CONTENTS

INTRODUCTION

An integral part of the program is to conduct conference in the regions of the country on “Priorities of further implementation and development of modern ICT in Uzbekistan, determined by Resolution of the President of the Republic of Uzbekistan № 1730 of March 21, 2012”, as well as the cycle of practical seminars, where managers and specialists of regional organs of government and economic management, state authorities at places, subjects of business and civil society institutions have the opportunity to get acquainted with the issues of practical experience in implementing IT projects, improving the effectiveness of activities of various stakeholders through the use of modern ICT, development of ICT infrastructure to serving citizens.

As part of the conference, participants were fully informed about the importance and role of the adopted legislative act, its importance for the implementation of comprehensive reforms of the information sphere, forming the National Information System based on phased integration of information systems, government agencies, as well as businesses and individuals. Issues of creating information systems of activities’ automation, expansion of the range and quality of online government services, providing access to relevant information resources, including those in rural areas were discussed at the conference. Speakers provided information on realized in the context of improving of the Resolution of the President of measures aimed at improving the regulatory system in the sphere of ICT, ensuring security of national information system, protect its information resources.

The revolution in ICTs has profound implications for economic and social development. It has pervaded every aspect of human life whether it is health, education, economics, governance, entertainment etc. Information and communication technologies are also crucial drivers for tourism providers to conduct business transactions, distribute their products and services, network with

trading partners and provide information to consumers worldwide. An increasing number of consumers are using the Internet to obtain information, plan and buy their travel. This session will provide an introduction on the role of tourism for development and will highlight the role of technology innovation in the tourism sector. Tourism is currently one of the fastest gowning industries across the world. It is primarily a service industry as it does not produce goods but renders services to various classes of people.

Development of Information and Communication Technologies (ICT) has transformed the contemporary business environment. It has led to new information economy which is digital in nature. ICT is a broad terminology referring to multiple communication technologies which range from simple and complex namely Cell Phone applications (SMS), Digital Cameras, Internet, Wireless (WiFi and WiMAN), VOIP,GPS, GIS, Convergence (data, voice, media), Digital radio, These technologies are creating a new global market place, which is more competitive. With e-commerce and e business the market has more opportunities and possibilities than ever before. The ability to reach a global audience, obtain instant market information and conduct electronic business transactions has increased economic efficiency and has opened markets for goods and services from the developing world.

Creative knowledge work and solid IT systems are necessarily at odds—the former requires high flexibility, the latter requires high stability. Reconciling their natures is difficult, so many organizations leave them separate. This can leave important knowledge work disconnected from mainstream systems and relegate mainstream systems to lower-value analysis work. Organizations seeking to maximize every analytical opportunity must bridge this gap by maximizing the efficiency of each area, then fully integrating them. This paper describes how to do it. Flexibility is a dominant characteristic of creative knowledge work. Actuaries may operate from a strong mathematical foundation, but new and emerging opportunities start as little more than vague ideas. Turning ideas into actionable

information requires looking at data—often lots of it— manipulating it, observing the outcome, getting more data, manipulating it more, observing it again, and so on through many iterations, often with several serendipitous epiphanies along the way. Stability is a dominant characteristic of IT systems. During development, systems must be built to a fixed timeline and budget, multiple IT skill sets must converge on a single outcome, developers must learn business logic, and changes to code once it is written can have large propagation effects. During operations, systems must meet service level agreements (SLAs), be supported by IT staff of lesser skill, exist in the corporate systems environment for many years, rarely crash, and have their changes coordinated with multiple business units. Little about these two worlds works well together.

Purpose and functions of graduate work: The purpose of the work is to create mini game collections for iOS mobile operating system. Mini game collections mainly destine to kids of school age. We know, children are interesting to games. With this collections they can play game during playing game can learn math. Because of this the main function of the MathLogicGames is teaching objects(math) to children with the way which they like.

Coming thereof possible select the following primary tasks of the development and introducing the collections game:

- Analyzing a game application and developing;
- Choosing the corresponding software;
- Making the functional structure of the game;
-Designing and steps of doing task;
-Results and effectiveness of game.

1. COMMON CONCEPT TO DEVELOP GAMES

1.1. Analyze common software for making game

Computer software, or simply software, also known as computer programs, is the non-tangible component of computers. Computer software contrasts with computer hardware, which is the physical component of computers. Computer hardware and software require each other and neither can be realistically used without the other.

Computer software includes all computer programs regardless of their architecture; for example, executable files, libraries and scripts are computer software. Yet, it shares their mutual properties: software consists of clearly defined instructions that upon execution, instructs hardware to perform the tasks for which it is designed. Software is stored in computer memory and cannot be touched, just as a 3D model shown in an illustration cannot be touched.

At the lowest level, executable code consists of machine language instructions specific to an individual processor– typically a central processing unit. A machine language consists of groups of binary values signifying processor instructions that change the state of the computer from its preceding state. For example, an instruction may change the value stored in a particular storage location inside the computer – an effect that is not directly observable to the user. An instruction may also (indirectly) cause something to appear on a display of the computer system – a state change which should be visible to the user. The processor carries out the instructions in the order they are provided, unless it is instructed to "jump" to a different instruction, or interrupted.

Software is usually written in high-level programming languages that are easier and more efficient for humans to use (closer to natural language) than machine language. High-level languages are compiled or interpreted into machine language object code. Software may also be written in a low-level assembly language, essentially, a vaguely mnemonic representation of a machine language

using a natural language alphabet. Assembly language is converted into object code via an assembler.

Capabilities of modern PCs are so great that an increasing number of people find a use for it in their work, school, home. The most important quality of a modern computer it is "friendly " to the user. Human-computer communication has become simple, clear and understandable. Computer itself tells a user what to do in a given situation, helps to get out of difficult situations. This is possible thanks to the software of the computer.

Again use the analogy between computer and human. Newborn people do not know and do not know how. Knowledge and skills he acquires in the course of development, learning, accumulating information in your memory. Computer, which gathered at the factory of chips, wires, circuit boards and other things, like a newborn person. We can say that booting the computer memory software similar to the learning process of the child. Created software programmers.

The entire set of programs stored in non-volatile memory devices of all computer makes its software.

The computer software is constantly updated, developing and improving. The cost of the programs installed on a modern PC often exceeds the cost of its technical devices. Development of modern software requires very high skills of programmers.

In computer software is a necessary part, without which it simply can not do anything. It is called by the system software. Buyer purchases a computer equipped with the system software, which is no less important for the computer than the memory or processor. In addition the system software of the computer software includes other applications and systems programming.

The computer software is divided into:

- System software ;
- Application software ;
- Programming system.

About system software and programming systems discussed later. And now look at the application software. Program, through which the user can solve their information problems without resorting to programming, called application programs. As a rule, all users prefer to have a suite of applications that need to almost everyone. They are called general-purpose programs. Among them are :

- Text and graphics editors, with which you can cook a variety of texts, to create drawings, building drawings; simply put, write, draw, paint ;

- A database management system (DBMS), allows you to turn your computer into a directory on any topic ;

- Tabular processors, allows to organize a very common practice tabular calculations;

- Communication (networking) programs designed to share information with other computers, combined with the data in a computer network.

Very popular application software are computer games. Most of the users it is with them begins its communication with a computer.In addition, a large number of special-purpose applications for professional activities. They are often called application packages. This, for example, accounting software, generating payroll and other payments that are made in the accounting department ; computer aided design, which help designers to develop projects of various technical devices ; Packages that solve complex mathematical problems without programming ; training programs for various school subjects and more.Set of programs designed to solve problems on your PC, called software. Composition of software called PC software configuration. Software can be divided into three categories:system software (programs shared use) that perform various support functions, such as making copies of the information used, the issue of background information about your computer, check the computer's performance, etc.application software, ensures implementation of the necessary work on your PC: Edit text documents, create drawings or pictures, bulk processing, etc. tool software (system software) that provides the development of new programs for the computer programming

language.

This public programs are not associated with a particular application PC and perform traditional functions : planning and task management, input-output control, etc.In other words, system programs perform various support functions, such as making copies of the information used, the issue of background information about your computer, check the computer's performance, etc.

By the system software include:

- operating system (this program is loaded into RAM at startup);
- programs - the shell (provide more convenient and intuitive way to communicate with the computer than using the command line DOS, for example, Norton Commander) operating envelope - Interface systems that are used to create graphical interfaces multiprogramming IT. Drivers (software designed for managing ports peripherals, typically loaded into memory when the computer starts);
- utilities (auxiliary or utilities, which represent the user a number of additional services);

By utilities are:

- file managers or file managers means of dynamic data compression (can increase the amount of information on the disk due to its dynamic compression).
- Viewer and playback diagnostic tools ; controls allow you to verify the configuration and test the computer devices, especially hard drives means of communication (communication program) designed to exchange information between computers means computer security (backup, anti-virus software).It should be noted that some utilities included in the operating system and the other part is functioning autonomously. Most of the total (system) software is included with the OS. Part of the overall software is part of the computer program (part of the OS and control tests stored in ROM or EPROM installed on the system board) Part of the overall software refers to a standalone program and shipped separately.

Application programs can be used alone or as part of software packages or

packages. Application software - programs directly ensuring the implementation of necessary work on your PC: Edit text documents, create drawings or pictures, create spreadsheets, etc. Application packages - is a system of programs that fall under the scope of application to the problem - oriented, general-purpose packages and integrated packages. Modern integrated packages contain up to five functional components : a test and a spreadsheet, database, graphics editor, telecommunications facilities.

For application software, for example, include:

- Set of office applications MS OFFICE;
- Accounting software;
- Financial analysis systems;
- Integrated packages of office;
- CAD - system (Computer Aided Design);
- HTML editors or Web – Editors;
- Browsers - Viewer Web – pages;
- Graphic Editors;
- expert Systems;
- And so on.

Tool software or programming system - a system to automate the development of new programs in the programming language.

In the most general case, to create a program on the selected programming language (the language of system programming) must have the following components:

1. Text editor to create a file containing source program.

2. Compiler or interpreter. Source code using the compiler translates into an intermediate object code. Source code of a large program consists of several modules (source files). Each module is compiled into a separate file with the object code, which then must be combined into one.

3. Linker or collector that performs binding object modules and outputs a

good application - executable code.

Executable code - it is a complete program that can be run on any computer where the operating system for which the program was created. Typically, the resulting file has an. EXE or. COM.

4. Lately become widespread visual programming techniques (using scripting languages), aimed at creating Windows-based applications. This process is automated in the rapid design environments. It uses ready- visual components are configured using special editors.

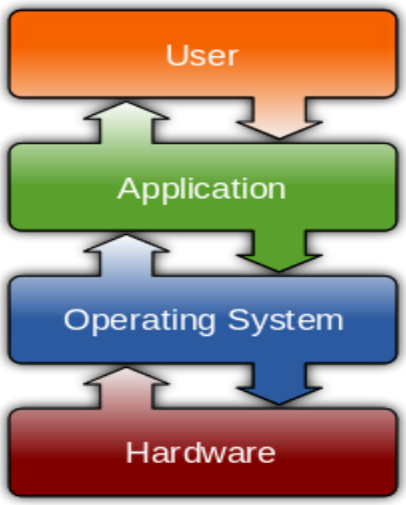
Most Popular Editors (programming system programs using visual aids) visual design :

Borland Delphi - is designed for virtually any application programming tasks

Borland C + + Builder - an excellent tool for developing DOS and Windows applications

Microsoft Visual Basic - is a popular tool for creating Windows-based programs

Microsoft Visual C + + - this tool allows you to develop any applications running in the environment of an OS like Microsoft Windows



1.1-Figure. Layer the operating system software and application software on a typical desktop computer.

Users often see things differently than programmers. People who use modern general purpose computers (as opposed toembedded systems, analog computers and supercomputers) usually see three layers of software performing a variety of tasks: platform, application, and user software.

- Platform software: Platform includes the firmware, device drivers, an operating system, and typically a graphical user interface which, in total, allow a user to interact with the computer and its peripherals (associated equipment). Platform software often comes bundled with the computer. On a PC one will usually have the ability to change the platform software.
- Application software: Application software or Applications are what most people think of when they think of software. Typical examples include office suites and video games. Application software is often purchased separately from computer hardware. Sometimes applications are bundled with the computer, but that does not change the fact that they run as independent applications. Applications are usually independent programs from the operating system, though they are often tailored for specific platforms. Most users think of compilers, databases, and other "system software" as applications.
- User-written software: End-user development tailors systems to meet users' specific needs. User software include spreadsheet templates and word processor templates. Even email filters are a kind of user software. Users create this software themselves and often overlook how important it is. Depending on how competently the user-written software has been integrated into default application packages, many users may not be aware of the distinction between the original packages, and what has been added by co-workers.

Computer software has to be "loaded" into the computer's storage (such as the hard drive or memory). Once the software has loaded, the computer is able to execute the software. This involves passing instructions from the application software, through the system software, to the hardware which ultimately receives the instruction as machine code. Each instruction causes the computer to carry out

an operation – moving data, carrying out a computation, or altering the control flow of instructions. Data movement is typically from one place in memory to another. Sometimes it involves moving data between memory and registers which enable high-speed data access in the CPU. Moving data, especially large amounts of it, can be costly. So, this is sometimes avoided by using "pointers" to data instead. Computations include simple operations such as incrementing the value of a variable data element. More complex computations may involve many operations and data elements together.

1.2. The game development process and steps

The game development process usually begins with a game concept. Among hobbies and small teams, the concept may be formed in the head or written on a napkin, but in the more serious studios it is written down carefully. This document guides the prototyping phase, low-fidelity paper versions of the game, or very simple digital versions of the game are developed, tested, and explored. If the concept looks promising, more documents are created, such as the art bible the production plan, and eventually leads to the creation of the game design document as well as the technical design document.

At this point, production is begun, a stage that can last from several weeks to a few years. An alpha version of the game is finally produced at which point the game is playable from beginning to end. This is tested and de-bugged until the beta version of the game is developed. The beta version is similar to the alpha version, but should include all final art assets, and be stable enough that a selection of customers can play the game and report problems. The final stage is sometimes called the gold stage, where the game has gone through complete testing and has passed a final review.

This document plans the stages of the development of the game in the form of a schedule, taking into account all of the necessary game design, prototyping,

creation of documents and other planning, and goes through actual production of alpha, beta, and gold versions of the game. Here the man-hours to create the game are calculated, as well as all other costs including costs for the software and any other expenses with operating the company.

Step 1: Decide what kind of game you want to make

Do you want to make a 2D game or a 3D game? Adventure game, sports game, or simulation? Should the game be playable on a PC, Mac, Linus, console, iphone, or cellphone? Is it a single-player game or multiplayer? Will you utilize First Person perspective, Third Person perspective, "god"; perspective? See the Game Genres page for a breakdown of the different types of games you could develop.

Step 2: Consider your skills, tools, budget, and resources

Now that you have an idea of what kind of game you want to create, take some time and do a reality check; how well do you and your resources measure up to the challenge you have set for yourself? Are you skilled with creating art assets such as a game environment, pickups, skyboxes, characters? Can you animate? Can you program or script a game engine? Do you know anyone who can help? Do you have the budget to hire help? Do you own or can you afford the necessary hardware and software to build the game? Have you decided on a game engine? Get an idea of the different options for Art Asset creation by clicking the Game Dev Art Assets page. Get an overview of the different Game Engines out there by clicking the Game Dev Game Engines page.

Step 3: Revise Plan based upon Reality Check

With your skills and resources clearly in mind, and having read through the sections on Art Asset creation, game engine selection, programming, and game genres, revise your plan so that it and reality coincide. The other aspect to this step is to simplify your plan so that it is doable in a few weeks or months. It is very common in the world of game development to have young, inexperienced artists and programmers with huge, impressive ideas but no real means of getting the

game accomplished. Start small and simple, finish something, and build from there. Create a basic game design document that establishes some framework for how the game should play, what the rules are, etc.

Step 4: Make a Paper or Low-Tech Prototype

Take your game idea and test it out as if it were a board game. Work through the game-play with your development team or some friends. If it's not interesting as a board game, it probably won't be interesting as a video game. This is not always possible, of course; a game like asteroids is not easy to simulate as a board game, so in that case you might need a digital prototype that is very simple but generates the "feel" of the gameplay. The bottom line here is to test your game idea out before investing too much time and money into it.

Step 5: A quire the Hardware and Software

When you feel sure of your objective and have a good idea of the hardware and software options, it is time to buy some equipment. If you don't have much money, you will at minimum need a computer; almost all of the software you will need to create a game can be had for free, but the tradeoff is that you may not be using software that is the best for the job, and it may be hard to find team members who are knowledgeable about that software. Also, if you are hoping to leverage your game development experience into a job, you may want to use, as much as possible, the same software the game studios are using. Please Note: If you are not developing a commercial game, you may be able to use educational versions of software to develop your game. See our Developing Game Art and Choosing a Game Engine pages for more information.

Step 6: Prototype High-Risk Features

Now that you've developed a realistic plan, are convinced that the gameplay is compelling, have acquired the hardware and software you will need, and have acquired the necessary team, you are ready to see if you can actually develop the features you will need for the game.

In any game development, there are going to be those features that are high-risk; perhaps you need to be able to incorporate a special kind of collision detection, or you want your character to be able to fly around in the game environment. What happens if you spend three months developing your game and then discover in-game flight is impossible in the game engine you have selected, or you find that you don't have the coding talent to pull it off properly? Well then you may have just wasted three months of work. Without in-game flight, your levels will need to be redesigned for some kind of land travel, and many of your game mechanics will have to change as well. With that in mind, it makes sense to create rough prototypes for every high-risk feature in your game before you begin full-blown production. It is important to know where you stand with the high-risk features, and to have the ability to accurately estimate how long each phase of development will take.

Step 7: Establish the Game Art Pipeline

Getting the art into the game can be straightforward or mindboggling, depending on the game engine you decide to use, and the 3D modeling and animation software you are developing it in. It is critical at this stage to truly understand exactly how you will get all of the game art, including terrains, skyboxes, pickups, and especially rigged characters, into the game. Develop a system that works, and document it so that everyone necessary knows how it is done, to avoid confusion and time loss down the road.

Step 8: Develop or Acquire the Game Art

The game art can be a massive part of any game development effort. While we go into the process of developing game art at GameDev: Creating Game Art, there is also the option of acquiring the game art. Sites like Turbosquid regularly sell entire 3D cities, fully rigged 3D models, fully animated models, perhaps everything you need, potentially saving you an enormous amount of time. However, depending on how the game art was rigged and animated, you may still need to make changes in order for that art to come successfully into your game

engine. Before investing in too much third-party game art, test to see how it is going to work out in the real world.

Step 9: Develop the Alpha Version

The next reasonable phase would be to create a working version of the game, where everything of importance works, though there may still be some rough edges and bugs. At this phase the the game art can still be simplistic or borrowed from stock characters, pickups, and landscapes.

This purpose of the Alpha version is to get the game out there in a playable format for testing. This will be the first time the game as a whole has been playable, from start to finish. This is a time not only for debugging, but to correct any last minute issues with design, playability, and interface.

Step 10: Develop the Beta Version

The beta version is the Alpha version, debugged, with finished game art. This version should be good enough for select members of the general public to play. There will still be bugs in this version, but they should not be so numerous that it is a frustration to test the game out.

Final Version

This version of the game is ready for distribution; it has been tested by numerous end-users, on all intended platforms, in as many situations as reasonably possible. Any significant bug fixes beyond this point can be distributed in either bug patches, or incorporated in subsequent releases.

1.3. iOS mobile operating system and its advantages

iOS (previously iPhone OS) is a mobile operating system developed and distributed by Apple Inc. Originally released in 2007 for the iPhone and iPod Touch, it has been extended to support other Apple devices such as the iPad and Apple TV. Unlike Microsoft's Windows CE (Windows Phone) and Google's Android, Apple does not license iOS for installation on non-Apple hardware. As of

2012 Apple's App Store contained more than 700,000 iOS applications, which have collectively been downloaded more than 30 billion times.

The user interface of iOS is based on the concept of direct manipulation, using multi-touch gestures. Interface control elements consist of sliders, switches, and buttons. The response to user input is immediate and provides a fluid interface. Interaction with the OS includes gestures such as swipe, tap, pinch, and reverse pinch, all of which have specific definitions within the context of the iOS operating system and its multi-touch interface. Internal accelerometers are used by some applications to respond to shaking the device (one common result is the undo command) or rotating it in three dimensions (one common result is switching from portrait to landscape mode).

iOS is derived from OS X, with which it shares the Darwin foundation, and is therefore a Unix operating system. iOS is Apple's mobile version of the OS X operating system used on Apple computers.

In iOS, there are four abstraction layers: the Core OS layer, the Core Services layer, the Media layer, and the Cocoa Touch layer. The current version of the operating system (iOS 5.1.1) dedicates 1-1.5 GB of the device's flash memory for the system partition, using roughly 800 MB of that partition (varying by model) for iOS itself.

iOS technology allows you to interact and manipulate your screen in a variety of ways. This makes it much more user friendly because you can alter the screen size by simply placing two fingers in the centre of the screen and dragging them away from each other. This is particularly useful if you have a visual impairment, or if you have just forgotten your glasses, because within an instant, everything can be made much bigger. Double tapping the screen returns the page to its normal size, or if you wish to zoom out, simply reverse the zoom in motion. There are plenty of other touch screen motions, such as sliding your finger to the left or the right to scroll through photos, as well as dragging and dropping items to customize your apps.

If you are an Android or Blackberry fan, you might be wondering what all the fuss is over IOS software. As you probably already know, IOS is Apple's mobile operating system which was originally developed for the iPhone, but has now expanded to support various other Apple devices including the iPad and iPod touch.

As much as people rave about Apple products, their one flaw is not being able to run Flash or Shockwave videos. This means that if you are on your iPhone and want to look at a site which might include video clips and animations, whether it's PartyPoker or Youtube, unless you've downloaded the alternative software, you cannot view the whole page properly. However, apart from this problem, iPhones have many great advantages.

These might all sound like nifty little moves, but there are some disadvantages too. For example, you cannot interact with the screen if you're wearing regular gloves which is certainly a pain in the winter. Sometimes the screen can freeze, before suddenly performing all the controls you were frantically pressing while the screen was frozen, which causes all sorts of problems. We are therefore not expecting the disappearance of regular computers soon, neither the one of not mobile adapted web sites. This is particularly significant when critical or sensitive information is at stake, or the user's money. A mobile device, be it an iSomething, a blackberry or Android device is used much more casually than a regular computer. You might use it on a bus, in a crowd, while queuing at the post office... and that makes impulse actions much more probable. This is an asset for cheap iOS apps or tunes, but could become a danger when a user may move his finances, take part to a party poker game, or do anything else she may later repent not having done with more concentration. This is the reason why responsible operators allow the use of their services only from the relative calm of a computer. Perhaps there will be fewer clients, but despaired ones are no good publicity.

iOS is made up of four abstraction layers: Core OS, Core Services, Media, and Cocoa Touch.

• Core OS, which includes basic low-level features: system support—threads, sockets, IO, DNS, math, memory—general security services—certificates, pri-vate/public keys, encryption—external hardware management, bluetooth, and sound and image processing.

• Core Services, which includes basic application services, including accounts,contacts, networking, data management, location, calendar events, store pur-chasing, SQLite, and XML support.

• Media Layer, which includes support for 2d and 3d graphics, audio, and video.

• Cocoa Touch, which includes APIs for building applications—multitasking, touch input, notifications, interface views, and access to device data.

2. DEVELOP TO MINI GAMES FOR IOS MOBILE OPERATING SYSTEM

2.1. Develop software technology and instrumentation

Many types of mobile computers have been introduced since the 1990s including the personal digital assistant/enterprise digital assistant, smart phone, tablet computer, ultra-mobile PC, and wearable computer. A personal digital assistant (PDA), also known as a palmtop computer, or personal data assistant, is a mobile device that functions as a personal information manager. PDAs are largely considered obsolete with the widespread adoption of smart phones. Commonly current PDAs are able to connect to the Internet. A PDA has all requirements of connecting to internet such as an electronic visual display, enabling it to include a web browser, audio capabilities enabling use as a portable media player. Most PDAs can access the Internet, intranets or extranets via wireless like Wi-Fi or Wireless Wide Area Networks. Most PDAs uses touchscreen technology.

This is a list and comparison of devices pro designed and marketed by Apple Inc. (formerly Apple Computer Inc.) that run a Unix-like operating system named iOS, often colloquially referred to simply as iDevices. The devices include the iPhone multimedia smartphone, the iPod Touch, which is similar to the iPhone but has no cell phone hardware, and the iPad tablet computer. All three devices function as digital audio and portable media players and Internet clients. The Apple TV, which ran iOS from the second generation of hardware onwards, is a set-top box for streaming media from local sources and from certain internet services to a connected television set, and has no screen of its own.

The operating system on iOS devices can be updated through iTunes, or, on iOS 5 or later, using firmware-over-the-air (FOTA) updates, more commonly known as OTA updates. A major version of iOS tends to be released every time a new type of iPhone is launched, (about once a year) and is normally free, although iPod Touch users were formerly required to pay for the update. Apple upgrades its

products' hardware periodically (approximately yearly). There have been eight major releases of iPhone (original iPhone, iPhone 3G, iPhone 3GS, iPhone 4, iPhone 4S, iPhone 5, iPhone 5C, and iPhone 5S), five of iPod Touch (first to fourth generations and fifth generation), five of iPad (first generation, iPad 2, third and fourth generations, and iPad Air), and two of iPad Mini (first and second generations).

iPhone (first generation)	iPhone 3G	iPhone 3GS	iPhone 4	iPhone 4S	iPhone 5	iPhone 5C	iPhone 5S
							
iPhone OS 1.0	iPhone OS 2.0	iPhone OS 3.0	iOS 4.0 (GSM) iOS 4.2.5 (CDMA)	iOS 5.0	iOS 6.0	iOS 7.0	
iPhone OS 3.1.3	iOS 4.2.1	iOS 6.1.6			iOS 7.1.1		

2.1-Figure. Apple iPhones models.

First generation	Second generation	Third generation	Fourth generation	Fifth generation
				
iPhone OS 1.1	iPhone OS 2.1.1 iPhone OS 3.1.1 (8 GB "MC" model)	iPhone OS 3.1.1	iOS 4.1	iOS 6.0
iPhone OS 3.1.3	iOS 4.2.1	iOS 5.1.1[67]	iOS 6.1.6	iOS 7.1.1

2.2-Figure. Apple iPod Touch.

iPad (first generation)	iPad 2	iPad (third generation)	iPad (fourth generation)	iPad Air
				
A1219 (Wi-Fi) A1337 (Wi-Fi + 3G)	A1395 (Wi-Fi) A1396 (GSM model) A1397 (CDMA model)	A1416 (Wi-Fi) A1403 (Wi-Fi + Cellular (VZ)) A1430 (Wi-Fi + Cellular)	A1458 (Wi-Fi) A1460 (Wi-Fi + Cellular (MM)) A1459 (Wi-Fi + Cellular)	A1474 (Wi-Fi) A1475 (Wi-Fi + Cellular)
iPhone OS 3.2	iOS 4.3	iOS 5.1	iOS 6.0 (Wi-Fi) iOS 6.0.1 (Wi-Fi + Cellular)	iOS 7.0.3
iOS 5.1.1[67]			iOS 7.1.1	

2.3 - Figure. Apple iPad.

First generation	Second generation
	
iOS 6.0 (Wi-Fi) iOS 6.0.1 (Wi-Fi + Cellular)	iOS 7.0.3
iOS 7.1.1	

2.4 - Figure. Apple iPad Mini.

Second generation	Third generation	Third generation Rev A
		
iOS 4.1 (Apple TV 4.0)	iOS 5.1 (Apple TV 5.0)	iOS 6.1 (Apple TV 5.2)
	iOS 7.0.4 (Apple TV 6.0.2)	

2.5 - Figure. Apple TV.

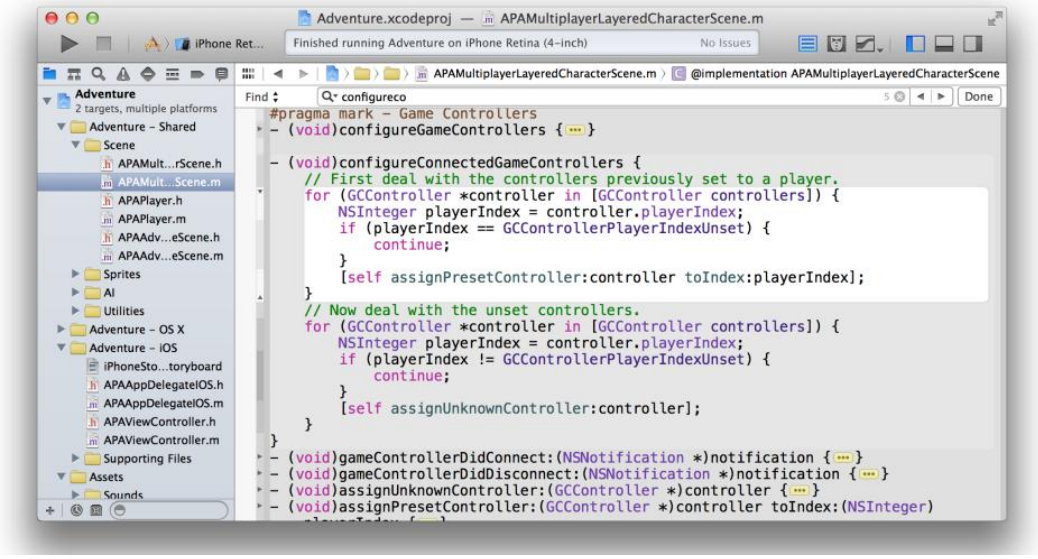
About Xcode

Develop iOS and Mac apps with Xcode, Apple's integrated development environment (IDE). Xcode provides tools to manage your entire development workflow—from creating your app, to testing, optimizing, and submitting it to the App Store. Xcode is built to help you build great apps for iPad, iPhone, and Mac.

2.6.Figure. Icon Xcode.

The Xcode interface integrates code editing, user interface design, asset management, testing, and debugging within a single workspace window. The window reconfigures its content as you work. For example, select a file in one area, and an appropriate editor opens in another area. Select a symbol or user interface object, and its documentation appears in a nearby pane. You can focus on a task by displaying only what you need, such as only your source code or only your user interface layout. Or you can work with your code and UI layout side by side. You can further customize your environment by opening multiple windows and multiple tabs per window.

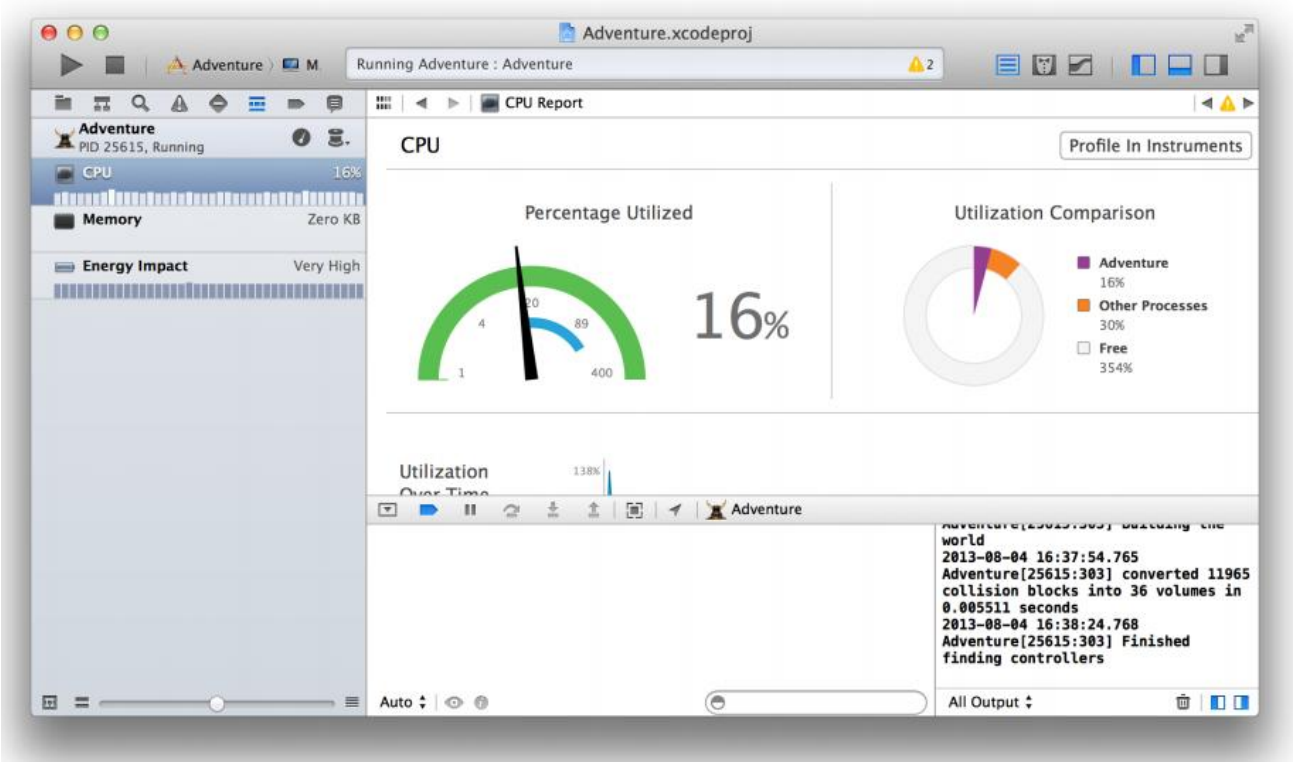
Xcode checks your source code as you type it, and when Xcode notices a mistake, the source code editor highlights the error. The source code editor then offers to fix it. Xcode speeds up your typing with intelligent code completion. Xcode reduces your typing by offering ready-to-use code snippets and source file templates. You can easily configure the source editor to display multiple views of the same file or to view multiple related files at once. Search-and-replace and refactoring operations help you make extensive changes to your code quickly and safely. With these and other capabilities, Xcode makes it easier for you to write better code more quickly than you thought possible.



2.7.Figure. Source code application in Xcode.

Interface Builder is a visual design editor that's integrated into Xcode. Use Interface Builder to create the user interfaces of your apps by assembling windows, views, controls, menus, and other elements from a library of configurable objects. Graphically connect these object to your implementation code. With the Auto Layout feature, define rules for your objects so that they automatically adjust to screen size, device orientation, window size, and localization. The asset catalog in Xcode helps you manage the many images you'll use for your app's user interface—items such as icons, launch images for iOS devices, and custom artwork. With the particle emitter editor in Xcode, you can enhance your iOS or Mac game by adding animation effects involving moving particles such as snow, sparks, and smoke. For Mac apps, the Scene Kit editor helps you work with scenes created in 3D authoring tools and exported as COLLADA Data Asset Exchange (DAE) files. Build your app, and Xcode launches it and immediately starts a debugging session. If you are running an iOS app, Xcode launches it either in iOS Simulator or on an iOS device connected to your Mac. If you are running a Mac app, Xcode launches it directly on your Mac. You can debug your app directly within the source editor with graphical tools like data tips and the variables Quick Look. The debug area and the debug navigator let you carefully control the execution of your app while you examine the code.

Debug gauges display your app's resource consumption to help you identify problems before your users do.



2.8.Figure. Process debugging

To help you build a better app, Xcode includes unit testing tools. Unit tests are programs that automatically exercise the features of your application and check the results. You should create unit tests that automatically exercise the features of your application, and then monitor the results of the tests and fix any issues from the Xcode test navigator. You can use the Xcode service, available in OS X Server, to automate the execution of unit tests. From Xcode on your development Mac, you create bots that run on a separate server to execute your unit tests either periodically or on every source code commit. In addition to running unit tests, bots automatically perform static analysis on your code, build your app, and archive it for distribution to testers or the App Store. While performing these continuous integrations of your app, bots report build errors and warnings, static analyzer problems, and unit test failures. While you work, Xcode automatically saves changes to source and project files. This feature requires no configuration, because Xcode continuously tracks your changes and saves them. You can revert a file to a previous state with Undo and Revert Document commands. You can revert an

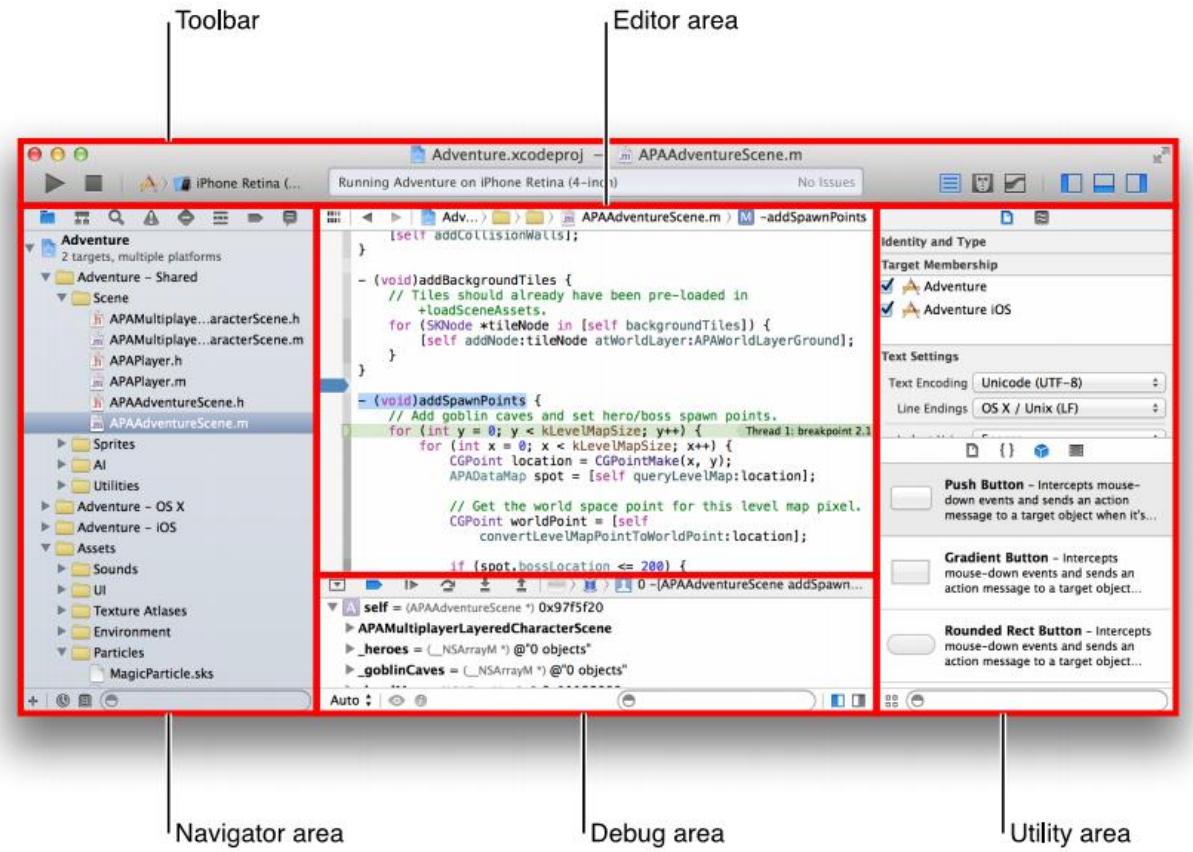
entire project to a previous “snapshot” of a known working version with the Restore Snapshot command. Snapshots provide an easy way to back up the current version of your project. Xcode automatically creates a snapshot before you perform any mass-editing operation, and you can set Xcode to automatically create snapshots in other circumstances. You can also create snapshots manually, such before you add a new feature to your app. To keep track of changes at a fine-grained level, use the Xcode source control management features. Xcode supports two popular source control systems: Git and Subversion. You can access remote Git and Subversion source code repositories, and you can create local Git repositories. Using the Xcode service, available with OS X Server, you can host Git repositories on your own server. While you’re coding, Xcode makes detailed technical information available at your fingertips. When you want it, Quick Help keeps concise API information always in view, and Xcode application help is always close at hand with step-by-step instructions for performing common Xcode tasks. Xcode includes extensive documentation for using Xcode, and it provides comprehensive SDK documentation, including programming guides, tutorials, sample code, detailed framework API references, and video presentations by Apple engineers, all viewable from the Xcode documentation viewer. As updated documentation becomes available, it downloads automatically in the background.

Most of your development time is spent on coding tasks, but to develop for the App Store, you need to perform a number of administrative tasks throughout the lifetime of your app. In addition to using Xcode, you’ll use the Member Center web tool to manage developer program accounts and entitlements, and you’ll use the iTunes Connect web tool to check the status of your contracts, set up tax and banking information, obtain sales and finance reports, and manage metadata about the app.

Xcode project configurations help prepare your app for distribution to beta testers and for submission to the App Store. Submitting your app is a multistep process that begins when you sign into iTunes Connect and supply necessary

product information. In Xcode, you create an archive of your project and submit it to the store. When your app is approved, you use iTunes Connect to release it by setting the date. (If you are distributing your Mac app outside the store, you follow a slightly different process.)

Perform your core development tasks in the Xcode workspace window. The workspace window is your primary interface for creating and managing projects. The workspace window automatically adapts itself to the task at hand, and you can further configure the window to fit your work style. You can open as many workspace windows as you need. The workspace window has five main components.



2.9.Figure. The workspace window Xcode.

The workspace window always includes the editor area. When you select a project file, its contents appear in the editor area, where Xcode opens the file in an

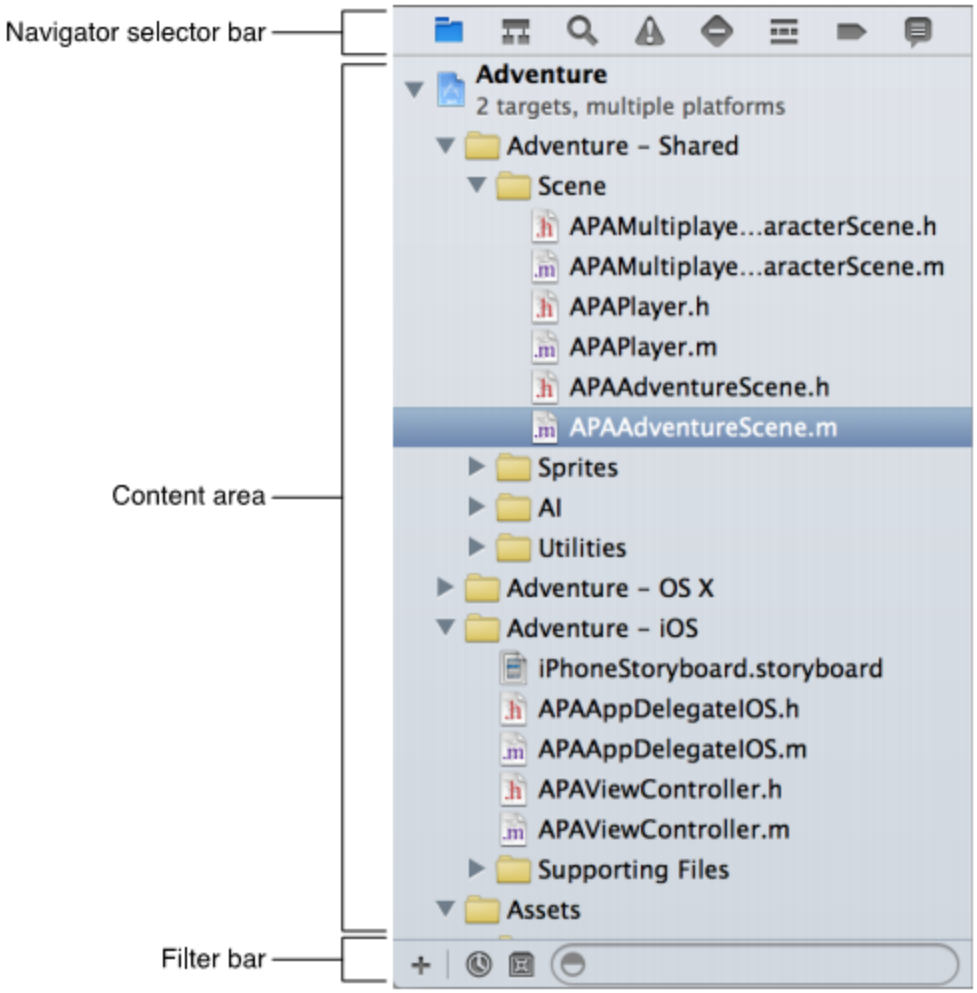
appropriate editor. The workspace window can also display three optional areas. You hide or show these optional areas by using buttons in the view selector in the toolbar:

 Show and hide the navigator area. The navigator area is where you view and maneuver through files and other facets of your project.

 Show and hide the debug area. The debug area is where you control program execution and debug code.

 Show and hide the utility area. You use the utility area for several purposes, most commonly to view and modify attributes of a file and to add ready-made resources to your project.

Access files, symbols, unit tests, diagnostics, and other facets of your project from the navigator area. In the navigator selector bar, you choose the navigator suited to your task. The content area of each navigator gives you access to relevant portions of your project, and each navigator's filter bar allows you to restrict the content that is displayed.



2.10.Figure. Navigate Workspace Xcode.

Choose from these options in the navigator selector bar:

Project navigator. Add, delete, group, and otherwise manage files in your project, or choose a file to view or edit its contents in the editor area.

Symbol navigator. Browse the class hierarchy of the symbols in your project.

Find navigator. Use search options and filters to quickly find any string within your project.

Issue navigator. View issues such as diagnostics, warnings, and errors found when opening, analyzing, and building your project.

Test navigator. Create, manage, run, and review unit tests.

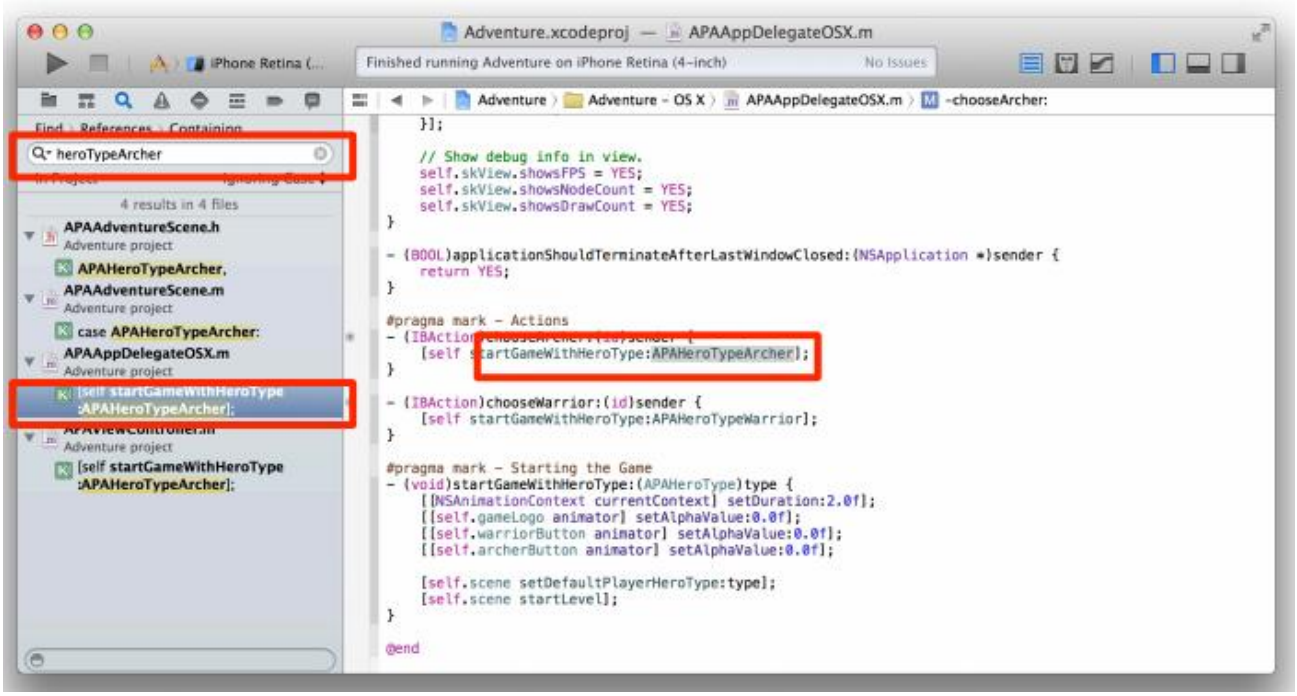
Debug navigator. Examine the running threads and associated stack information at a specified point or time during program execution.

Breakpoint navigator. Fine-tune breakpoints by specifying characteristics such as triggering conditions.

Log navigator. View the history of your build, run, debug, continuous integration, and source control tasks.

The following screenshot shows a number of search results appearing in the find navigator's content area.

One of the results is selected, and its text string appears in the source editor. Configure the editor area for a given task with the editor selector, found in the toolbar.



2.11.Figure. Search code in Xcode.

The editor selector offers three controls:

Standard editor. Fills a single editor pane with the contents of the selected file.

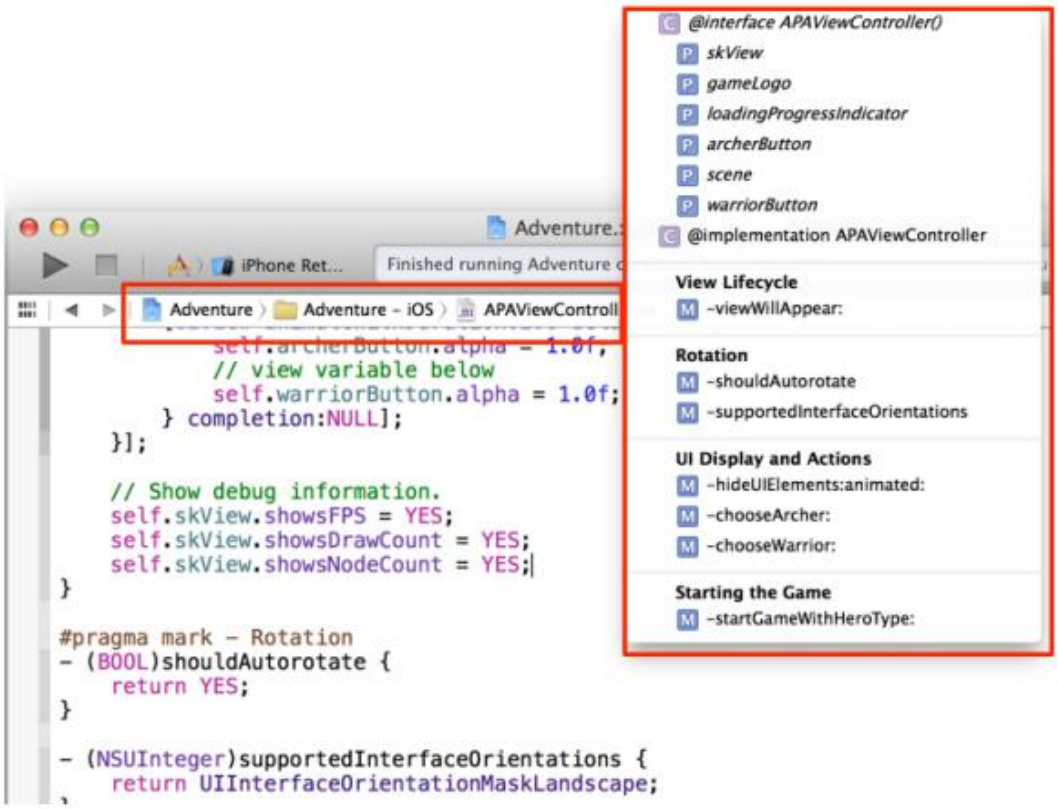
Assistant editor. Presents a separate editor pane with content logically related to that in the standard editor pane.

Version editor. Shows the differences between the selected file in one pane and another version of that same file in a second pane.

This screenshot shows an implementation file, APAAdventureScene.m, open in the standard editor pane. The three optional workspace areas—navigator, debugger, and utility—are hidden to maximize the editor's content display.Within the source code editor, the assistant pane displays the implementation file's associated header file, APAAdventureScene.h.

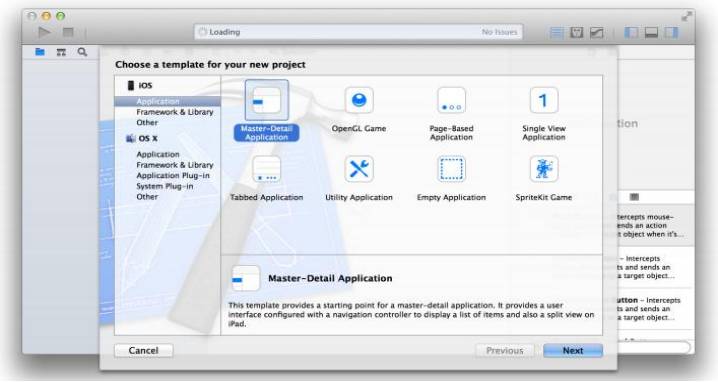
2.12.Figure. Standard editor pane Xcode.

Click a segment in the hierarchical path menu to see a pop-up menu of related sub items. For example, if the segment identifies the name of the project, you can use the jump bar to find folders within the project. If the segment identifies the name of a folder, you can use the jump bar to find files within the folder. If the segment identifies the name of a source file, you can use the jump bar to find symbols within the file.



2.13.Figure. The hierarchical path menuXcode.

Apps you create in Xcode require a project, which keeps the necessary files and resources organized. You start a project by choosing File > New > New Project. Xcode opens a new workspace window and displays a dialog in which you choose a project template. Xcode provides built-in templates for developing common styles of iOS and Mac apps. These templates include essential project files that help you start your development effort quickly.



2.14.Figure. Create new Project in Xcode.

Advantages Objective C

Objective-C is a general-purpose, object-oriented programming language that adds Smalltalk-style messaging to the C programming language. It is the main programming language used by Apple for the OS X and iOS operating systems, and their respective application programming interfaces (APIs), Cocoa and Cocoa Touch.

The Objective-C language was chosen for a variety of reasons. First and foremost, it's an object-oriented language. The kind of functionality that's packaged in the Cocoa frameworks can only be delivered through object-oriented techniques. Second, because Objective-C is an extension of standard ANSI C, existing C programs can be adapted to use the software frameworks without losing any of the work that went into their original development. Because Objective-C incorporates C, you get all the benefits of C when working within Objective-C. You can choose when to do something in an object-oriented way (define a new class, for example) and when to stick to procedural programming techniques (define a structure and some functions instead of a class).

Moreover, Objective-C is a fundamentally simple language. Its syntax is small, unambiguous, and easy to learn. Object-oriented programming, with its self-conscious terminology and emphasis on abstract design, often presents a steep learning curve to new recruits. A well-organized language like Objective-C can make becoming a proficient object-oriented programmer that much less difficult.
Compared to other object-oriented languages based on C, Objective-C is very dynamic. The compiler preserves a great deal of information about the objects themselves for use at runtime. Decisions that otherwise might be made at compile time can be postponed until the program is running. Dynamism gives Objective-C programs unusual flexibility and power. For example, it yields two big benefits that are hard to get with other nominally object-oriented languages:

- Objective-C supports an open style of dynamic binding, a style that can accommodate a simple architecture for interactive user interfaces. Messages are

not necessarily constrained by either the class of the receiver or even the method name, so a software framework can allow for user choices at runtime and permit developers freedom of expression in their design. (Terminology such as dynamic binding, message, class, and receiver are explained in due course in this document.)

- Dynamism enables the construction of sophisticated development tools. An interface to the runtime system provides access to information about running applications, so it's possible to develop tools that monitor, intervene, and reveal the underlying structure and activity of Objective-C applications.

iPhone development is a great and fun hobby, which can bring in a significant amount of income. When you develop an application for the iPhone OS, you want to make it as user intuitive and as user friendly as possible. The way you do this though is completely your choice. You can program an iPhone app in several different types of languages, I personally prefer Objective-C (or any OOP language) over, lets say, HTML etc. As you begin to develop applications, you will realize that using a Objective-C is a lot easier, and it creates a better outcome. Obj-C is harder to learn if you do not already have prior knowledge of an OOP language (preferably C, C++), but once you get the hang of it, you will find out it is a much better language than Titanium or HTML. Objective -C has more methods and classes, and most of the apple documentation is geared towards Objective-C, so it is to your benefit to use Objective -C when creating iPhone or Mac apps.

Data and Operations

Programming languages have traditionally divided the world into two parts—data and operations on data. Data is static and immutable, except as the operations may change it. The procedures and functions that operate on data have no lasting state of their own; they're useful only in their ability to affect data.

This division is, of course, grounded in the way computers work, so it's not one that you can easily ignore or push aside. Like the equally pervasive distinctions

between matter and energy and between nouns and verbs, it forms the background against which we work. At some point, all programmers—even object-oriented programmers—must lay out the data structures that their programs will use and define the functions that will act on the data.

With a procedural programming language like C, that's about all there is to it. The language may offer various kinds of support for organizing data and functions, but it won't divide the world any differently. Functions and data structures are the basic elements of design.

Object-oriented programming doesn't so much dispute this view of the world as restructure it at a higher level. It groups operations and data into modular units called objects and lets you combine objects into structured networks to form a complete program. In an object-oriented programming language, objects and object interactions are the basic elements of design.

Every object has both state (data) and behavior (operations on data). In that, they're not much different from ordinary physical objects. It's easy to see how a mechanical device, such as a pocket watch or a piano, embodies both state and behavior. But almost anything that's designed to do a job does, too. Even simple things with no moving parts such as an ordinary bottle combine state (how full the bottle is, whether or not it's open, how warm its contents are) with behavior (the ability to dispense its contents at various flow rates, to be opened or closed, to withstand high or low temperatures).

It's this resemblance to real things that gives objects much of their power and appeal. They can not only model components of real systems, but equally as well fulfill assigned roles as components in software systems.

C structures and functions are able to express significant abstractions, but they maintain the distinction between data and operations on data. In a procedural programming language, the highest units of abstraction still live on one side or the other of the data-versus-operations divide. The programs you design must always reflect, at the highest level, the way the computer works.

Object-oriented programming languages don't lose any of the virtues of structures and functions—they go a step further and add a unit capable of abstraction at a higher level, a unit that hides the interaction between a function and its data.

Suppose, for example, that you have a group of functions that act on a particular data structure. You want to make those functions easier to use by, as far as possible, taking the structure out of the interface. So you supply a few additional functions to manage the data. All the work of manipulating the data structure—allocating memory for it, initializing it, getting information from it, modifying values within it, keeping it up to date, and freeing its memory—is done through the functions. All the user does is call the functions and pass the structure to them.

With these changes, the structure has become an opaque token that other programmers never need to look inside. They can concentrate on what the functions do, not on how the data is organized. You've taken the first step toward creating an object.

The next step is to give this idea support in the programming language and completely hide the data structure so that it doesn't even have to be passed between the functions. The data becomes an internal implementation detail; all that's exported to users is a functional interface. Because objects completely encapsulate their data (hide it), users can think of them solely in terms of their behavior.

With this step, the interface to the functions has become much simpler. Callers don't need to know how they're implemented (what data they use). It's fair now to call this an object.

The hidden data structure unites all the functions that share access to it. So an object is more than a collection of random functions; it's a bundle of related behaviors that are supported by shared data. To use a function that belongs to an object, you first create the object (thus giving it its internal data structure), and then

tell the object which function it should perform. You begin to think in terms of what the object does, rather than in terms of the individual functions.

This progression from thinking about functions and data structures to thinking about object behaviors is the essence of learning object-oriented programming. It may seem unfamiliar at first, but as you gain experience with object-oriented programming, you find it's a more natural way to think about things. Everyday programming terminology is replete with analogies to real-world objects of various kinds—lists, containers, tables, controllers, even managers. Implementing such things as programming objects merely extends the analogy in a natural way.

A programming language can be judged by the kinds of abstractions that it enables you to encode. You shouldn't be distracted by extraneous matters or forced to express yourself using a vocabulary that doesn't match the reality you're trying to capture.

If, for example, you must always tend to the business of keeping the right data matched with the right procedure, you're forced at all times to be aware of the entire program at a low level of implementation. While you might still invent programs at a high level of abstraction, the path from imagination to implementation can become quite tenuous—and more and more difficult as programs become bigger and more complicated.

By providing another, higher level of abstraction, object-oriented programming languages give you a larger vocabulary and a richer model to program in.

The Object Model

The insight of object-oriented programming is to combine state and behavior—data and operations on data—in a high-level unit, an object, and to give it language support. An object is a group of related functions and a data structure that serves those functions. The functions are known as the object's methods, and the fields of its data structure are its instance variables. The methods wrap around

the instance variables and hide them from the rest of the program, as 2.15. Figure illustrates.

2.15. Figure. An object

If you've ever tackled any kind of difficult programming problem, it's likely that your design has included groups of functions that work on a particular kind of data—implicit "objects" without the language support. Object-oriented programming makes these function groups explicit and permits you to think in terms of the group, rather than its components. The only way to an object's data, the only interface, is through its methods.

By combining both state and behavior in a single unit, an object becomes more than either alone; the whole really is greater than the sum of its parts. An object is a kind of self-sufficient "subprogram" with jurisdiction over a specific functional area. It can play a full-fledged modular role within a larger program design.

Mechanisms of Abstraction

To this point, objects have been introduced as units that embody higher-level abstractions and as coherent role-players within an application. However, they couldn't be used this way without the support of various language mechanisms. Two of the most important mechanisms are encapsulation and polymorphism.

Encapsulation keeps the implementation of an object out of its interface, and polymorphism results from giving each class its own namespace. The following sections discuss each of these mechanisms in turn.

To design effectively at any level of abstraction, you need to be able to leave details of implementation behind and think in terms of units that group those

details under a common interface. For a programming unit to be truly effective, the barrier between interface and implementation must be absolute. The interface must encapsulate the implementation—that is, hide it from other parts of the program. Encapsulation protects an implementation from unintended actions and from inadvertent access.

In C, a function is clearly encapsulated; its implementation is inaccessible to other parts of the program and protected from whatever actions might be taken outside the body of the function. In Objective-C, method implementations are similarly encapsulated, but more importantly so are an object's instance variables. They're hidden inside the object and invisible outside it. The encapsulation of instance variables is sometimes also called information hiding.

It might seem, at first, that hiding the information in instance variables might constrain your freedom as a programmer. Actually, it gives you more room to act and frees you from constraints that might otherwise be imposed. If any part of an object's implementation could leak out and become accessible or a concern to other parts of the program, it would tie the hands both of the object's implementer and of those who would use the object. Neither could make modifications without first checking with the other.

Suppose, for example, that you're interested in the Faucet object being developed for the program that models water use and you want to incorporate it in another program you're writing. Once the interface to the object is decided, you don't have to be concerned when others work on it, fix bugs, and find better ways to implement it. You get the benefit of these improvements, but none of them affects what you do in your program. Because you depend solely on the interface, nothing they do can break your code. Your program is insulated from the object's implementation.

Moreover, although those implementing the Faucet object might be interested in how you use the class and might try to make sure it meets your needs, they don't have to be concerned with the way you write your code. Nothing you do

can touch the implementation of the object or limit their freedom to make implementation changes in future releases. The implementation is insulated from anything that you or other users of the object might do.

Polymorphism

The ability of different objects to respond, each in its own way, to identical messages is called polymorphism. Polymorphism results from the fact that every class lives in its own namespace. The names assigned within a class definition don't conflict with names assigned anywhere outside it. This is true both of the instance variables in an object's data structure and of the object's methods:

- Just as the fields of a C structure are in a protected namespace, so are an object's instance variables.
- Method names are also protected. Unlike the names of C functions, method names aren't global symbols. The name of a method in one class can't conflict with method names in other classes; two very different classes can implement identically named methods.

Method names are part of an object's interface. When a message is sent requesting that an object do something, the message names the method the object should perform. Because different objects can have methods with the same name, the meaning of a message must be understood relative to the particular object that receives the message. The same message sent to two different objects can invoke two distinct methods.

The main benefit of polymorphism is that it simplifies the programming interface. It permits conventions to be established that can be reused in class after class. Instead of inventing a new name for each new function you add to a program, the same names can be reused. The programming interface can be described as a set of abstract behaviors, quite apart from the classes that implement them.

Overloading: The terms polymorphism and parameter overloading refer basically to the same thing, but from slightly different points of view.

Polymorphism takes a pluralistic point of view and notes that several classes can each have a method with the same name. Parameter overloading takes the point of the view of the method name and notes that it can have different effects depending on the parameters passed to it. Operator overloading is similar. It refers to the ability to turn operators of the language (such as == and + in C) into methods that can be assigned particular meanings for particular kinds of objects. Objective-C implements polymorphism of method names, but not parameter or operator overloading.

For example, suppose you want to report the amount of water used by an Appliance object over a given period of time. Instead of defining an amountConsumed method for the Appliance class, an amountDispensedAtFaucet method for a Faucet class, and a cumulativeUsage method for a Building class, you can simply define awaterUsed method for each class. This consolidation reduces the number of methods used for what is conceptually the same operation.

Polymorphism also permits code to be isolated in the methods of different objects rather than be gathered in a single function that enumerates all the possible cases. This makes the code you write more extensible and reusable. When a new case comes along, you don't have to reimplement existing code; you need only to add a new class with a new method, leaving the code that's already written alone.
For example, suppose you have code that sends a draw message to an object. Depending on the receiver, the message might produce one of two possible images. When you want to add a third case, you don't have to change the message or alter existing code; you merely allow another object to be assigned as the message receiver.

Classes are objects

Each class is an instance of a meta-class automatically created and managed by the run-time. We can define class methods, pass classes as arguments, put them in collections and so on. To create an object, we just send a message to the class

we want to instantiate. No need to reinvent a "factory" system. No need for a specific constructor mechanism at the language level. This helps keeping the language simple and powerful.

Dynamic typing

As in Ruby, Python, Smalltalk, Groovy… Extremely useful because we don't always know beforehand what our objects are going to be at run-time. Dynamic typing in Objective-C is simple to use. For example, this declares a variable that can hold a reference to an object:

```
id myObject;
```

Optional static typing

Still, Objective-C also has support for static typing. Best of both worlds. This declares a variable that can hold a reference to an object of class (or subclass of) NSView:

```
NSView *myObject;
```

Categories

Categories let us define new methods and add them to classes for which we don't have the source code (such as the standard Cocoa classes provided by Apple). This makes it easy to extend classes without resorting to subclassing. Extremely useful to adapt existing classes to the requirements of frameworks we want to use or create.

Message sending

We interact with objects by sending them messages. Often, the receiver of a message will have a method that directly matches the message (i.e. that has the same name, or, in Objective-C terms, the same selector). In this case the method will be invoked. But this is not the only possible outcome, as an object can choose to handle a message in other ways such as forwarding it to another object, broadcasting it to a number of objects, introspecting it and applying custom logic, etc.

Expressive message syntax

Message patterns in Objective-C are like natural language sentences with holes in them (prefixed with colons). When we write code that sends a message to an object, we fill the holes with actual values, creating a meaningful sentence. This way of denoting messages comes from Smalltalk and makes the code very expressive.

Example, sending a message to an ordered collection, asking it to insert a given object at index 10:

```
[myCollection insert:myObject atIndex:10]
```

A message sending expressions can be read like a sentence where the receiver is the subject and the message is the rest of the sentence (for instance, an action that we would like the receiver to perform): "myCollection insert myObject at index 10".

Introspection

Introspecting objects is easy. For example, we can ask an object for its class like this:

```
[myObject class]
```

Determine if an object has a method "foo":

```
[myObject respondsToSelector:@selector(foo)]
```

Ask an object for the signature of its method "foo":

```
[myObject methodSignatureForSelector:@selector(foo)]
```

Ask a class whether it is a subclass of another class:

```
[class1 isSubclassOfClass:class2]
```

Dynamic run-time

Objective-C has a dynamic run-time. It allows crafting messages at run-time, dynamically creating classes, dynamically adding methods to existing classes, changing method implementations and so on…

For a nice example of what can be achieved with Objective-C introspection and dynamism you can have a look at this article I wrote about the graphical object browser provided by F-Script.

Automatic garbage collection

The automatic garbage collector runs in its own thread, concurrently with the application code. It uses a generational model to improve its efficiency by targeting in priority memory zones that are more likely to be garbage. It works for objects and also for raw C memory blocks allocated with theNSAllocateCollactable() and similar functions. malloc() works as usual, providing control over memory not managed by the collector.

The garbage collector is an opt-in service: you can choose to not make use of it in your application and instead rely on a reference counting system. This system includes a rather ingenious delayed release mechanism that goes a long way to reduce the burden of manual reference counting.
Note that at the time of this writing, the automatic garbage collector is not available on the iPhone.

C inside

Objective-C is primarily an object-oriented extension to the C language and constitutes a superset of C. This means that the raw power of C is available, and that C libraries can be accessed directly (as you know, there is quite a number of them available out there.). Beside, this creates a symbiotic relationship between the language and the operating system, as Mac OS X, which is a UNIX system, is primarily written in C and, for the upper-level parts, in Objective-C.

C++ fluent

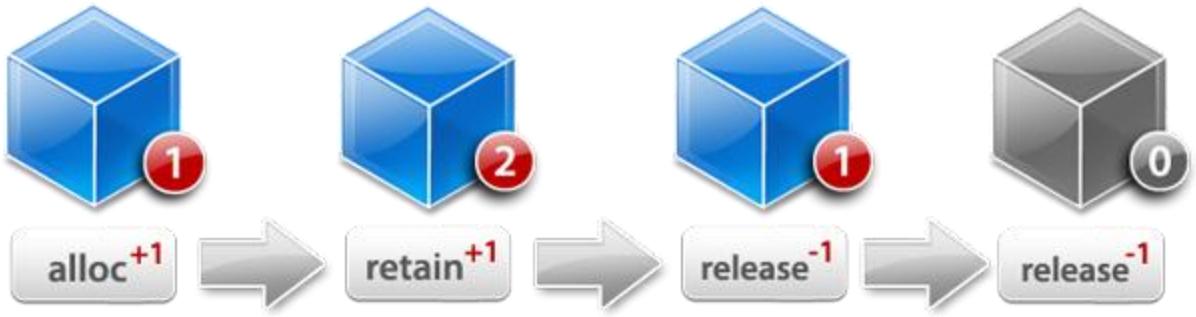
Not only is Objective-C a superset of C but it can also understand and call C++ code. Used in this configuration, the language is named Objective-C++ and allows mixing Objective-C and C++ code in the same code statements. It also allows directly using C++ libraries.

Simplicity

Objective-C's Smalltalk-inspired object system is leaning toward simplicity. Many features that tend to render languages complex (templates, overloading, multiple inheritance, etc.) are simply absent from Objective-C, which offer simpler programming models taking advantage of its dynamic nature.

Access to Apple technologies

Each new version of Mac OS X, and now, of the iPhone OS, is full of interesting new technologies which are directly available from Objective-C to play with. This contributes significantly to make Objective-C fun to use.



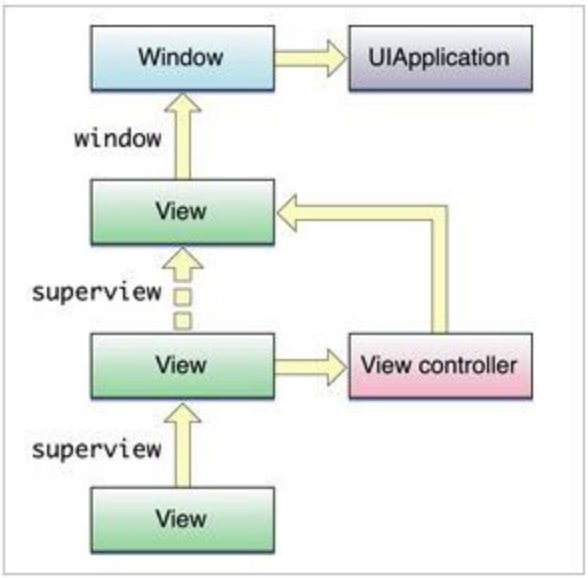
2.16.Figure. Memory Management in Objective C.

2.2. Designing and steps of doing task

This application created in Xcode Mac OS operating system.

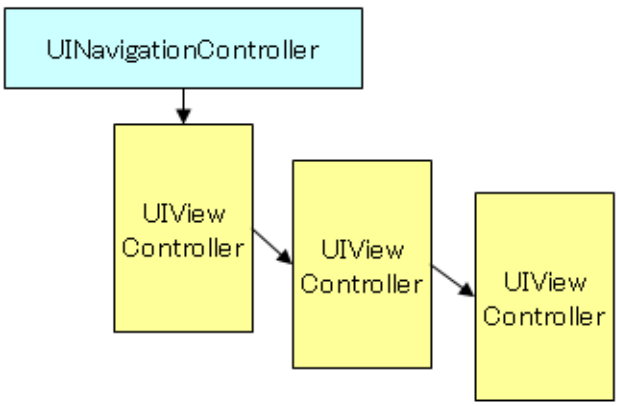
Used components :

1.UIViewController - vital link between an app's data and its visual appearance.



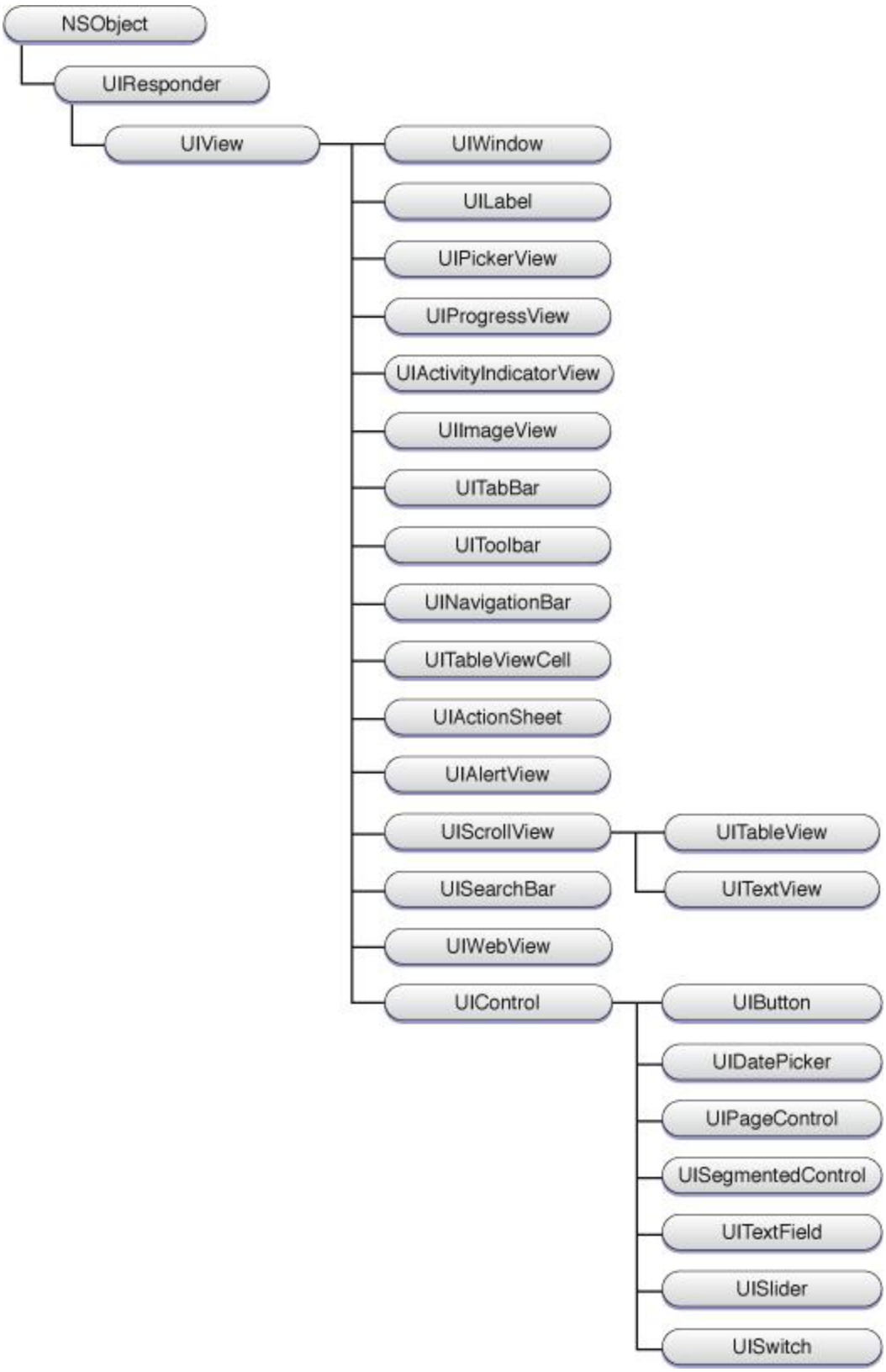
2.17. Figure. UIViewController scheme.

2. UINavigationController - implements a specialized view controller that manages the navigation of hierarchical content.



2.18. Figure. UINavigationController scheme.

3. UIView- defines a rectangular area on the screen and the interfaces for managing the content in that area.



2.19. UIView class hierarchy.

4.UIButton - implements a button on the touch screen.

5.UIImage- high-level way to display image data.

6.UILabel – Display static text on screen.

7.UITextView-Display dynamic text on screen.

8.UIImageView-Display image on screen.

Developer classes:

1.NSString – work with stirng.

2.NSMutableString – work with mutable string.

3.NSArray – work with array.

4.NSDictionary – work with dictionary.

5.NSTimer – timer class.

6.NSBundle – manage resources project.

Used frameworks:

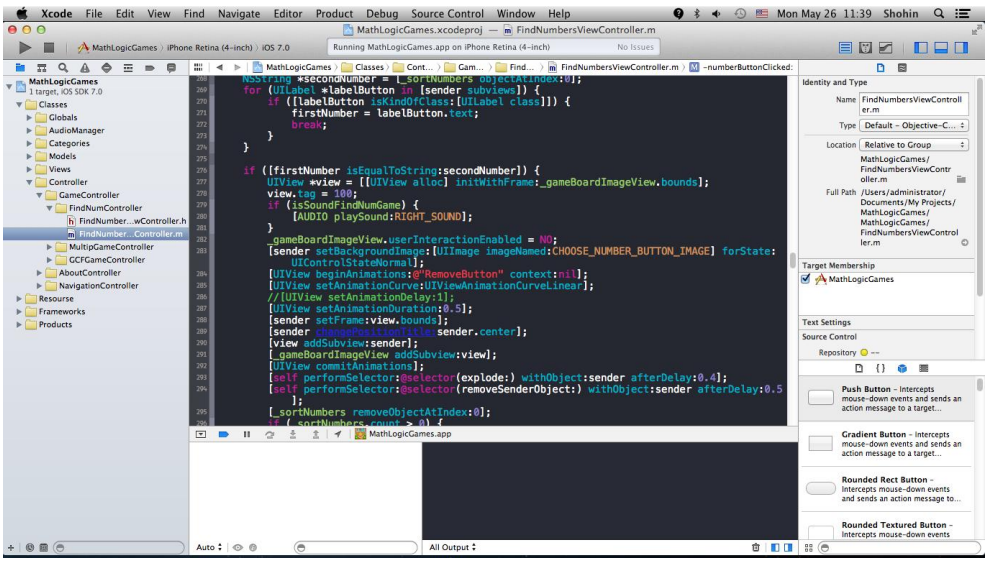
1.UIKit framework – UI components framework.

2.QuartzCore - graphical actions,effects and animations.

3.Foundation framework – developer classes.

4.CoreGraphics framework – shapes, geometry components.

5.AudioToolBox framework – audio framework.



2.20.Figure.Application in Xcode.

Steps of doing task:

1. Create main window game. In the main window display game menu;

2. Create games level;

3. Create sound class for games;

4. Custom components classes;

5. Create “Sonni top” (“Find number”) game;

5.1. Create menu for “Sonni top” (“Find number”) game;

5.2. Create game board for “Soni top” (“Find number”) game;

5.2.1. In game board play game “Soni top”. When the game board controller loaded, there display buttons with random numbers and at the left side board display a view which display level, timer and points.

5.3. So in the “Sonni top” game used animation, effects, etc.

6. Create “Ko`paytirish” (“Multiplication”) game.

6.1. Create menu for “Ko`paytirish” (“Multiplication”) game;

6.2. Create game board for “Ko`paytirish” (“Multiplication”) game;

6.2.1. In game board play game “Ko`paytirish”. When the game board controller loaded, there display buttons with random numbers and at the right side board display a view which display level, timer and points.

6.2.2. So in the “Ko`paytirish” game used animation, effects, etc.

5.2.3. Create “Help” controller for “Ko`paytirish” game. In the “Help” controller displayed multiplication table. With this multiplication table kids can learn multiplication table.

7. Create “EKUB o`yini” (“CGF game”) game;

7.1. Create menu for “EKUB o`yini” (“CGF game”) game;

7.2. Create game board for “EKUB o`yini” (“CGF game”) game;

7.2.1. In game board play game “EKUB o`yini”. When the game board controller loaded, there display buttons with random numbers and at the right side board display a view which display level, timer and points.

7.3. So in the “EKUB o`yini” game used animation, effects, etc.

Requirements: minimum iOS version - iOS 6.0,maximum iOS vesion - iOS 7.0, 4mb physical memory, minimum iOS device random access memory – 500kb.

2.3. Instruction for users

1.When you clicked to icon MathLogicGames from your iOS device screen, displayed this window:

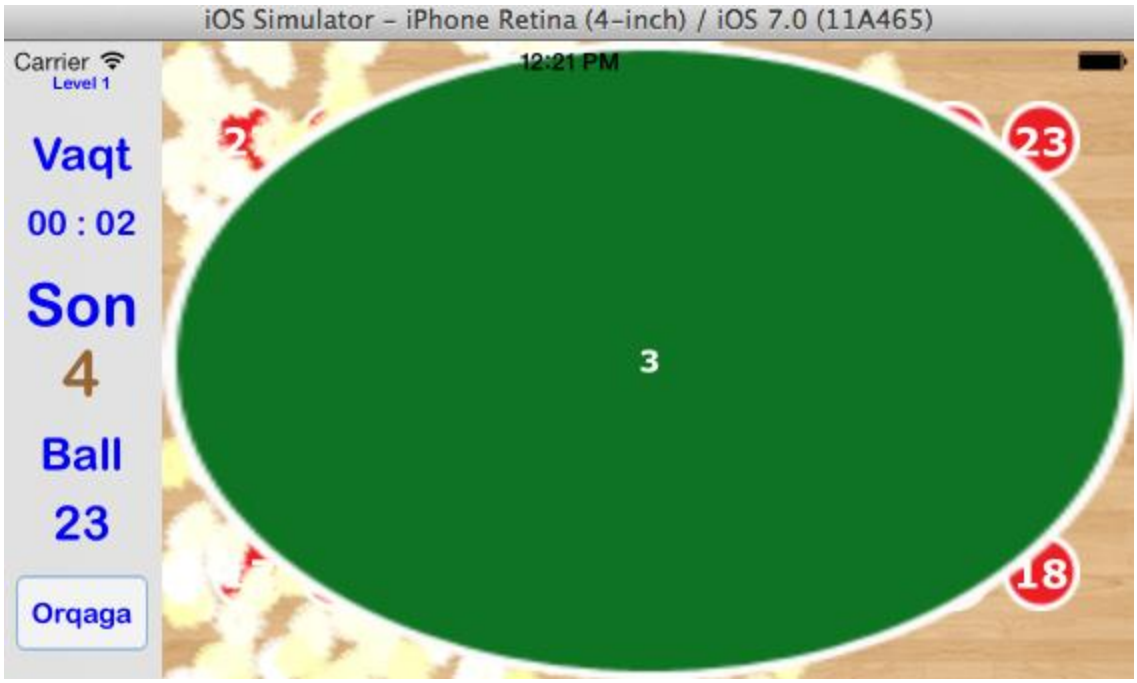
2.3.1.Figure. Main menu MathLogicGames.

2.The first game is “Sonni top”(“Find number”). This game mainly destine to learning reckon up. Play rule games is very easy. You must find the number into shape numbers and down up to number. For each find number append point to you. If you choose wrong number, your point is decrease. If you find correct all numbers, you win game and you has perhaps to go to next level. If your time is finish or your point equal to zero you lost game.

2.3.2.Figure. Main menu game “Sonni top”.

2.3.3.Figure. Levels game “Sonni top”.

2.3.4.Figure. Game board game “Sonni top”.



2.3.5.Figure. Process play game “Sonni top”.

2.3.6.Figure. Game over game “Sonni top”.

3.The second game is “Ko`paytirish”(“Multiplication”) game. This game mainly destine to learning multiplication table. Play rule games is very easy. You must find numbers into shape numbers, thereafter must multiplication chosen numbers equal the number. For each find numbers append point to you. If you choose wrong number, your point is decrease. If you find correct all numbers, you win game and you has perhaps to go to next level. If your time is finish or your point equal to zero you lost game.

2.3.7.Figure. Main menu game “Ko`paytirish”.

2.3.8.Figure. Levels game “Ko`paytirish”.

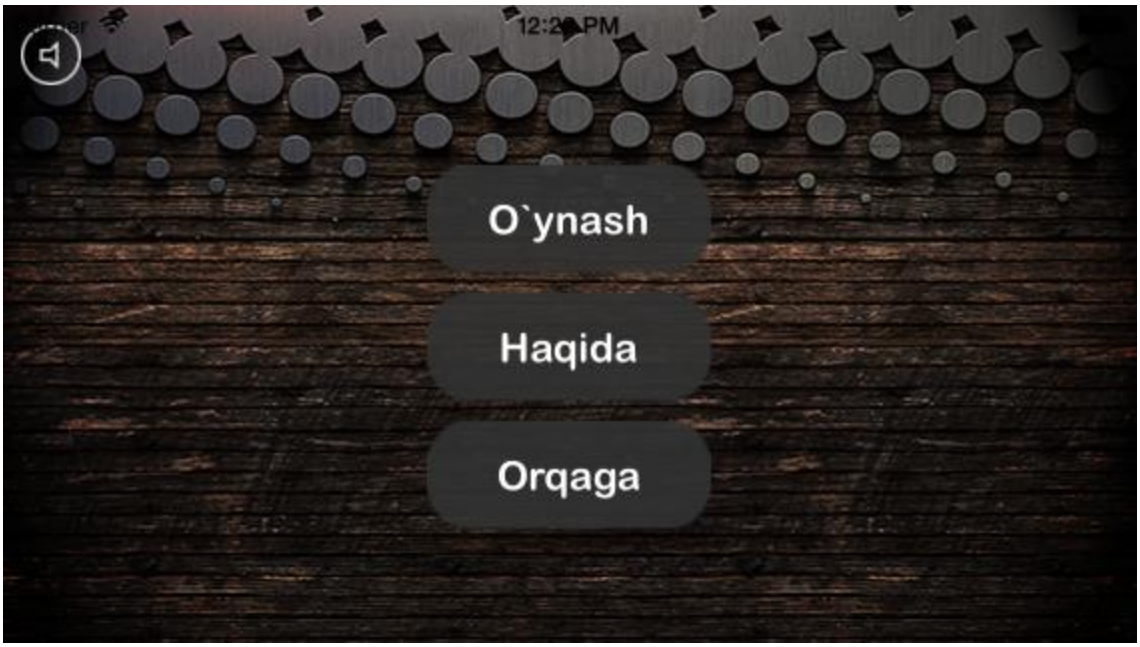
2.3.9.Figure. Game board game “Ko`paytirish”.

2.3.10.Figure. Process play game “Ko`paytirish”.

2.3.11.Figure. Game over game “Ko`paytirish”.

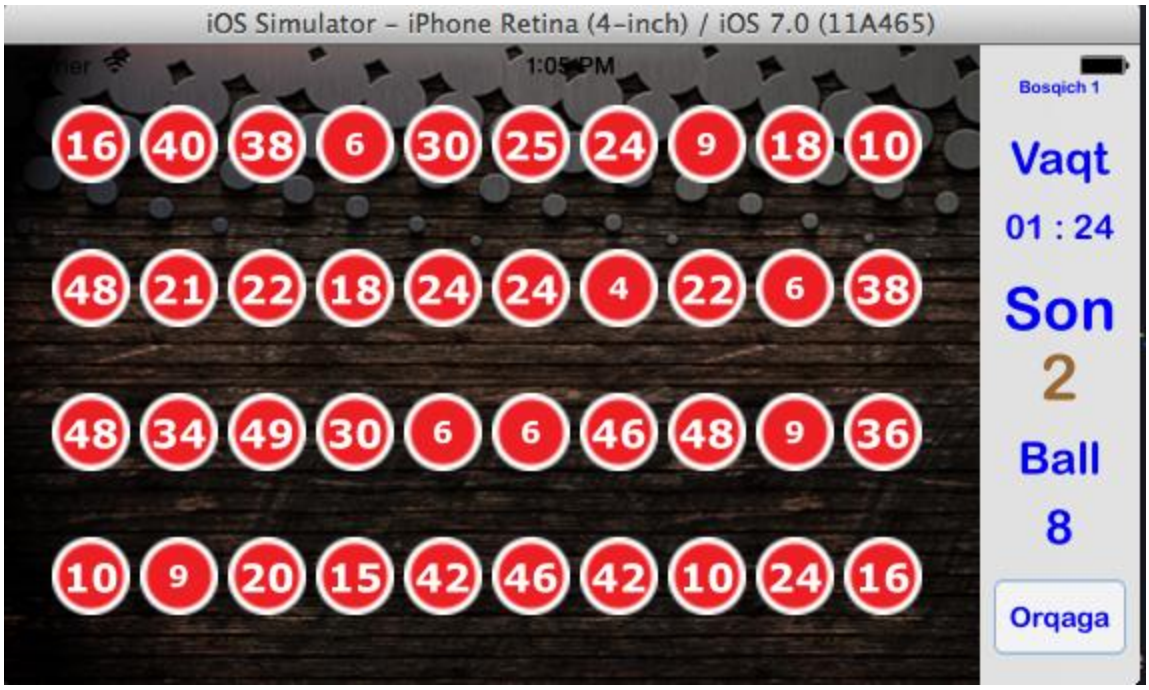
4.The third game is “EKUB o`yini”(“ GCF game”). This game mainly destine to learning GCF (Greatest Common Factor). Play rule games is very easy. You must find numbers into shape numbers, thereafter must GCF chosen numbers equal the number. For each find numbers append point to you. If you choose

wrong number, your point is decrease. If you find correct all numbers, you win game and you has perhaps to go to next level. If your time is finish or your point equal to zero you lost game.

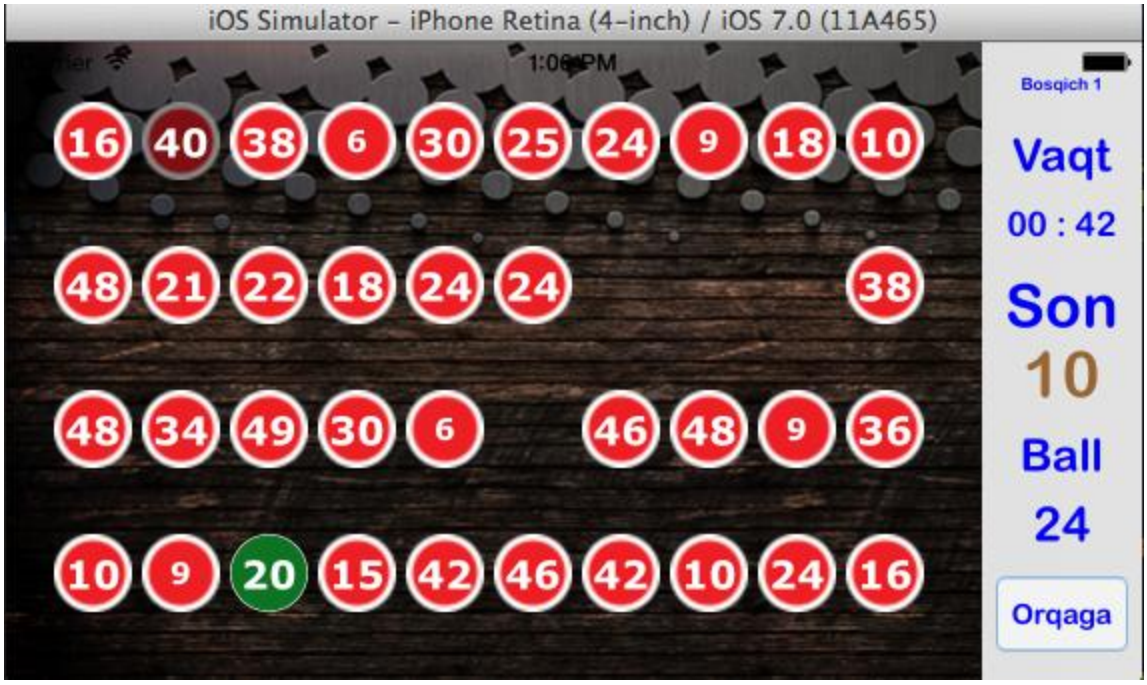


2.3.12.Figure. Main menu game "EKUB o`yini".

2.3.13.Figure. Levels game "EKUB o`yini".



2.3.14.Figure. Game board game “EKUB o`yini”.



2.3.15.Figure. Process play game “EKUB o`yini”.

2.3.16.Figure. Game over game “EKUB o`yini”.

3. LIFE SAFETY

3.1. Hypodynamy

Physical inactivity (from the Greek hypo - below and dynamis - force) - the weakening of muscle activity caused by a sedentary lifestyle and the limitation of motor aktivnosti.Gipodinamiya - a fairly common phenomenon in the modern world. The term is commonly used inactivity in narrow circles of specialists, such as, for example, a doctor of medical physical culture. It indicates decreased motor activity, a man little moves are not involved in sports or even little walks (eg, as a result of injury or various other diseases). Of course, the reasons for conducting such a sedentary lifestyle great mnozhestvo.V medicine there is a sufficiently close concept hypokinesia, which implies the reduction or complete absence of motor activity, usually caused quite objective reasons. Doctors refer to these reasons, some serious diseases, specific working conditions in a confined space, prolonged bed rest or plaster bandage and some others. The main difference from hypokinesia inactivity is that in the second case, the movements are carried out, but in a very small volume and a lack of muscle load on the machine. In both cases, the muscular load is minimal, leading to a decrease in muscle strength, decrease in size and weight are often referred to as muscular tkani.Gipodinamiyu scourge of modern civilization. The validity of this statement becomes evident when we remember how the human body was formed during the evolution of the animal world. Physical activity was necessary for our ancestors just to survive. Ancient people had to be in constant movement to obtain food and save a life for himself and his posterity. Thus, the need for the movement was founded in human genes, as one of the conditions for the normal functioning of the body in the harsh conditions of the program sredy.Geneticheskaya human remains unchanged over the past few millennia, but the way his life has undergone very significant changes. Movement for survival is no longer a necessity. Scientific and technological progress helps modern man to secure a comfortable living environment with

minimal physical exertion. Perfect cars, trains and planes move us across vast distances, tons of cargo lifted by simply pressing the desired button. Working day city dweller, like his vacation, often takes place in an armchair at the computer. Thus, all our movements restricted road from the entrance to the car. Even television channels we switch by remote control. Of course, a certain amount of movement inevitably makes everyone.

However, these very monotonous movement aimed at one group of muscles and did not contribute to the physical development of the organism.

The first reason would be, rather, will be such global terms as the urbanization of the population, the development of mechanical engineering, the automotive industry in particular, and other equally serious words. Urbanization of the population - is a long process. Once upon a time it was lots and lots of villages. People lived in the open air, fed exclusively on their own farm products and the gifts of nature, thereby moved a lot (plowed land, hunting, in general, worked comrades). And then moved into the city, cozy apartments and houses the central heating and water supply, which in fact is called urbanization. So, yesterday's farmers and began to move a little. And then there have been producing good cars, and at an affordable price. Bought a car and go to work even longer. Involve only the muscles involved in pressing the pedals. As a result, even less steel dvigatsya.So time there are many different specialties such as : manager (something), programmer, web designer and many others, that such sedentary occupations.

It is noteworthy that even ordinary students affected by this scourge (incidentally, this condition is known as a disease even twenty-first century - no more, no less.), They are very long and hard to sit. Imagine school 3:00 and then another home as much (probably preparing homework well, or, of course, playing computer games, as it often happens). While still in Russia Peter knew the solution to this problem : all students engaged martyrs standing behind tall pedestals. Here

also were intelligent people. But let us not digress from the topic, but rather continue. Are the following symptoms :

1. Lethargy
2. Drowsiness
3. General malaise
4. Decreased appetite
5. Reducing health
6. Insomnia

If you've watched a at least one symptom of this list, you should think about buying a gym membership.

If no action is taken for the prevention or treatment of inactivity, it may further lead to serious violations of various pathogenesis :

1. Vegetative-vascular dystonia.
2. Reduced muscle strength and endurance.
3. Abnormal perfusion of various organs and tissues.
4. Due to malnutrition and suffering brain, reducing its efficiency.
5. Fabrics eventually begin to atrophy.
6. Due endocrinological disorders may develop obesity.

That is, the predictions are not very comforting. What, then take to protect themselves from such severe complications ? Of course, it is necessary to raise the level of physical activity, move more, longer and more often walk in the fresh air. In general, there are very good you will run. How useful running.

Here is a list of several recommendations for the prevention of this insidious disease :

1. Perform daily morning exercises for 30 minutes (standard school charge).
2. Vespers walk for 20 minutes.
3. Walk Shopping is also prevention (especially with huge bags and other attributes).

4. Refusal of bad habits (smoking, poor and unhealthy food) is also an important factor.

5. Useful to do some physical work outdoors (eg, chop firewood in the country or weeding the flower bed with strawberries).

6. Ten minute jog will be very helpful.

7. Advisable to buy a cheap but effective trainer (bicycle, you can even buy a simple rope).

8. Enrol in any section (dancing, swimming pool, just a gym).

9. Include in your diet as much as possible supply of fruits, vegetables and honey are also advised to increase fluid intake (juices, fruit drinks, green tea - very good tones, soups and mineral waters).

And it is not the whole list of recommendations for the prevention of inactivity, because the world's a huge variety of different kinds of sports, from which both prevention can get a lot of pleasure. How to pick a sport for themselves. In general, you just need to live a healthy lifestyle and to be always on our toes. However, would not prevent a doctor's consultation.

Human genetic program was laid in view of sufficient physical activity. Let's consider what happens in the body due to the limitation of physical activity, ie gipodinamii.Pervymi suffer from inactivity, our muscles. Deprived of the necessary training, the muscles weaken, become flabby and eventually atrophy. Muscle weakness inevitably affects the work of all organs and systems of the human body. Restriction of muscular activity naturally leads to reduced activity of energy metabolism. Slows down the formation of energy-rich phosphate compounds, calcium slows output, causing increased bone fragility. Comes the inevitable decline in gas exchange, leading to a decrease in lung ventilation and the overall decrease in efficiency. That's why people leading a sedentary lifestyle, often tired more than his fellow actively moving. However, the consequences are far from exhausted inactivity general weakness, and fatigue. Reduced motor activity negatively affects the state of all the muscles of the human body. Muscle

tissue of vascular walls and the heart muscle also inevitably suffer the consequences of inactivity. Insufficient supply of oxygen and reduced blood flow often leads to impaired heart rate, coronary heart disease and hypertension. Physical inactivity is accompanied by a decrease in the load on the bone human skeleton. Such a reduction in the load in combination with impaired mineral metabolism reduces the strength of bone tissue, leading to skeletal deformities. Speaking about the impact of physical inactivity should also mention the influence of reduction of motor activity on the mental and emotional state. Reflex connections formed during the evolution imply close cooperation muscles with the nervous system. Physical inactivity is inevitably reflected in the emotional sphere, causing increased reactivity to environmental factors, reducing the adaptive capacity of a person, leading to depression and nervous breakdowns. Particularly noteworthy is the connection inactivity with disease such as obesity. When sedentary person requires a minimum number of calories to sustain the body. Dietary habits of modern man is most commonly associated with a significant excess of this minimum. Slow energy metabolism in combination with an excess intake of calories and leads ultimately to obesity. Speaking about overweight, you should not forget about the many diseases that are a frequent consequence of obesity. Such diseases include doctors atherosclerosis, diabetes and a number of other serious zabolevaniy.Nauchnye studies have shown convincingly that one of the serious consequences of physical inactivity is to reduce the overall resilience of the human body. Prolonged lack of exercise contributes to the development of infectious diseases, making it difficult for them and slowing the recovery processes in the body. It seems that the above is enough to realize the danger of inactivity for human health. Special attention is certainly deserves inactivity influence the course of pregnancy, when the limitation of physical activity, many women seem a prerequisite for a healthy baby. Wellness and preventive effect of mass physical culture is inextricably linked with increased physical activity, increased function of the musculoskeletal system, metabolic activation. As a result of lack of motor

activity in the human body violated neuro- reflex connections, inherent nature and enshrined in the hard physical labor that leads to frustration regulation of cardiovascular and other systems, metabolic disorders and the development of degenerative diseases (atherosclerosis, etc.).

For the normal functioning of the human body and health protection requires a certain "dose" of physical activity. This raises the question of the so -called habitual motor activity, ie, the activities carried out in the daily professional work and at home. The most adequate expression of the amount of work produced by the muscles is the amount of energy consumption. Minimum value of daily energy required for the normal functioning of the body is 12 - 16 MJ (c. depending on age, sex and body weight), which corresponds to 2880 - 3840 kcal. Motor activity is among the main factors that determine the level of metabolic processes of the body and the condition of his bone, muscle and cardiovascular system. It is closely linked with the three aspects of health : physical, mental, and social and during the life of a man playing a different role. Body's need for motor activity is individual and depends on many physiological, their socio-economic cultural factors. The level of motor activity needs largely caused by hereditary and genetic traits. For normal development and functioning of the body maintaining health requires a certain level of nat. activity. This range has a minimum, the optimal levels of motor activity and a maximum. Minimum level allows you to maintain a normal functional state. At the optimum achieved the highest level of functionality and life of the organism ; maximum limits separated excessive exercise, which can lead to fatigue, a sharp decrease in efficiency. This raises the question of habitual physical activity, which can determine the level and nature of energy consumption in the normal course of life. Rate This motor activity is based on two components, professional and unprofessional. There are several methods to quantify motor activity :

1) at the timing of work performed per day ;

2) in terms of energy consumption on the basis of indirect kallorimetrii ;

3) by calculating the energy balance. Since heart rate quite accurately reflects the degree of stress on the cardiovascular system during muscle activity and is directly dependent on the oxygen consumption. Therefore, the value of heart rate during muscular work can serve as a quantitative indicator, physical activity, checked during the various tests. In economically developed countries in the last 100 years, the proportion of muscle work as a generator of energy for human use fell by almost a factor of 200, resulting in lower energy consumption for muscle activity (work sharing) to an average of 3.5 MJ.

In this regard, to compensate for the lack of energy in the workplace modern man must do exercises with the consumption of at least 350 - 500 kcal per day (or 2000 - 3000 kcal per week). Marked limitation of motor activity in recent decades has led to a decrease in functional capacity of middle-aged, so most of the modern population of economically developed countries there is a real danger of the development of hypokinesia.

Moreover, it consumes a certain amount of nutrients and a specific wear rate (as well as the whole organism). The man did not trained heart makes per minute more cuts, also consume more nutrients and certainly faster aging. Everything else in well-trained people. The number of beats per minute can be 50, 40 or less. Efficiency of the heart muscle is much higher than usual. Therefore such a heart wears much slower.

Exercise give rise to a very interesting and useful effect in the body. During load metabolism is greatly accelerated, but thereafter - begins to slow down, and finally reduced to a level below normal. In general, in exercising human metabolism slower than normal, the body works more economically, and life expectancy increases. Daily load trained body have markedly less devastating effect that prolongs life. Improving the system of enzymes, normal metabolism, a person sleeps better and recovering from sleep, which is very important. In the event of inactivity (lack of exercise), as well as with age appear negative changes in the respiratory organs. Reduced the amplitude of the respiratory movements.

Especially reduced ability to deep exhalation. In this regard, increasing the volume of residual air that adversely affects gas exchange in the lungs. Vital capacity is also reduced. All this leads to hypoxia. In a trained body, on the contrary, the amount of oxygen is higher (though reduced demand), and this is very important as oxygen deficiency generates a huge number of metabolic disorders. Significantly strengthens the immune system. In special studies conducted on humans shows that exercise increases the immunobiological properties of blood and skin, as well as resistance to some infectious diseases. Besides the above, an improvement of a variety of factors : the speed of movements may increase by 1.5 - 2 times, endurance - several times, the strength of 1.5 - 3 times the minute volume of blood during a 2 - 3 times the oxygen uptake 1 minute during operation - 1.5 - 2 -fold, etc.

Importance of exercise is that they increase the body's resistance to the action of a number of various unfavorable factors. For example, such as the low atmospheric pressure, overheating, some poisons, radiation, etc. In special experiments on animals have shown that rats that every 1 - 2 hours coached swimming, jogging or hovering on a thin pole, after X-ray irradiation survived a high percentage of cases. Upon repeated irradiation with low doses 15% of untrained rats perished after a total dose of 600 X-rays, and the same percentage of trained - after a dose of X-rays in 2400. Exercise increases the body's resistance to them after transplantation mice cancers.

Stress have on the body strong destructive action. Positive emotions on the contrary contribute to the normalization of many functions. Exercise promote conservation of energy and joie de vivre. Physical activity has a strong anti-stress effect.

From poor lifestyle or simply over time can accumulate in the body of harmful substances, so-called slag. Acidic environment, which is formed in the body during physical activity substantially oxidizes toxins into harmless compounds, and then they are easily found.

As you can see, the beneficial effects of exercise on the human body is truly endless. A man was originally designed by nature to increased motor activity. Reduced activity leads to many disorders and premature fading body.

Seemingly well- organized physical. exercise should bring us particularly impressive results. However, for some reason we do not notice that the athletes lived much longer than ordinary people.

Need to move - one of the general biological needs of the body, which plays an important role in shaping his life and at all stages of its evolution. Development is inseparable from the active muscular activity.

3.2. Microclimate

Mikroklimattika (greč. μικρός (Mikros) + κλίμα (klimatos)) - features of climate on small spaces, measured kilometers or in tens kilometers and caused by features regions (forest, field, meadow, marsh, beach, water, slope direction, securty by winds and t. p.).

Study of the microclimate has GREAT practical value, especially at zoning agricultural crops, sanatoriums organizations, community recreation.

This set of meteorological conditions in the room : temperature, relative humidity, the amount of ions *, air, air velocity, concentrations of particulate matter (dust), the presence of pleasant odors (aromatherapy), etc. The temperature difference measured horizontally from the windows to the opposite walls should not exceed 2 ° C, and the vertical 1 ° C per meter height spaces. For create a comfortable indoor climate using special system : during the cold season - heating : gas, wood or coal stoves ; centralized water system, in rare cases (as obsolete) steam heating (for residential and public buildings is prohibited) ; heaters and electric heaters. System ventilation (possibly using heat recovery exhaust air). Humidification systems (steam, ultrasonic, with traditional evaporation (cold)).

Local microclimate atmospheric zone where the climate differs from the surrounding area. The term may refer to areas as small as a few square feet (eg garden bed) or as large as many square miles. There microclimate, for example, near bodies of water that can cool the local atmosphere, or in heavily urban areas where brick, concrete, and asphalt absorb the sun's energy, heat up, and re-emit that heat to the surrounding air : as a result of the urban heat island is a kind of microclimate. Another reason is climate slope aspect or area. The southern slopes of the northern hemisphere and the northern slopes of the southern hemisphere are at direct sunlight than opposite slopes and therefore warmer longer.

An area in developed industrial zone can vary greatly from a wooded park nearby, as natural flora in parks absorb light and heat in the leaves that the roof of the building or parking radiates back into the air. Proponents of solar energy argue that widespread use of solar collection can mitigate overheating of urban environments, absorbing sunlight and put it to work instead of heating the outer surface of objects.

Microclimate can offer an opportunity as a small growing region for crops that can not thrive in the wider region; this concept is often used in permaculture practiced in northern temperate regions. Microclimate can be used in favor of gardeners who carefully choose and position their plants. Cities often raise the average temperature zoning and protected location can reduce the severity of winter. Roof gardening, however, exposes plants more extreme temperatures in both summer and winter.

Tall buildings to create its own microclimate, both by overshadowing large areas and directing strong winds down to ground level. The effects of wind around tall buildings are assessed as part of the research climate.

CONCLUSION

In the process of researching and writing my graduation work, I got to know a lot of useful information iOS mobile operating system, Objective C, develop game. I have accomplished the requirements and objectives in the graduate work. I have also become familiar with the recent trends and updates regarding the applications of this programming tool.

I created three games while doing task. The first game name is “Sonni top” (“Find number”). By this game kids can learn recon up.

The second game name is “Multiplication”. By this game kids can learn multiplication table.

The third game name is “EKUB o`yini” (“GCF game”). By this game kids can learn greatest common factor. So games make on uzbek language.

In future I can develop my games. Example I create for multi language.
So I learn from “Life safety” these themes: “Hypodynamy”, “Microlimate”.

LIST OF RESOURCES

1. The lecture the President of the Republic of Uzbekistan № 1730 of March 21, 2012.
2. Youth & Information and Communication Technologies (ICT). World Youth Report, 2013.
3. Object-Oriented Programming and the Objective-C Language. 2010.
4. From C++ to Objective C. Aaron Vegh. 2011.
5. iPhone Open Application Development. Jonathan A.Zdziarski. 2010.
6. Become an Xcoder. Bert Altenberg, Alex Clarke, Philippe Mougin.2013.
7. Learn iPhone and iPad cocos2D Game Development. Steffen Itterheim.2010.
8. Core Animation Simplified AnimationTechniques for Mac and iPhone Development. Marcus Zarra, Matt Long. 2011.
9. http://www.apple.com
10. http://www.wikipedia.org.
11. http://www.lex.uz
12. Decrees of the President Republic of Uzbekistan about IT: http://www.software.uz
13. http://www.njf.nu.
14. stackoverflow.com
15. github.com

APPENDIX

```
//
//  MLGViewController.m
//  MathLogicGames
//
//  Created by Shohin on 9/30/13.
//  Copyright (c) 2013 Shohin. All rights reserved.
//
#import "MLGViewController.h"
#import <QuartzCore/QuartzCore.h>
#import "GlobalConstants.h"
#import "MultiplicationGameViewController.h"
#import "FindNumbersViewController.h"
#import "EKUBGAMEViewController.h"
#import "AboutViewController.h"

#define NUMBERMENU 4

@interface MLGViewController (){
    NSMutableArray *_labels;
    NSArray *_gamenameArr;
}@end

@implementation MLGViewController

- (void)viewDidLoad
{
    //NSLog(@"%@", NSHomeDirectory());
    [super viewDidLoad];
    self.view.backgroundColor = [UIColor colorWithPatternImage:[UIImage
imageNamed:GCFGAMEBG_IMAGE]];
    _labels = [[NSMutableArray alloc] initWithCapacity:0];
    _gamenameArr = [[NSArray alloc] initWithArray:@[@"Sonni top",
@"Ko`paytirish", @"EKUB O`yini", @"Haqida"]];
    [self createList];
}

- (void) createList
{
    NSString *gameName = [NSString string];

    float offsetX = MAXLANDSCAPEWIDTH / 30;
```

```
    float offsetY = MAXLANDSCAPEHEIGHT / 20;

    float width = 12 * offsetX;

    float height = 8.5 * offsetY;

    float x = 2 * offsetX;
    float y = offsetY;

    int maxJ = 2;
    int tag = 1;

    for (int i = 0; i < 2; i++) {
        for (int j = 0; j < maxJ; j++) {

            CGRect _pageBtnFrame = CGRectMake(x, y, width, height);
            UIButton *_pageButton = [[UIButton alloc]
initWithFrame:_pageBtnFrame];
            //[_pageButton setBackgroundColor:[UIColor blueColor]];
            _pageButton.layer.borderColor = [UIColor blackColor].CGColor;
            _pageButton.layer.borderWidth = 1;
            _pageButton.layer.cornerRadius = 10;
            _pageButton.clipsToBounds = YES;
            _pageButton.showsTouchWhenHighlighted = YES;
            _pageButton.tag = tag;
            [_pageButton addTarget:self action:@selector(clickedPageButton:)
forControlEvents:UIControlEventTouchUpInside];

            [_pageButton setBackgroundImage:[UIImage
imageNamed:ICON_GAMES_iPHONE[tag - 1]] forState:UIControlStateNormal];

            CGRect _gameNameFrame = CGRectMake(0, 3 *
_pageButton.bounds.size.height / 4, _pageButton.bounds.size.width,
_pageButton.bounds.size.height / 4);
            UIView *_gameNameView = [[UIView alloc]
initWithFrame:_gameNameFrame];
            _gameNameView.backgroundColor = [UIColor blackColor];
            _gameNameView.alpha = 0.7;
            UILabel *_gameNameLabel = [[UILabel alloc]
initWithFrame:_gameNameFrame];
            _gameNameLabel.backgroundColor = [UIColor clearColor];
            _gameNameLabel.textColor = [UIColor whiteColor];
            _gameNameLabel.textAlignment = NSTextAlignmentCenter;
            gameName = [_gamenameArr objectAtIndex:tag - 1];
```

```
            _gameNameLabel.text  = gameName;
            _gameNameLabel.font  = [UIFont fontWithName:FONTNAME
size:sizeTextMenu];

            [_labels addObject:_gameNameLabel];
            [_gameNameLabel release];

            [_pageButton addSubview:_gameNameView];
            [_pageButton addSubview:_gameNameLabel];
            [self.view addSubview:_pageButton];

            [_gameNameLabel release];
            [_gameNameView release];
            [_pageButton release];

            tag++;
            x += width + x;
        }
        x = 2 * offsetX;
        y += height + offsetY;
    }

}

- (void)clickedPageButton:(UIButton *)sender
{
    switch (sender.tag) {
        case 1: {
            FindNumbersViewController *findNumberGameController =
[[FindNumbersViewController alloc] init];
            [self.navigationController
pushViewController:findNumberGameController animated:YES];
            [findNumberGameController release];
        }
            break;
        case 2: {
            MultiplicationGameViewController *multipGameController =
[[MultiplicationGameViewController alloc] init];
            [self.navigationController pushViewController:multipGameController
animated:YES];
            [multipGameController release];
        }
            break;
        case 3: {
```

```
            EKUBGAMEViewController *ekubGameController =
[[EKUBGAMEViewController alloc] init];
            [self.navigationController pushViewController:ekubGameController
animated:YES];
            [ekubGameController release];
        }
            break;
        case 4: {
            AboutViewController *about = [[AboutViewController alloc] init];
            [self.navigationController pushViewController:about animated:NO];
            [about release];
        }
            break;
        case 5: {
        }
            break;
        case 6: {

        }
            break;
        default:
            break;
    }
}

@end

//
//  FindNumbersViewController.m
//  UZLogicGames
//
//  Created by Shohin on 8/18/13.
//  Copyright (c) 2013 Shohin. All rights reserved.
//

#import "FindNumbersViewController.h"
#import "NumberButton.h"
#import "GlobalConstants.h"
#import "HUDView.h"
#import "FindNumGameMenu.h"
#import "LevelView.h"
#import "ResultView.h"
#import "ExplodeView.h"
#import "InformationView.h"
```

```
@interface FindNumbersViewController ()
{
    FindNumGameMenu *_menuView;
    LevelView *_levelView;
    float _widthHudView;
    float _widthGameBoard;
    UIView *_resultSuperView;
    ResultView *_resultView;
    InformationView *_infoTextView;
}
@end

@implementation FindNumbersViewController

@synthesize hudView = _hudView;

#pragma mark - life

- (void)viewDidLoad
{
    [super viewDidLoad];
    self.view.backgroundColor = [UIColor colorWithPatternImage:[UIImage
imageNamed:FINDGAME_BG_IMAGE]];

    _level = [[Level alloc] init];
    _numManager = [[NumberManager alloc] init];
    _sortNumbers = [[NSMutableArray alloc] init];

    _dataDict = [[NSMutableDictionary alloc] init];
    _levelsArr = [[NSMutableArray alloc] init];

    _menuView = [[FindNumGameMenu alloc] initWithImage:nil
andWithFrame:CGRectMake(0, 0, MAXLANDSCAPEWIDTH,
MAXLANDSCAPEHEIGHT)];
    _menuView.delegate = self;
    [self.view addSubview:_menuView];

    _levelView = [[LevelView alloc] initWithImage:nil
andWithFrame:CGRectMake(0, -MAXLANDSCAPEHEIGHT,
MAXLANDSCAPEWIDTH, MAXLANDSCAPEHEIGHT)];
    _levelView.delegate = self;

    _widthHudView = 80;
```

```
    _hudView = [[HUDView alloc] initWithFrame:CGRectMake(0, -
MAXLANDSCAPEHEIGHT, _widthHudView, MAXLANDSCAPEHEIGHT)];
    [_hudView createSubViews];
    [_hudView.menuButton addTarget:self action:@selector(backToMenu)
forControlEvents:UIControlEventTouchUpInside];

    _widthGameBoard = (MAXLANDSCAPEWIDTH -
_hudView.frame.size.width);
    _gameBoardImageView = [[UIImageView alloc]
initWithFrame:CGRectMake(_hudView.frame.size.width, -
MAXLANDSCAPEHEIGHT, _widthGameBoard,
MAXLANDSCAPEHEIGHT)];
    _gameBoardImageView.backgroundColor = [UIColor clearColor];
    _gameBoardImageView.userInteractionEnabled = YES;

    _resultSuperView = [[UIView alloc] initWithFrame:CGRectMake(0, 0,
MAXLANDSCAPEWIDTH, MAXLANDSCAPEHEIGHT)];
    _resultSuperView.backgroundColor = [UIColor blackColor];
    _resultSuperView.alpha = 0.8;
    float x = MAXLANDSCAPEWIDTH / 8;
    _resultView = [[ResultView alloc] initWithImage:[UIImage
imageNamed:FINDGAME_BG_IMAGE] andWithFrame:CGRectMake(x, x,
MAXLANDSCAPEWIDTH - 2 * x, MAXLANDSCAPEHEIGHT - 2 * x)];
    //[_resultView.backButton setFontImageButton:[UIFont
fontWithName:FONTNAME size:20]];
    [_resultView.backButton addTarget:self action:@selector(backToLevel)
forControlEvents:UIControlEventTouchUpInside];

    //[_resultView.againButton setFontImageButton:[UIFont
fontWithName:FONTNAME size:20]];
    [_resultView.againButton addTarget:self action:@selector(againGame)
forControlEvents:UIControlEventTouchUpInside];

    //[_resultView.nextLevelButton setFontImageButton:[UIFont
fontWithName:FONTNAME size:20]];
    [_resultView.nextLevelButton addTarget:self action:@selector(nextLevel)
forControlEvents:UIControlEventTouchUpInside];

    _infoTextView = [[InformationView alloc] initWithImage:@""
andWithFrame:CGRectMake(0, -MAXLANDSCAPEHEIGHT,
MAXLANDSCAPEWIDTH, MAXLANDSCAPEHEIGHT)];
    _infoTextView.informationTextView.text =@"\n\nBu o`yin asosan sanashni
o`rganish uchun mo`ljallangan. O`yinni o`ynash qoidasi juda oson. Buning uchun
berilgan sonni hosil bo`lgan sonlar orasidan topib, ustiga bosish kerak. Bunda har
```

bir topilgan son uchun ball qo`shiladi. Agar son xato topilsa, to`plangan ball kamayadi. Agar siz hamma berilgan sonlarni to`g`ri topsangiz, o`yinda yutgan bo`lasiz va sizda keyingi bosqichga o`tish imkoniyati paydo bo`ladi. Agar sizning vaqtingiz tugab qolsa yoki balingiz nolga teng bo`lib qolsa, siz o`yinda yutqazasiz.";

```
    [_infoTextView.backButton addTarget:self action:@selector(hideInfoView)
forControlEvents:UIControlEventTouchUpInside];
    [_infoTextView.infoButton removeFromSuperview];
}

#pragma mark - updateUI

- (void)showMenu
{
    [UIView beginAnimations:@"ShowMenu" context:nil];
    [UIView setAnimationCurve:UIViewAnimationCurveLinear];
    [UIView setAnimationDuration:0.3f];
    _menuView.center = self.view.center;
    [UIView commitAnimations];
    [self.view addSubview:_menuView];
}
- (void)hideMenu
{
    _menuView.frame = CGRectMake(-MAXLANDSCAPEWIDTH, 0,
MAXLANDSCAPEWIDTH, MAXLANDSCAPEHEIGHT);
    [_menuView removeFromSuperview];
}
- (void)showLevel
{
    [_dataDict setDictionary:[SAVEGAMEDATA
dictionaryForKey:FINDNUMBERGAME_KEY]];
    [_levelsArr setArray:[_dataDict objectForKey:LEVELS_KEY]];

    [UIView beginAnimations:@"ShowLevel" context:nil];
    [UIView setAnimationCurve:UIViewAnimationCurveLinear];
    [UIView setAnimationDuration:0.3f];
    _levelView.center = self.view.center;
    [_levelView creteSubViews];
    [UIView commitAnimations];

    if (_levelsArr.count > 0) {
        for (ImageButton *levelBtn in _levelView.subviews) {
            if ([levelBtn isKindOfClass:[ImageButton class]]) {
```

```
            if (levelBtn.tag > 1 && levelBtn.tag < _level.maxLevelNum + 1) {
                levelBtn.userInteractionEnabled = [[_levelsArr
objectAtIndex:(levelBtn.tag - 1)] boolValue];
                if (levelBtn.userInteractionEnabled) {
                    [levelBtn setImage:LEVEL_BUTTON_OPEN_IMAGE];
                    [levelBtn setHiddenTitle:NO];
                }
            }
        }
    }
    } else {
        NSMutableArray *arr = [NSMutableArray array];
        for (int i = 0; i < _level.maxLevelNum; i++) {
            [arr addObject: i > 0 ? @"NO" : @"YES"];
        }
        [_dataDict setObject:arr forKey:LEVELS_KEY];
        [SAVEGAMEDATA setObject:_dataDict
forKey:FINDNUMBERGAME_KEY];
    }

    [self.view addSubview:_levelView];
}

- (void)hideLevel
{
    _levelView.frame = CGRectMake(0, -MAXLANDSCAPEHEIGHT,
MAXLANDSCAPEWIDTH, MAXLANDSCAPEHEIGHT);
    [_levelView removeSubViews];
    [_levelView removeFromSuperview];
}

- (void)showHudView
{
    [_hudView.pointLabel setPoints:_level.pointLevel];
    [_hudView.timerLabel setSeconds:_level.timeLevel];
    _hudView.levelLabel.text = [NSString stringWithFormat:@"Bosqich %d",
_level.levelNum];
    [self updateNumberLabel];
    [UIView beginAnimations:@"ShowHudView" context:nil];
    [UIView setAnimationCurve:UIViewAnimationCurveLinear];
    [UIView setAnimationDuration:0.3f];
    [UIView setAnimationDuration:1];
    _hudView.frame = CGRectMake(0, 0, _widthHudView,
MAXLANDSCAPEHEIGHT);
```

```
    [UIView commitAnimations];
    [self.view addSubview:_hudView];
}

- (void)hideHudView
{
    _hudView.frame = CGRectMake(0, -MAXLANDSCAPEHEIGHT,
_widthHudView, MAXLANDSCAPEHEIGHT);
    [_hudView removeFromSuperview];
}

- (void)showGameBoard
{
    [self createNumberButtons:_level.rowNum withMaxJ:_level.colsNum];
    _gameBoardImageView.frame = CGRectMake(_hudView.frame.size.width, 0,
_widthGameBoard, MAXLANDSCAPEHEIGHT);
    [self.view addSubview:_gameBoardImageView];
    [self startTimer];
}

- (void)hideGameBoard
{
    _gameBoardImageView.frame = CGRectMake(_hudView.frame.size.width, -
MAXLANDSCAPEHEIGHT, _widthGameBoard, MAXLANDSCAPEHEIGHT);
    [self stopTimer];
    [self clearViews:_gameBoardImageView.subviews];
    [_sortNumbers removeAllObjects];
    [_gameBoardImageView removeFromSuperview];
}

- (void)showResultView
{
    _resultView.timeLevel.text = [NSString stringWithFormat:@"Vaqt %@",
_hudView.timerLabel.text];
    _resultView.pointLevel.text = [NSString stringWithFormat:@"Ball %@",
_hudView.pointLabel.text];

    _resultView.timeLevel.font = [UIFont fontWithName:FONTNAME size:2 *
_resultView.timeLevel.bounds.size.width / _resultView.timeLevel.text.length];
    _resultView.pointLevel.font = [UIFont fontWithName:FONTNAME size:1.5 *
_resultView.pointLevel.bounds.size.width / _resultView.pointLevel.text.length];

    [self.view addSubview:_resultSuperView];
    [self.view addSubview:_resultView];
```

```
}

- (void)hideResultView
{
    [_resultView removeFromSuperview];
    [_resultSuperView removeFromSuperview];
}

- (void)showInfoView
{
    [UIView beginAnimations:@"ShowInfoView" context:nil];
    [UIView setAnimationCurve:UIViewAnimationCurveLinear];
    [UIView setAnimationDuration:0.3f];
    [UIView setAnimationDuration:0.3];
    _infoTextView.frame = self.view.bounds;
    [UIView commitAnimations];
    [self.view addSubview:_infoTextView];
}

- (void)hideInfoView
{
    [_infoTextView removeFromSuperview];
    [self showMenu];
}


- (void)updateNumberLabel
{
    _hudView.numberLabel.text = [_sortNumbers objectAtIndex:0];
}

- (void)createNumberButtons:(int)maxI withMaxJ:(int)maxJ
{
    int n = maxI * maxJ;
    int index = 0;
    NSMutableArray *_differentNumbers = [[NSMutableArray alloc]
initWithArray:[_numManager
randomizeDifferentNumberWithoutZeroToNumber:n andNumbers:n]];

    [_sortNumbers addObjectsFromArray:[_numManager
sortNumbers:_differentNumbers]];

    CGSize gameBoardSize = _gameBoardImageView.bounds.size;
```

```
    float lengthHeight  = ceilf(gameBoardSize.height*0.9  / (float) maxI) -
NUMBERBUTTONMARGIN;
    float lengthWidth = ceilf(gameBoardSize.width*0.9  / (float) maxJ) -
NUMBERBUTTONMARGIN;

    float yOffset = (gameBoardSize.height  - maxI * (lengthHeight  +
NUMBERBUTTONMARGIN))/2;
    float xOffset = (gameBoardSize.width  - maxJ * (lengthWidth  +
NUMBERBUTTONMARGIN))/2;

    xOffset += lengthWidth  / 2;
    yOffset += lengthHeight  / 2;

    for (int i = 0; i < maxI; i++) {
        for (int j = 0; j < maxJ; j++) {
            NumberButton *numButton = [[NumberButton alloc]
initWithNumber:[[_differentNumbers  objectAtIndex:index]  intValue]
withSideLength:lengthWidth];
            numButton.center = CGPointMake(xOffset + j*(lengthWidth  +
NUMBERBUTTONMARGIN), -(yOffset + i*(lengthHeight  +
NUMBERBUTTONMARGIN)));
            numButton.tag = 0;
            [numButton addTarget:self action:@selector(numberButtonClicked:)
forControlEvents:UIControlEventTouchUpInside];

            [UIView beginAnimations:@""  context:nil];
            [UIView setAnimationCurve:UIViewAnimationCurveLinear];
            [UIView setAnimationDuration:0.04  * (i + 1)];
            [UIView setAnimationDelay:0.04  * (j + 1)];
            numButton.center = CGPointMake(xOffset + j*(lengthWidth  +
NUMBERBUTTONMARGIN), yOffset + i*(lengthHeight  +
NUMBERBUTTONMARGIN));
            [UIView commitAnimations];

            [_gameBoardImageView  addSubview:numButton];
            [numButton release];
            index++;
        }
    }
}

- (void) numberButtonClicked:(NumberButton  *)sender
{
    NSString *firstNumber = nil;
```

```
    NSString *secondNumber = [_sortNumbers objectAtIndex:0];
    for (UILabel *labelButton in [sender subviews]) {
        if ([labelButton isKindOfClass:[UILabel class]]) {
            firstNumber = labelButton.text;
            break;
        }
    }

    if ([firstNumber isEqualToString:secondNumber]) {
        UIView *view = [[UIView alloc]
initWithFrame:_gameBoardImageView.bounds];
        view.tag = 100;
        if (isSoundFindNumGame) {
            [AUDIO playSound:RIGHT_SOUND];
        }
        _gameBoardImageView.userInteractionEnabled = NO;
        [sender setBackgroundImage:[UIImage
imageNamed:CHOOSE_NUMBER_BUTTON_IMAGE]
forState:UIControlStateNormal];
        [UIView beginAnimations:@"RemoveButton" context:nil];
        [UIView setAnimationCurve:UIViewAnimationCurveLinear];
        //[UIView setAnimationDelay:1];
        [UIView setAnimationDuration:0.5];
        [sender setFrame:view.bounds];
        [sender changePositionTitle:sender.center];
        [view addSubview:sender];
        [_gameBoardImageView addSubview:view];
        [UIView commitAnimations];
        [self performSelector:@selector(explode:) withObject:sender afterDelay:0.4];
        [self performSelector:@selector(removeSenderObject:) withObject:sender
afterDelay:0.5];
        [_sortNumbers removeObjectAtIndex:0];
        if (_sortNumbers.count > 0) {
            [self updateNumberLabel];
        }
        [self performSelector:@selector(setGameBoardUserInteraction:)
withObject:@"YES" afterDelay:0.5];
        [_hudView.pointLabel zoominPoints:5];
    } else {
        if (isSoundFindNumGame) {
            [AUDIO playSound:ERROR_SOUND];
        }

        [_hudView.pointLabel zoomoutPoints];
```

```
    }
}

- (void)setGameBoardUserInteraction:(NSString  *)userInteraction
{
    if ([userInteraction isEqualToString:@"YES"] || [userInteraction
isEqualToString:@"NO"]) {
        _gameBoardImageView.userInteractionEnabled  = [userInteraction
boolValue];
    }
}

- (void)setHudViewUserInteraction:(NSString  *)userInteraction
{
    if ([userInteraction isEqualToString:@"YES"] || [userInteraction
isEqualToString:@"NO"]) {
        _hudView.userInteractionEnabled  = [userInteraction boolValue];
    }
}

#pragma mark - control methods

- (void)backToMenu
{
    [self hideHudView];
    [self hideGameBoard];
    [self showLevel];
}

- (void)backToLevel
{
    [self hideGameBoard];
    [self hideHudView];
    [self showLevel];
    [self hideResultView];
}

- (void)againGame
{
    [self hideGameBoard];
    [self hideHudView];
    [self showGameBoard];
    [self showHudView];
    [self hideResultView];
```

```
}

- (void)nextLevel
{
    if (_level.levelNum  < _level.maxLevelNum)  {
        _level.levelNum++;
    } else {
        _level.levelNum  = 1;
    }
    [self hideGameBoard];
    [self hideHudView];
    [self showGameBoard];
    [self showHudView];
    [self hideResultView];
}

#pragma mark - timerSection

-(void)startTimer
{
    _seconds = _level.timeLevel;
    _timer = [NSTimer scheduledTimerWithTimeInterval:1.0
                                              target:self
                                            selector:@selector(timerFunc:)
                                            userInfo:nil
                                             repeats:YES];
}

- (void)stopTimer
{
    [_timer invalidate];
    _timer = nil;
}

- (void)timerFunc:(NSTimer  *)timer
{
    _seconds--;
    [_hudView.timerLabel  setSeconds:_seconds];
    if (_seconds==0) {
        [self stopTimer];
    }
    [self win];
    [self lost];
}
```

```
#pragma mark - clearSection

- (void)removeSenderObject:(UIButton  *)sender
{
    [sender removeFromSuperview];
    [[_gameBoardImageView  viewWithTag:100]  removeFromSuperview];
}

- (void)clearViews:(NSArray  *)subViews
{
    for (UIView  *object in subViews) {
        [object removeFromSuperview];
    }
}

#pragma mark - animationSection

#pragma mark - effects

- (void)explode:(UIView  *)sender
{
    ExplodeView *explode = [[ExplodeView  alloc] initWithFrame:sender.bounds];
    [sender addSubview:explode];
    [sender sendSubviewToBack:explode];
}

#pragma mark - gameStateSection

- (void)win
{
    if (_sortNumbers.count == 0) {
        NSLog(@"Win");

        [self stopTimer];

        _hudView.userInteractionEnabled  = NO;
        [_levelsArr  replaceObjectAtIndex:_level.levelNum  withObject:@"YES"];
        [_dataDict setObject:_levelsArr  forKey:LEVELS_KEY];

        [SAVEGAMEDATA  setObject:_dataDict
forKey:FINDNUMBERGAME_KEY];
        [SAVEGAMEDATA  synchronize];
        if (isSoundFindNumGame) {
```

```
            [AUDIO playSound:WIN_SOUND];
        }

        [self performSelector:@selector(explode:)
withObject:_gameBoardImageView afterDelay:0.3];

        _resultView.stateLabel.text = @"Yutdingiz";
        _resultView.stateLabel.font = [UIFont fontWithName:FONTNAME
size:_resultView.stateLabel.bounds.size.width /
(_resultView.stateLabel.text.length)];
        _resultView.nextLevelButton.userInteractionEnabled = YES;

        [self performSelector:@selector(showResultView) withObject:nil
afterDelay:1.3];
        [self performSelector:@selector(setHudViewUserInteraction:)
withObject:@"YES" afterDelay:1.4];
    }
}
- (void)lost
{
    if (_seconds == 0 || [_hudView.pointLabel.text isEqualToString:@"0"]) {
        NSLog(@"Lost");

        [self stopTimer];
        if (isSoundFindNumGame) {
            [AUDIO playSound:LOST_SOUND];
        }

        _resultView.stateLabel.text = @"O`yin tugadi";
        _resultView.stateLabel.font = [UIFont fontWithName:FONTNAME
size:_resultView.stateLabel.bounds.size.width /
(_resultView.stateLabel.text.length)];
        _resultView.nextLevelButton.userInteractionEnabled = NO;

        [self showResultView];
    }
}

#pragma mark - delegate

- (void)pressMenuButton:(ImageButton *)sender
{
    //    [UIView beginAnimations:@"counterclockwiseAnimation"
    //                    context:NULL];
```

```
    //    /* 5 seconds long */
    //    [UIView setAnimationDuration:5.0f];
    //    /* Back to original rotation */
    //    _menu.transform =
    //    CGAffineTransformMakeRotation((90.0f * M_PI) / 180.0f);;
    //    [UIView commitAnimations];
    //_levelView.frame = CGRectMake(-MAXLANDSCAPEWIDTH, 0,
MAXLANDSCAPEWIDTH, MAXLANDSCAPEHEIGHT);
    if (isSoundFindNumGame) {
        [AUDIO playSound:CLICKED_SOUND];
    }
    switch (sender.tag) {
        case 1:
        {
            [self hideMenu];
            [self showLevel];
        }
            break;
        case 2:
        {
            [self hideMenu];
            [self showInfoView];
        }
            break;
        case 3:
        {
            [self.navigationController popViewControllerAnimated:NO];
        }
            break;
        case 4:
        {
            if (isSoundFindNumGame) {
                isSoundFindNumGame = NO;
                [sender setImage:SOUND_OFF_BUTTON_IMAGE];
            } else {
                isSoundFindNumGame = YES;
                [sender setImage:SOUND_ON_BUTTON_IMAGE];
            }
            [SAVEGAMEDATA setBool:isSoundFindNumGame
forKey:MULTIPGAME_SOUND_KEY];
            [SAVEGAMEDATA synchronize];
            NSLog(@"is %i", isSoundFindNumGame);
        }
            break;
```

```
        default:
            break;
    }
}

- (void)pressLevelButton:(UIButton *)sender
{
    if (isSoundFindNumGame) {
        [AUDIO playSound:CLICKED_SOUND];
    }

    [self hideLevel];
    if (sender.tag < _level.maxLevelNum + 1) {
        [_level setLevelNum:sender.tag];
        [self showGameBoard];
        [self showHudView];
    } else {
        [self showMenu];
    }
}

#pragma mark - dealloc

- (void)dealloc
{
    [_numManager release];
    [_sortNumbers release];

    [_level release];
    [_timer release];
    [_hudView release];
    [_menuView release];
    [_levelView release];
    [_gameBoardImageView release];
    [_resultView release];
    [_resultSuperView release];
    [_infoTextView release];
    [_dataDict release];
    [_levelsArr release];
    [super dealloc];
}

#pragma mark - rotate
```

```
- (BOOL)shouldAutorotate
{
    return YES;
}

- (NSUInteger)supportedInterfaceOrientations
{
    return UIInterfaceOrientationMaskLandscapeRight;
}

//-
(BOOL)shouldAutorotateToInterfaceOrientation:(UIInterfaceOrientation)interface
Orientation {
//    return interfaceOrientation == UIInterfaceOrientationMaskLandscapeRight;
//}

@end
```