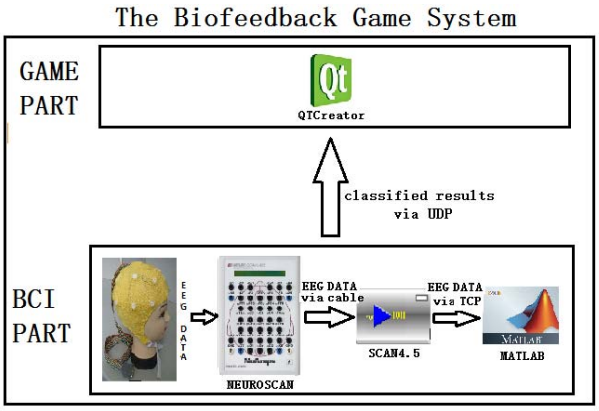

Fig. 1. System Overview

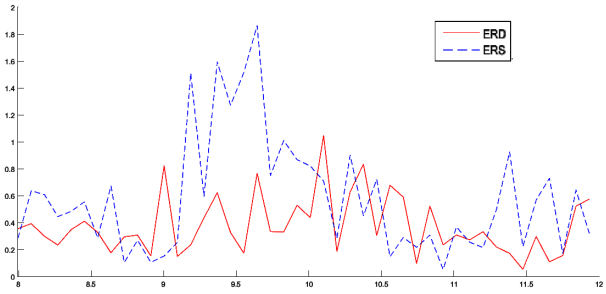

Fig. 2. ERS/ERD Phenomenon

scalp will reduce. This is called the Event Related Desynchronization (ERD) phenomena. On the contrary, when we stop imagining to move, the energy level will rise which is called the Event Related Synchronization (ERS) phenomena. Both phenomenon perform obviously in α rhythm and β rhythm. They are shown in Fig.2. According to this different performance in energy level, we can classify EEG signals into two categories, imaging or relaxation. This technique is called motor imagery [11] and there are many methods to classify the different patterns in motor imagery [12] [13].

For EEG signal collection, the Neuroscan, with high resolution, high speed, multi-channel and 40 sensors, is a good signal amplifier. We use it to collect the raw EEG data on the scalp. In the computer, the raw data must be received by the matching software of Neuroscan named SCAN4.5 firstly. Then it will be sent to MATLAB for processing via TCP socket.

B. Algorithm for Pretreatment and Feature Extraction

As there are many noisy signals mixing in the raw EEG signals, pretreatment is needed. The most important task is to filter out the signal in the band of α rhythm and β rhythm where the ERS/ERD phenomenon performs obviously. There are two ways for pretreatment. Firstly, we can set up both low pass and high pass frequency of a built-in filter in SCAN4.5. Secondly, we can use MATLAB's filter functions.

After pretreatment, we will extract the feature of the signal by employing the Common Spatial Patterns (CSP) algorithm. The main idea of this algorithm is to construct a converter which can maximize the variance of one kind of signal and minimize the other one so that we can distinguish them more obviously. To complete this algorithm, we calculate the covariance of the EEG signals matrix firstly. The formula is shown as follow:

$$C = \frac{EE^T}{\text{trace}(EE^T)} \tag{1}$$

where the matrix $E \in R^{N\times T}$ means the data of EEG signals with N represents the number of electrodes and T represents the number of sampling points in a collection. "trace()" means to calculate the sum of diagonal elements of a matrix.

Using this formula, we compute the covariance of both left hand motor imagery and left hand relaxing data in n times, denoted as C_{li} and C_{ri}. Then we can calculate the average of them:

$$C_l = \sum_i^n C_{li}/n \tag{2}$$

$$C_r = \sum_i^n C_{ri}/n \tag{3}$$

where C_l means the the average of all C_{li}. C_r means the average of all C_{ri}. Then, we should calculate the covariance of the mixing space:

$$C_c = C_l + C_r \tag{4}$$

where C_c means the covariance of the mixing space.

Secondly, we decompose the feature of the matrix C_c:

$$C_c = U_c \times A_c \times U_c^T \tag{5}$$

where we can get two matrices, the feature vector matrix U_c and the eigenvalue matrix A_c of the matrix C_c.

Thirdly, we do the whiten transform of the matrix C_c:

$$P = A_c^{-0.5} \times U_c^T \tag{6}$$

$$S_l = P \times C_l \times P^T \tag{7}$$

$$S_r = P \times C_r \times P^T \tag{8}$$

Then, we decompose the feature of the matrix S_l and S_r.

$$S_l = B \times A_l \times B^T \tag{9}$$

$$S_r = B \times A_r \times B^T \tag{10}$$

Now, the desired converter is shown below:

$$W = (B^T \times P)^T \tag{11}$$

C. Algorithm for Classification

For classification, we employ the Linear Discriminant Analysis (LDA) algorithm. It is a simple and stable classifier. Its main idea is to construct a classification criterion base on the known samples and then classify new samples according to this criterion.

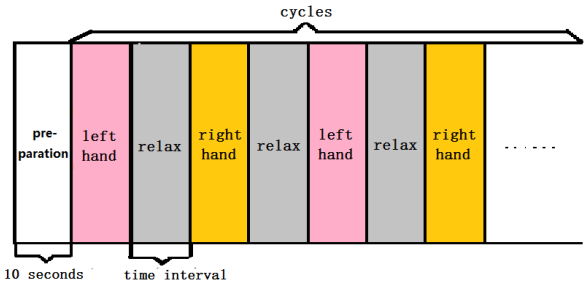

Fig. 3. Training pattern

Fig. 4. Training window(screen snap)

The expression of LDA is shown below

$$D(x) = b + \omega^T \times x \tag{12}$$

where 'x' means the new sample, 'b' and 'w' are parameters computed from known samples and '$D(x)$' is the result of classification. If $D > 0$, it means x is one kind of signal and $D < 0$ means the other kind.

D. Experiment of the Off-Line BCI

Based on the algorithms mentioned above, now we can establish the off-line BCI. The training pattern for the experiments is shown in Fig.3. At the first stage, the subject has 10 seconds to make preparation. Then, he is supposed to perform motor imagery of his left hand. At the next interval, he stops imagining. Then, he should perform motor imagery of his right hand and then stops and repeats this cycle.

After we set up the training pattern, a problem rises concerning how we can guarantee the synchronism between the off-line EEG data and training intervals. Simply speaking, after we have finished a training experiment and get an off-line file in which the EEG data is recorded, can we know the exact correspondence between each interval and its EEG data? To solve this problem, we have created a training window by QTCreator, as shown in Fig.4

In this window, we can set parameters for the training such as time interval, cycle index and so on. More importantly, it can communicate with SCAN4.5. If we click the button of "start experiment", it will send commands to SCAN4.5 to let it start collecting and recording EEG data. Once we have completed the training, it will send commands to SCAN4.5 and let it stop collecting. This has guaranteed the synchronism between the data file and the training patterns at the beginning and ending moments. Then, we can calculate each interval's EEG data based on the time interval, cycle index and the sampling rate which we set before training.

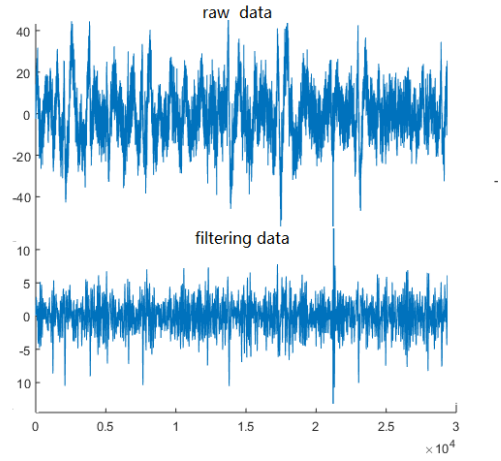

Fig. 5. Comparison of the raw EEG data and the filtering data

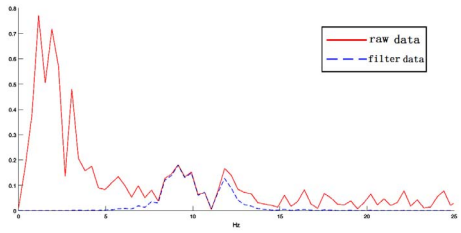

Fig. 6. The spectrum of the raw data and the filtering data

After all these preparations, the experiment subjects just need to sit in front of the monitor and look at the training window to follow the instruction given by the window. After an experiment, we will get an off-line file. From the file, we can get EEG data and process them via the aforementioned algorithms. The experiment results will be shown in the next section.

E. Experiment Result

After we finish an experiment, we get an off-line EEG data file.At first we load the raw data from the file and make pretreatment of it to filter out a certain range of frequency between 8 to 12Hz. The raw data and the filtering data are compared in Fig.5. From time domain, we can hardly see the difference between them. But if we visualize them in frequency domain, we can distinguish them more easily as shown in Fig.6. Then, we can make CSP transform of the filtering data and extract the features and classify them. The classified results of an experiment is shown in Fig.7. In the figure, the red circle means motor imagery and the blue 'x'

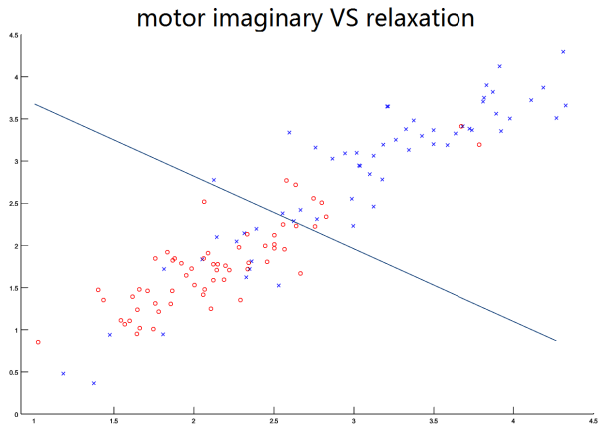

Fig. 7. The classification of two signals

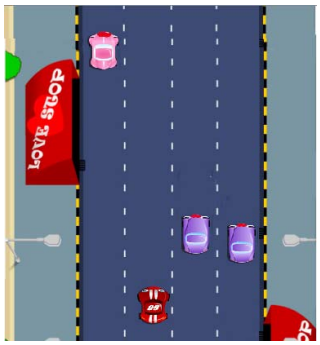

Fig. 8. The interface of the multimedia game

means relaxation. As the Fig.7 shows, the accuracy rate of these algorithms is up to 81.3%.

III. The Design of Multimedia Game

A. Design of the Game Rule

The interface of the multimedia game is illustrated in Fig.8. The images of the user's car, obstacle cars and background are all downloaded from the Internet [14].

There are four lanes in the scene in this game. The red car on the bottom of the scene is user's car. And the other cars are the obstacle cars. They will appear in the scene randomly. At the beginning, the speed of the user's car is 0 m/s. Then it accelerates automatically and catches up with the obstacle cars. If the player is too late to change lanes, it will collide with them. Then its speed will go back to 0 and has to accelerate again.

We have defined two important events about collision in the game. The first one is the event of collision which we has mentioned above. It happens when the user's car collides with the back of the obstacle cars. In this situation, the user's car will stop and the obstacle car will be hit, flying some distance ahead. The second is the event of cutting in. It happens when the user's car runs parallel with the obstacle car and then changes lane to the obstacle car's lane. In this situation, the obstacle car has to give way to the user or will be destroyed. And the speed of the user's car will increase. The second event will increase the interest of this game.

B. Design of the Game Framework

After we determine the game rules, we design the framework of the game programs. As we mentioned in section I, QTCreator provides us with a powerful graphic view framework for the game design. In its framework, there are mainly three important components, which are QGraphicsScene, QGraphicsView and QGraphicsItem. Giving an analogy to introduce them, QGraphicsScene is like an empty house where we can add a lot of "furniture" (QGraphicsItems). It is not just a simple house. It knows every member's coordinates, contour, running state and so on. It also equips with a "broadcasting system". Once some important events happen, such as collision and click of the mouse, it will inform all members or just some specific members inside it. The QGraphicsView is similar to a window of the house. Through this window, we can observe the items in the QGraphicsScene locally or globally. We can also set the size and transparency of this window. The QGraphicsItem is the furniture in this house. For example, in this game, the user's car, the obstacle cars and the background picture are all QGraphicsItems. It provides us with some basic functions to draw the shape of the items, to move the items, to detect the edge and collisions and so on.

Based on this framework, we make the game scene the center of our frame because it can manage all the graphic items contained in it easily. The graphic view, which may be regarded as the windows in the window system, is used to observe the game scene from its perspective. The obstacle cars are designed into a linked list structure for better management. The background picture is also an item of the scene. It moves in the screen quickly or slowly according to the speed of the user's car so that we will have an illusion that the car is going forward which actually always stays at the bottom of the scene. The details will be introduced in next section. This framework is shown in Fig.9.

C. Realization of Basic Game

To visualize the game design, we need to draw the graphic items such as the user's car, obstacle cars and background picture in the scene. In this step, we use the member function in the QGraphicItems named paint(). Using this function, we can use QT's painter to draw any shapes or load a picture to draw a item.

In the next step,we use the moveby(x,y) function to move the items. The parameters in the function's bracket means the item's displacement in x and y directions. To move the items automatically in real time, we can use QT's timer. The basic mechanism of a QT's timer is as follow: we firstly set the interval for the timer and run it. The timer will run and when it finishes an interval, it will execute the timeout function and then start the next interval repeatly. So we need to set time interval and write the "moveby()" function into

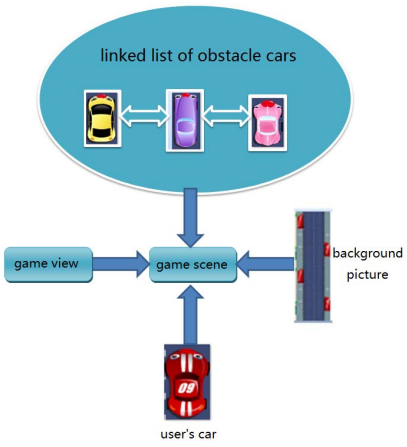

Fig. 9. The frame of the multimedia game

the timeout function of the timer. Then the items will move in the screen automatically at a specific speed. If we want to change the speed, we can change the interval of the timer or the parameters in the moveby() function.

Next, we consider the velocity mechanism of this game. There are mainly two kinds of speed in the game, the speed of the user's car and the obstacle cars. For obstacle cars, they move forward or backward in the screen according to their speeds. If they run outside of the view, they disappear. However, for the user's car, it can't disappear in the screen. Actually it always stays at the bottom of the scene. Instead of moving the user's car, we move the background picture in the opposite direction to create the illusion that the user's car is running ahead. But if we keep moving the background picture during the game, we need to find a long picture. It is hard to find one and will be a limit to the total game time as when the picture ends, the game has to stop. So we make a cycle in the background movie. Firstly we find a background picture and stick a copy to the top of it. Then we let the viewport rise along the new picture. Once the viewport's bottom edge line arrives at the initial background picture's top edge line, it will return to the initial background picture's bottom edge line, and goes on this cycle.

Now we introduce how to calculate the speed of the background and the obstacle cars relative to the screen. The formulas are as follows:

$$V_{bs} = -V_{ua} \tag{13}$$

$$V_{os} = V_{oa} - V_{ua} \tag{14}$$

where V_{bs} and V_{os} denote the velocities of the background and the obstacle cars relative to the screen. They are their actual moving speeds in the screen. V_{ua} and V_{oa} are the velocities of the user's car and the obstacle cars in theory.

Next, we translate these speeds into time interval in QT's timer. The formula is shown below:

$$I = \alpha \times \frac{1}{V} \tag{15}$$

where I is the interval of the timer and V is the object's speed relative to the screen and α is a variable coefficient. By increasing α, we can increase the difficulty of the game.

D. Realization of Collision Detection and Multimedia

One of the advantages to use QTCreator as our developing platform is that it has an efficient mechanism to process collisions among graphic items. Once we add an item such as the obstacle car into the game scene, QT will supervise its statement automatically. If it collides with the other items in the scene, QT will turn to execute the collision processing functions immediately. Indeed, these functions are QT's "slots" functions. They can be correlated with "signal" functions to deal with the signal events such as the collisions and the input of keyboard. In this case, the event is collision. To achieve the collision detection and management, we need to add the items into the scene and integrate collision managing into the "slot" functions in QTCreator, having no worry about how to detect whether the item has collided with the others. It will be implemented automatically. As a result, this mechanism makes our program more readable and simpler.

For better game experience, we can also add some background music to the game. To achieve it, we can use another QT's class named Phonon. Firstly, we create a new Phonon class variant. Then we set the output way and the path of background music. After that, we command the music to play.

E. Control the Game Using Biofeedback

In this section, we will complete our target that we can play the multimedia game by using biofeedback technology. In the previous processes, we have finished the design of BCI and multimedia game by some softwares. Now the core issue is how to combine these softwares together which are MATLAB, QTCreator and SCAN4.5. Among them, SCAN4.5 is used for collecting EEG data from Neuroscan, MATLAB is for data processing and classifying and QTCreator is for game design. As Fig.1 shows, the EEG data is firstly sent from SCAN4.5 to MATLAB, and the classification results are sent from MATLAB to QTCreator. So our task is to build up these two communication pathways.

As mentioned in Section I, SCAN4.5 provides a TCP port for us to acquire EEG data in real time. In the specification of SCAN4.5, there is a detailed description of the TCP data packet. The structure of a packet is shown in Fig.10. From the picture, we see that the packet consists of two parts, the header part and the data part. There are four members in the header: The "bodysize" means how many bytes the data part will own in this packet; The combination of the other three members with different values represents different commands between MATLAB and SCAN4.5. We can use these commands to make SCAN4.5 start collecting data, send data, stop collecting and so on. Thus, our work is to write a MATLAB TCP client program according to the agreement provided by SCAN4.5. Upon writing the client program, two important issues must be considered. One is how to receive

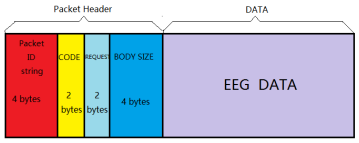

Fig. 10. the structure of the TCP packet

and analyse the header precisely. This issue will cause a series of errors upon failure because the data stream of TCP is continuous. The other one is how to match the speed between the sender and receiver because we are not able to receive the data too fast or too slow. To solve this problem, we'd better use MATLAB timer and find out the best timer interval.

To build up the communication channel between QTCreator and MATLAB, there are several ways. One is to use MATLAB engine. This is stable but too slow and will undermine user's experience in the game. Another choice is to use dynamic link library which is faster, but dependent on the edition of MATLAB and QTCreator very much. If we change our edition of the software, it may cause many errors. In this paper, we employ a new way by using UDP socket as the channel between these two softwares. The reasons are as follows. Firstly, it is stable and fast and does not depend on the edition or the operation system at all. Two software processes can run in parallel in the system without any interference. Secondly, in comparison to TCP, UDP is much simpler and faster, such that we don not need to establish connections between two terminals. Thirdly, we don not have to worried about the problem of data packet dropout because we only use the local network communication in one computer. To establish this channel, we first bind a specific port of UDP in MATLAB and in QTCreator. For example, MATLAB binds at '6667' and QTCreator binds at '6666'. Then, if MATLAB wants to send classification results to QTCreator, it can send data to the UDP at port '6666' with the function "frwite(UDP,data)" in MATLAB. QTCreator will supervise the port in real time and once MATLAB has sent the results, QTCreator will receive it immediately.

Upon establishment of these two communication channels, we have now completed our target too. As shown in Fig.11, after training, users just need to sit before the monitor, put on the electrode cap and look at the game scene to play the game. When the user wants the user's car to turn left, he performs motor imagery and when to turn right, he relaxes.

IV. Conclusion

In this paper, we intend to design a BCI interface using motor imagery firstly. We employ the CSP and LDA algorithms for feature extraction and classification. And the experimental results have proved the validity of the algorithms. Then we designed a multimedia game via QTCreator. Using unique framework and abundant code resource of QTCreator, we finished the game design more effectively

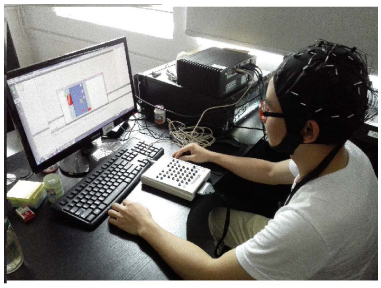

Fig. 11. The scene of game play(photo taken at South China University of Technology)

and clearly. Which will be better for us to complete the game or extend the game to other platforms in the future. At last, we combine the game with biofeedback technology by using TCP and UDP socket. We have considered both the merits and the drawbacks of them and decide to transport EEG data via TCP for TCP's reliability and to transport game commands via UDP which is simpler and faster. The results have demonstrated the stability and rapidity of these communication channels.

References

[1] N. E. Miller, "Biofeedback and visceral learning.," *Annual Review of Psychology*, vol. 29, no. 1, pp. 373–404, 1978.
[2] E. T. Horta, I. C. Lopes, J. J. P. C. Rodrigues, and M. L. Proenca, "A mobile health application for falls detection and biofeedback monitoring," in *e-Health Networking, Applications & Services (Healthcom), 2013 IEEE 15th International Conference on*, pp. 36–40, 2013.
[3] E. Broek and J. Westerink, "Biofeedback systems for stress reduction: Towards a bright future for a revitalized field," *SciTePress-Science and Technology Publications*, 2012.
[4] B. C. J. L. K. Kim, "Attention enhancement system using virtual reality and eeg biofeedback," *Virtual Reality Annual International Symposium*, 2002.
[5] S. G. Mason, A. Bashashati, M. Fatourechi, K. F. Navarro, and G. E. Birch, "A comprehensive survey of brain interface technology designs.," *Annals of Biomedical Engineering*, vol. 35, no. 2, pp. 137–169, 2007.
[6] M. A. Lebedev and M. A. L. Nicolelis, "Brain-machine interfaces: past, present and future.," *Trends in Neurosciences*, vol. 29, no. 9, p. 536C546, 2006.
[7] V. Cmiel, O. Janousek, and J. Kolarova, "Eeg biofeedback," in *International Symposium on Applied Sciences in Biomedical & Communication Technologies*, 2011.
[8] J. Duan, Z. Li, C. Yang, and P. Xu, "Shared control of a brain-actuated intelligent wheelchair," in *Intelligent Control and Automation (WCICA), 2014 11th World Congress on*, pp. 341–346, 2014.
[9] C. C. Postelnicu, D. Talaba, and M. I. Toma, *Controlling a Robotic Arm by Brainwaves and Eye Movement*. Springer Berlin Heidelberg, 2011.
[10] J. Pitsch and J. U. Quevedo-Torrero, "Cases of study on the social effects of online gaming," in *International Conference on Information Technology: New Generations*, pp. 1241–1242, 2010.
[11] G. Pfurtscheller and C. Neuper, "Motor imagery and direct brain-computer communication," *Proceedings of the IEEE*, vol. 89, no. 7, pp. 1123–1134, 2001.
[12] C. S. Tsui, J. Q. Gan, and H. Hu, "A self-paced motor imagery based brain-computer interface for robotic wheelchair control.," *Clinical Eeg & Neuroscience Official Journal of the Eeg & Clinical Neuroscience Society*, vol. 42, no. 4, pp. 225–9, 2011.
[13] P. Doynov, J. Sherwood, and R. Derakhshani, "Classification of imagined motor tasks for bci," in *Region 5 Conference, 2008 IEEE*, pp. 1 – 6, 2008.
[14] http://www.4399.com/.