The European Proceedings of
Social & Behavioural Sciences
EpSBS

Future Academy

ISSN: 2357-1330

https://dx.doi.org/10.15405/epsbs.2018.11.02.55

ICPE 2018
International Conference on Psychology and Education

GAMIFICATION DEVICE WOWCUBE: DESIGN AND PROGRAM ENVIRONMENT

Ilya V. Osipov (a) *, Evgeny Nikulchev (b), Ilya Egoryshkin (c), Semyon Orlov (d)
*Corresponding author

(a) Cubios Inc., P.O. Box 22, Novato, CA, 94948, USA, ilyavosipov@gmail.com
(b) MIREA – Russian Technological University, Moscow, Russia & Russian Academy of Science, Moscow, Russia
nikulchev@mail.ru
(c) MIREA – Russian Technological University, Moscow, Russia, egoryshkin-ilya@rambler.ru
(d) Cubios Inc., P.O. Box 22, Novato, CA, 94948, USA, into@shtorkin.ru

Abstract

We describe the software development framework for the newly developed transreality gaming device called WOWCube, which is a mechatronic game console consisting of eight identical section equipped with individual computation and communication tools that integrate the sections in a common logical device. The main means of controlling the console, by analogy with the well-known Rubik's Cube puzzle is varying the mutual position of the sections. WOWCube has 24 displays distributed over the faces of all sections. Its winning combinations and the gaming strategy are determined in software and depend on the game software loaded to the console. Thus, the developed WOWCube device employs the cutting-edge transreality technologies and has a significant educational gaming potential. Combination of the properties of a real puzzle, whose faces are in motion, with new information technologies, open up new possibilities related to education and development of the cognitive functions. The paper describes the principles of the development of gaming and educational software for the WOWCube. The WOWCube development framework consists of three modules, specifically, programming of the operating logic of the gaming educational program in the PAWN language, which is a byte code compiler and interpreter; the 3D emulator of the WOWCube physical device, which is written in the processing graphical language; and the Tasks required to control the assembly of, e.g., the processes of compilation, testing, and debugging, and stored in a json file.

Keywords: Gamification, transreality gaming, puzzle, software development.

https://dx.doi.org/10.15405/epsbs.2018.11.02.55
Corresponding Author: Ilya V. Osipov
Selection and peer-review under responsibility of the Organizing Committee of the conference
eISSN: 2357-1330

1. Introduction

Pervasive games are computer games, which have a function or several functions expanding conventional gaming boundaries in time, space, or social life (Kasapakis & Gavalas, 2015). Such games are based on such novel information technologies as virtual reality and augmented reality and employ a wide range of sensors, e.g., motion-sensing devices and viewing angle sensors (Lv et al, 2015), as well as various special visualization tools and network technologies (Montola et al, 2009). Pervasive gaming applications are in common use in multiple domains, from everyday life (Valente et al., 2017) to education (Mayo, 2007; Morin et al., 2018) and rehabilitation (Rego et al., 2010; Afyouni et al., 2017).

Proliferation of mobile devices supported by communication systems and data transmission channels has led to ubiquitous computerization, where multimedia information is incorporated in our physical environment (Capra et al,, 2005). This underpins further progress, including widespread acceptance of educational gaming applications for mobile devices (Jordine et al., 2015; Osipov & Nikulchev, 2018).

Many psychological studies deal with the affect of gaming on the development of human working memory, human mobility, and cognitive functions (Huang et al, 2017). In particular, researchers found the connection between gaming and depression, as well as a stable dependence of behavior patterns on Internet gaming disorder (IGD). The research results demonstrate, however, that such IGD dependences occur less frequently, if mobile devices are used for gaming (Paik et al., 2017).

Our work aims at creating a special device for puzzle solving, which is based on potentialities of new technologies, such as transreality, network techniques, and microprocessors (Osipov, 2017). WOWCube is designed to simplify the learning process and augment leisure interest, while staying user-friendly and intuitive (see Fig. 1). WOWCube gaming allows cultivating response actions, reasoning, rational thinking, and fine motor skills, since this gaming device employs the Tangible User Interface approach.

Figure 01. General view WOWCube

https://dx.doi.org/10.15405/epsbs.2018.11.02.55
Corresponding Author: Ilya V. Osipov
Selection and peer-review under responsibility of the Organizing Committee of the conference
eISSN: 2357-1330

2. Problem Statement

2.1.WOWCube Design

WOWCube is a mechatronic game console, which consists of eight identical sections equipped with individual means of computation and communication that integrate the sections in a common logic device. The main method of controlling the console is varying the mutual position of the sections in a manner similar to well-known Rubik's Cube. As opposed to Rubik's Cube, where the color of each cube face is static and predetermined, and the search for one and only one combination is the aim of the game, the WOWCube has 24 displays on the faces of the elements, whereas the winning combinations and the gaming strategy are specified in software and depend on the game uploaded to the console.

The possibility to vary the gaming process in the WOWCube software is ensured by a set of microcontrollers, which render on-screen graphics, determine the game mechanics, and support the interaction of adjacent sections via a couple of serial ports (universal synchronous asynchronous receiver/transmitters, USART). These ports are full-duplex and support data exchange between not only neighboring sections, but any other sections as well. In this case, some sections act as relay agents.

Each section shaped as a cube has three internal and three external sides (faces), where "internal" means conventionally the sides, which face the sides of the neighboring sections, and "external" means the sides visible to the user. Each external face of each section is equipped with a 240x240 TFT display, which has a maximum color depth of 18 bits and is connected to the microcontroller directly via a parallel bus (see Fig. 2).

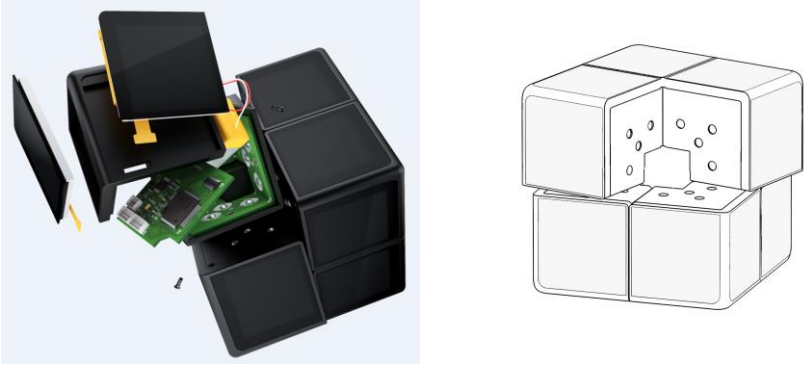

Figure 02. WOWCube Design

After any change in the configuration, i.e., rotation of the console sections relative to each other, newly contacted sections are identified by each section in software. This allows controlling the gaming process, for example, a game item can move from the display of one section to the display of another section, which is adjacent to the former one, within the common plane formed by the displays of four adjacent sections. From the player's viewpoint, this looks like a continuous motion of the item along a common display formed by the face of the console in the stable position, excluding the technological gap between the TFT displays that comprise it.

The internal sides are equipped with groups of magnetic contacts ensuring the electric contact between the mating contacts of neighboring sections (Fig. 2). Each group consists of four contacts, namely: TX for USART data transmission, RX for USART data reception, Charge being the common

https://dx.doi.org/10.15405/epsbs.2018.11.02.55
Corresponding Author: Ilya V. Osipov
Selection and peer-review under responsibility of the Organizing Committee of the conference
eISSN: 2357-1330

charging bus for all sections (which also has the function of rebooting the sections it joins at a specific moment in the case of “grounding”) with a charging voltage of +5 V relative to the common wire, and Ground being the “ground” or “common wire”. The latter contact is the reference point for formation of logical levels at the RX and TX contacts, as well as the charging voltage and the low level of rebooting at the Charge contact.

The contacts are located in such a way as to ensure that the Ground and Charge contacts of neighboring sections always connect to the peer contacts, respectively, whereas the connection of the USART contacts, contrariwise, is always “crossover” in full accordance with the data transmission protocol, i.e., an RX contact is always connected to the neighboring TX contact, and a TX contact, to the neighboring RX contact. This is achieved by installing the Ground and Charge contacts in the same radius from the device center, and the RX and TX contacts, in the same circle, symmetrically relative to the radius, at which the rest of the contacts are situated.

The patented design of the contacts (patent WO2018075714, see Fig. 3) makes it possible to maintain electric connections between arbitrary contacts, which are elements of the console structure, independently of the initial orientation of the magnetic poles of ball-shaped neodymium magnets, while making the console position stable when the neighboring displays on adjacent faces lie in the same plane.

Magnetic interaction between neodymium magnets, which is stronger than the interaction of the magnets with their collars ensure reorientation of the magnets after each variation in the console configuration, thus maintaining reliable electric contacts in the USART data links.

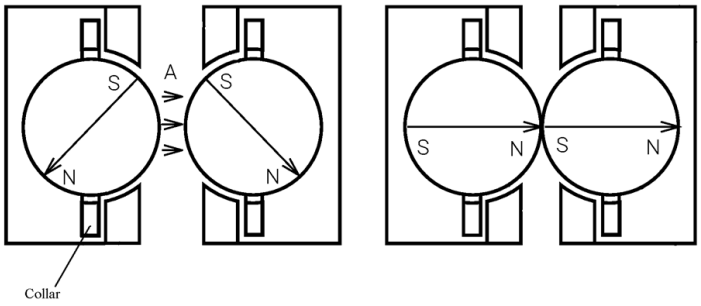

Figure 03. Design of contact groups

2.2.Development framework

The gaming logic module and the emulator exchange data using the user datagram protocol (UDP). UDP is not a random choice, since it is included in all operating systems (OS) making it possible to port the development framework to other platforms making as few changes as possible.

The communication between the unit cubes is based on the Tag-Length-Value (TLV) method, which is used to record small data in computer files and telecommunication protocols. The method specifies a structure of three fields, namely, the tag, the data length, and the data. The fields of the tag and data length have a size of 1 or 2 bytes, and the size of the third field is determined by the value of the second field.

The WOWCube game logic is programmed in the PAWN language, with an individual script for each game.

https://dx.doi.org/10.15405/epsbs.2018.11.02.55
Corresponding Author: Ilya V. Osipov
Selection and peer-review under responsibility of the Organizing Committee of the conference
eISSN: 2357-1330

3. Research Questions

In this work, we consider the principles underlying the programming process for the designed device.

The WOWCube development framework consists of three modules:

- the software logic written in the PAWN language (bytecode compiler and interpreter);
- the 3D emulator of the physical device (WOWCube) written in the Processing graphic language; and
- the tasks (Tasks) required to control integration of, e.g., the processes of compilation, testing, and debugging are stored in a json file.

4. Purpose of the Study

The purpose of the study is to develop the programming principles for the designed structure of the gaming device.

5. Research Methods

The microcontroller of each individual cube is an inexpensive microcontroller with a limited functionality and a memory of only 192 kB, which should accommodate the binary code and all resources of the code. The programme code is written in PAWN and then compiled. The code for the microcontrollers is written in the C language, which the PAWN language itself is written in.

To write the program logic, the PAWN script language is chosen, since it is simple and easy to comprehend. Additionally, since it is written in the C programming language, it is ported easily to different platforms. PAWN allows running large scripts in a small memory space, which is a significant advantage for working with the limited memory of the microcontroller.

6. Findings

6.1.Processing module

The code for emulation of operation of the physical device is common for all games. it contains the following classes:

- CDisplay is the display object responsible for rendering of a display image;
- CFace describes one of the three sides of the tube and contains CDisplay;
- CCube is the object corresponding to one of the eight cubes; totally, there are eight such objects, each containing three CFace objects;
- CCubeSet is the object of the entire WOWCube; it creates 8 CCube objects and is responsible for animation of the cube rotation in the emulator;
- CPawnLogic describes the interface of data exchange with PAWN via UDP.

The main task fulfilled by the model is to emulate the operation of the physical model and serve as the API. The communication with PAWN is supported by the command pattern. The code of the object command, which is received and transmitted by the module, is as follows:

```
class CPawnCmd {
  public int cubeN;
```

https://dx.doi.org/10.15405/epsbs.2018.11.02.55
Corresponding Author: Ilya V. Osipov
Selection and peer-review under responsibility of the Organizing Committee of the conference
eISSN: 2357-1330

```
 public byte[] pkt;
  CPawnCmd(int _cubeN, byte[] _pkt) {
  this.cubeN = _cubeN;
  this.pkt = new byte[_pkt.length];
  arrayCopy(_pkt,this.pkt);
 }
}
```

The following commands are sent to the Processing interface in the current version of the development framework (0.2):

- CMD_GUI_DEBUG is the command required for debugging;
- CMD_FILL is the command for monochrome filling of the display (the above-mentioned CDisplay class; it is required for display clearing or highlighting for emphasis);
- CMD_BITMAP is the most frequent command; it renders a display image corresponding to the number received with the command. The image is taken from the common resource set (PImage res[]).

The following commands are sent to the binary interface (cubios_abi.pwn):

- CMD_PAWN_DEBUG is the command required for debugging in the PAWN module already;
- CMD_TICK is the command for imitation of one cycle of microcontroller operation; it is sent every 100 milliseconds;
- CMD_ATTACH is the command sent at an instant when the cube halves stop rotating;
- CMD_DETACH is the command sent at an instant when the cube halves start rotating.

It should be noted that the commands are sent within a cycle via each of the eight ports. As it has been already mentioned, modules exchange data using the UDP protocol, and an individual port corresponds to each of the eight cubes. A separate port corresponds to the processing module.

The diagram of the module classes is shown in Fig. 4.

The same module describes functions/events:

void mouseDragged() is for rotation of the 3D cube projection using a mouse;

void keyPressed() is for rotation of the 3D cube projection using a keyboard; and

void receive(byte[] pkt, String ip, int port) is for reception of cube data batches using the UDP protocol.

https://dx.doi.org/10.15405/epsbs.2018.11.02.55
Corresponding Author: Ilya V. Osipov
Selection and peer-review under responsibility of the Organizing Committee of the conference
eISSN: 2357-1330

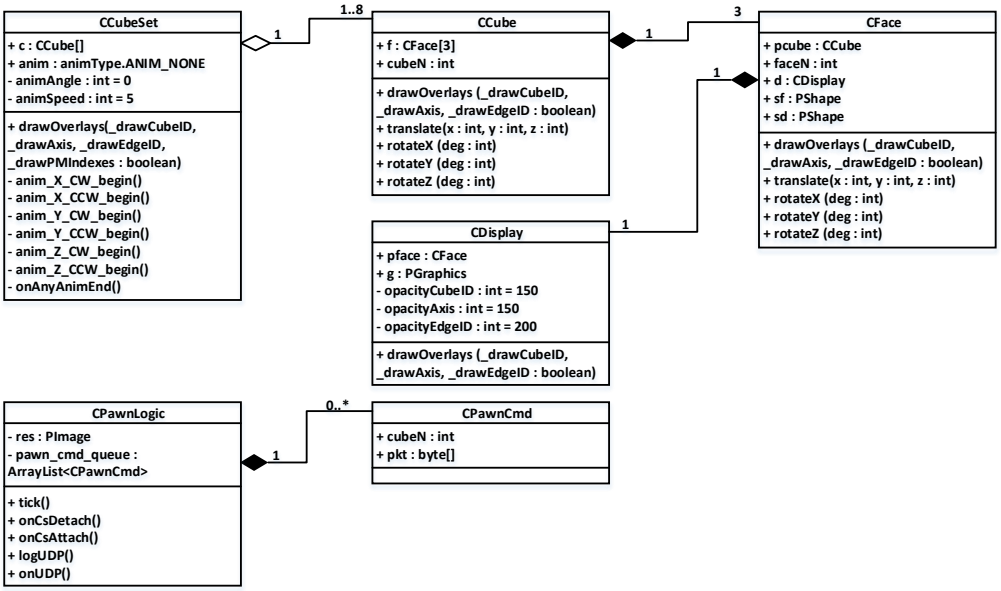

Figure 04. Diagram of the classes of the Processing module

6.2.PAWN module

The module where the game logic is written contains:

- cubios_abi.pwn, the binary interface responsible for the data exchange with the Processing via UDP;
- %gamename%.pwn describing the game logics; and
- game.amx, the complied binarty file (byte code) containing the logics of all games.

The transmitted and received commands were described above in Section 2.1, and will not be repeated here. Note only that the GUI commands are received by the following functions:

```
@receivepacket(const packet[], size, const source[]) {
        run(packet, size, source);
}
```

The **run** function launches processing of the received command. The commands are sent in the game logic script by calling the functions abi_CMD_BITMAP and abi_CMD_FILL.

The cubios_abi binary interface includes the following functions:

- abi_CMD_BITMAP, abi_CMD_FILL are the above-mentioned commands for data transmission to the Processing;
- abi_InitialFacePositionAtProjection is the initial location of the face side of the cube on the projection;
- abi_FacePositionAtProjection is the location of the current position of the face side of the cube on the projection;

https://dx.doi.org/10.15405/epsbs.2018.11.02.55
Corresponding Author: Ilya V. Osipov
Selection and peer-review under responsibility of the Organizing Committee of the conference
eISSN: 2357-1330

- abi_DeserializePositonsMatrix is deserialization of the batch of data received from the Processing;
- abi_LogSndPkt, abi_LogRcvPkt is the function of logging of the data sent and received, respectively;
- abi_LogPositionsMatrix is the matrix of the current position of the cubes.

The %gamename%.pwn contains the game logic. The main function of all eight cubes is identical:

```
main() {
  new opt{100};
  argindex(0, opt);
  abi_cubeN = strval(opt);
  listenport(PAWN_PORT_BASE+abi_cubeN);
  }
```

Receiving the cube number at its output, the cube starts sounding this port in order to exchange commands with the processing interface. The rest of the methods in the given file differ depending on the game.

6.3.json module

This module contains a file with tasks for automatic assembling, compiling, emulating, etc. One can have a look at the tasks at https://code.visualstudio.com/docs/editor/tasks.

To launch a game on the WOWCube, it is necessary to first compile it. The task for game compilation looks as follows:

```
"label": "1) WOWCube compile game logic (Pawn language)",
"type": "shell",
"command": "../../Pawn/bin/pawncc %gameName%.pwn -ogame",
"options": {
   "cwd": "./pawn"
},
"problemMatcher": []
```

The framework will compile the game as a game.amx file. Then, the game logic should be started. Since each cube is an individual element, they all should be launched separately. Sequential starting of all eight cubes is executed by the following task which launches tasks of individual cubes:

```
"label": "2) WOWCube run #0-7 Cube game logic (Pawn language)",
"dependsOn": [
  "WOWCube run #0",
  "WOWCube run #1",
  "WOWCube run #2",
  "WOWCube run #3",
  "WOWCube run #4",
```

https://dx.doi.org/10.15405/epsbs.2018.11.02.55
Corresponding Author: Ilya V. Osipov
Selection and peer-review under responsibility of the Organizing Committee of the conference
eISSN: 2357-1330

"WOWCube run #5",
"WOWCube run #6",
"WOWCube run #7",
],
"problemMatcher": []

The task of an individual cube has the following code:

"label": "WOWCube run # %cubeNumber%",
"type": "shell",
"command": "../../Pawn/bin/pawnrun game.amx %cubeNumber%",
"options": {
"cwd": "./pawn"
},
"problemMatcher": []

The last but not least element to launch is GUI:

"label": "3) WOWCube run GUI emulator (Processing language)",
"type": "shell",
"command": "../Processing/processing-java.exe --sketch=../WOWCube --run",
"problemMatcher": []

7. Conclusion

The developed WOWCube device employs cutting-edge transreality technologies and has a significant educational gaming potential. The principles that underlie the development of gaming and educational programs for the WOWCube have been formed.

The language chosen to write the program logic is the script PAWN language, which is simple and easy to comprehend. Additionally, since PAWN is written in the C programming language, it is easily ported to various platforms.

PAWN allows executing large scripts in small memory volumes, which is a great advantage in the limited memory conditions.

Acknowledgments

The authors express their gratitude to the following people, without whom the project would not have taken place: Vadim Filipov, Vadim Savvateev, Evgeny Zhukov, Savva Osipov, Mikhail Soloviev, Valentin Zubkov, Alex Pristenskiy, Andrey Antar, Denis Mitrofanov, Sergey Syrov and Grigory Bubnov.

References

Afyouni, I., Rehman, F. U., Qamar, A. M., Ghani, S., Hussain, S. O., Sadiq, B., ... & Basalamah, S. (2017). A therapy-driven gamification framework for hand rehabilitation. *User Modeling and User-Adapted Interaction, 27*(2), 215-265.

Capra, M., Radenkovic, M., Benford, S., Oppermann, L., Drozd, A., & Flintham, M. (2005, November). The multimedia challenges raised by pervasive games. In *Proceedings of the 13th annual ACM international conference on Multimedia* (pp. 89-95). ACM.

Huang, V., Young, M., & Fiocco, A. J. (2017). The Association Between Video Game Play and Cognitive Function: Does Gaming Platform Matter? *Cyberpsychology, Behavior, and Social Networking, 20*(11), 689-694.

Jordine, T., Liang, Y., & Ihler, E. (2015). A Mobile Device Based Serious Gaming Approach for Teaching and Learning Java Programming. *International Journal of Interactive Mobile Technologies, 9*(1), 53-59

Kasapakis, V., & Gavalas, D. (2015). Pervasive gaming: Status, trends and design principles. *Journal of Network and Computer Applications, 55*, 213-236.

Lv, Z., Halawani, A., Feng, S., Ur Réhman, S., & Li, H. (2015). Touch-less interactive augmented reality game on vision-based wearable device. *Personal and Ubiquitous Computing, 19*(3-4), 551-567.

Mayo, M. J. (2007). Games for science and engineering education. *Communications of the ACM, 50*(7), 30-35.

Montola, M., Stenros, J., & Waern, A. (2009). *Pervasive games: theory and design*. CRC Press.

Morin, S., Robert, J. M., & Gabora, L. (2018). How to train future engineers to be more creative? An educative experience. *Thinking Skills and Creativity, 28*, 150-166.

Osipov, I. V. (2017). Cubios Transreality Puzzle as a Mixed Reality Object. *International Journal of Virtual and Augmented Reality (IJVAR), 1*(2), 1-17.

Osipov, I. V., & Nikulchev, E. (2018). Review puzzles and construction sets falling under the category of augmented reality games. *ITM Web of Conferences, 18*, 02003.

Paik, S. H., Cho, H., Chun, J. W., Jeong, J. E., & Kim, D. J. (2017). Gaming Device Usage Patterns Predict Internet Gaming Disorder: Comparison across Different Gaming Device Usage Patterns. *International journal of environmental research and public health, 14*(12), 1512.

Rego, P., Moreira, P. M., & Reis, L. P. (2010). Serious games for rehabilitation: A survey and a classification towards a taxonomy. In *2010 5th Iberian Conference on* I*nformation Systems and Technologies (CISTI)*, (pp. 1-6). IEEE.

Valente, L., Feijó, B., & do Prado Leite, J. C. S. (2017). Mapping quality requirements for pervasive mobile games. *Requirements Engineering, 22*(1), 137-165.