Міністерство освіти і науки України
Вищий приватний навчальний заклад міжнародний економіко-гуманітарний університет імені академіка Степана Дем'янчука

С.В. Максимчук

РОЗРОБКА ГРИ ПІД ПЛАТФОРМУ iOS ДЛЯ iPHONE

Науковий керівник:
Р.М.Літнарович, доцент,к.т.н.
Рівне – 2012 р.

УДК 614.2 Максимчук С.В. Розробка гри під платформу iOS для iPhone. Науковий керівник Літнарович Р.М. . МЕГУ, Рівне, 2012.-155 с.

Рецензенти: В.Г.Бурачек, доктор технічних наук, професор
Є.С. Парняков, доктор технічних наук, професор
В.О.Боровий , доктор технічних наук, професор
Відповідальний за випуск: Й.В. Джунь, доктор фізико-математичних наук, професор

Сучасна комп'ютерна гра – це багатофункціональна програма, яку використовують не тільки з розважальними, а й із навчальними та пропагандистськими цілями. Гра заявлена в даній роботі має розвиваючий характер. Адже в ній гравець підвищує свою реакцію, за рахунок швидкої зміни цільових об'єктів. Також за рахунок цікавого дизайну, має більш привабливий сприйняття вигляд. Монографія містить повноцінну iPhone гру, та інформацію про засоби які необхідні розробнику для створення аналогічного продукту.

Ключові слова: мобільна гра, iPhone ігри, розробка гри, iOS .

Современная компьютерная игра - это многофункциональная программа, которую используют не только с развлекательными, но и с учебными и пропагандистскими целями. Игра заявлена в данной работе имеет развивающий характер. Ведь в ней игрок повышает свою реакцию, за счет быстрой смены целевых объектов. Также за счет интересного дизайна, имеет более привлекательный восприятия вид. Монография содержит полноценную iPhone игру, и информацию о средствах необходимых разработчику для создания аналогичного продукта.
Ключевые слова: мобильная игра, iPhone игры, разработка игры, iOS

Modern computer games - is a multipurpose program that is used not only entertaining, but also educational and propaganda purposes. The game is stated in this paper is developing character. Because in it the player raises his reaction by quickly changing targets. Also, due to interesting design has a more attractive form of perception. The monograph contains a full iPhone game, and information on the means necessary for developers to create a similar product.
Keywords: mobile game, iPhone Games, Game Development, iOS.

Сергій Валентинович Максимчук,
магістрант інформаційних технологій

Зміст

Вступ

Розвиток цивілізації у нашому столітті неодмінно йде у парі з розвитком різноманітних технологій, зокрема комп'ютерних. Комп'ютерний ринок постійно наповнюється новими, досконалішими програмами, збільшується швидкість процесорів, об'єм носіїв збереження пам'яті. У цій технологічній боротьбі не останнє місце займає явище, яке виникло разом з комп'ютерами, а саме, комп'ютерні ігри.

Перша комп’ютерна гра «Зоряні війни» вийшла у світ 1962 року. Її завдання полягало в тому, щоб відбити астероїди і напади ворожих космічних кораблів. Згодом було створено багато інших ігор. А з поширенням у 1970–1980 роках потужніших комп’ютерів електронних ігор побільшало: пригодницькі ігри, ігри-головоломки, стратегічні ігри та ігри «екшн». Багато ігор імітують різні види спорту, як от хокей на льоду чи гольф. Чимало з них здобули високу оцінку громадськості, оскільки вони дуже

цікаві й допомагають у навчанні. Однак ігри «екшн», як і ті, що їх називають «шутерами» (стрілялками), часто критикують через їхній агресивний характер. Зазвичай мета цих ігор — вибрати зброю і знищити всіх ворогів: людей та інших істот.

Стає дедалі популярнішим он-лайновий вид комп'ютерної ігри. Що особливого у ньому? Її персонажами керує не комп'ютер, а гравці, які через Інтернет одночасно беруть участь у грі. Їх можуть бути тисячі. Популярність таких забав пояснюється можливістю поспілкуватися з іншими. Гравці "розмовляють" одні з одними і відчувають себе частиною всесвітньої родини.

На початку 21 століття, в основному грали в ігри на DVD-CD-ROM. Це було реальним зміною стилю гри. Останні ігрові системи, такі як Sony Playstation і X-box від Microsoft підтримують можливість ігор онлайн, завдяки чому в них можуть грати люди з усього світу. Все більше зростання швидкісного Інтернету зробило можливість ігор онлайн реальністю

у повному розумінні цього слова.

На щастя для завзятих геймерів, зростає і індустрія роздрібної торгівлі онлайн іграми. Цей зріст - всього лише ще один елемент мінливої історії онлайн ігор.

Розділ 1. Ігри під iOS

1.1 Історія компанії Apple. iOS як операційна система

XX століття прийнято називати століттям авіації та космонавтики. Але 70-ті роки минулого століття також можна назвати епохою становлення сучасної IT-індустрії. І далеко не останню роль тут зіграла компанія Apple.

Історія компанії Apple почалася тридцять років тому, коли двоє друзів Стів Джобс і Стів Возняк вирішили заснувати власну компанію з виробництва і випуску комп'ютерів. Офіційно Apple була заснована 1 квітня 1976 року, і саме тоді на ринку з'явився зібраний вручну комп'ютер Apple Computer I - за десять місяців було зібрано і продано 175 штук за ціною 666,66 доларів. По суті справи, Apple I представляв собою

материнську плату без корпусу, клавіатури, звука та графіки.

У лютому 1977 року президентом Apple стає Майкл Скотт. Істотним кроком вперед став випуск в квітні цього ж року Apple Computer II - перший персональний комп'ютер з кольоровою графікою. Тут була підтримка виведення на екран різних кольорів, розроблені команди для роботи зі звуком і, отже, був вбудований динамік і клавіатура, з'явилася можливість оперативної зміни зображення на дисплеї, був блок живлення і т.д. Вся ця «начинка» була упакована в спеціально розроблений литий пластиковий корпус, що дуже вигідно відрізняло новий Apple від інших комп'ютерів, що мали вигляд незграбних ящиків з листового металу. Тоді ж з'явився відомий тепер всьому світу логотип - надкушене різнокольорове яблуко - створений рекламним агентством Regis McKenna.

У травні 1979 року співробітник Apple Джеф Раскін почав працювати над новим комп'ютером «все в

одному», який був орієнтований на рядового користувача. Саме цей період і можна назвати початком народження першого Macintosh.

У 1983 році Apple випустив Lisa - перший персональний комп'ютер з графічним призначеним для користувача інтерфейсом, названий так на честь дочки Джобса. Але через досить високу ціну і обмежений набір програм ця модель не отримала широкого розповсюдження. Хоча з комерційної точки зору Lisa була повним провалом, тим не менше, її випуск не пройшов даремно - використана тут операційна система Lisa 7/7 мала віконний інтерфейс, буфер, який дозволяв передавати дані між додатками і багато іншого.

Найперший свій Macintosh компанія Apple офіційно представила 22 січня 1984 рекламним роликом «1984», заснованим на сюжеті книги Д. Оруелла. Цей кліп був названий рекламним кліпом десятиліття, отримав гран-прі в Каннах і до цього дня вважається в історії реклами одним з

найоригінальніших. Ця подія стала значущою віхою в історії персональних комп'ютерів. З появою Macintosh докорінно змінилося уявлення людей про комп'ютери, і були закладені ті основи, які в подальшому стали використовувати всі комп'ютерні фірми. У вересні цього ж року Apple починає продавати комп'ютери Macintosh 512K за ціною 2495 доларів.

Йшли роки і, природно, що комп'ютерна галузь теж не стояла на місці.Протягом усього часу вдосконалення комп'ютерів Macintosh розвивалася і їх операційна система Mac OS (System), тобто розширювалися нові можливості, мінялися її окремі елементи і впроваджувалися нові технології. Коли Mac OS була вперше представлена в 1984 році, то для того часу вона являла собою досконалість сучасного програмування.

Система з самого початку розроблялася з розрахунком на те, щоб будь-який користувач, не маючи потрібної кваліфікації, вперше сівши за

комп'ютер, міг, відразу ж через кілька хвилин почати працювати на ньому, не вникаючи в технічні параметри. Операційні системи для інших платформ, які з'явилися кілька років потому, пропонували такі речі, як захищена пам'ять і багатозадачність, що в кінцевому підсумку давало кращий користувальницький досвід. Apple не вбудовувала цих речей в Mac OS тому, що мікропроцесори, які були доступні під час розробки Mac OS, не були достатньо потужними. Але часом Mac OS обростала додатковою функціональністю, і при цьому залишалася найбільш легкою у вивченні і використанні. Зручність і про ¬ стота роботи в цій системі були одними з головних цілей її творців, і внутрішня логіка взаємодії окремих частин операційної системи була в першу чергу підпорядкована цим вимогам.

Коли перед Apple постало питання про розробку операційної системи нового покоління, то, зваживши всі «за» і «проти» компанія як основу для нової Mac

OS вирішила скористатися розробками фірми NeXT, яка використовувала операційні системи під загальною назвою UNIX. Саме система OpenStep (NeXTStep) стала основою для побудови нової Mac OS. Розробникам нової Mac OS треба було забезпечити плавний перехід від старої системи до нової так, щоб користувачі могли як і раніше працювати з великою кількістю вже наявних програм, поки не будуть створені більш потужні та зручні. Так з'явилася нова система Mac OS X.

Операційна система — це базовий комплекс програмного забезпечення, що виконує управління апаратним забезпеченням комп'ютера або віртуальної машини; забезпечує керування обчислювальним процесом і організує взаємодію з користувачем.

Операційну систему можна розглядати за різними критеріями :

1) Операційна система як віртуальна машина

При розробці ОС широко застосовується абстрагування, що є важливим методом спрощення й дозволяє сконцентруватися на взаємодії високорівневих компонентів системи, ігноруючи деталі їхньої реалізації. У цьому змісті ОС являє собою інтерфейс між користувачем і комп'ютером.

Архітектура більшості комп'ютерів на рівні машинних команд дуже незручна для використання прикладними програмами. Наприклад, робота з диском припускає знання внутрішнього пристрою його електронного компонента – контролера для уведення команд обертання диска, пошуку й форматування доріжок, читання й записи секторів і т.д. Ясно, що середній програміст не в змозі враховувати всі особливості роботи встаткування (у сучасній термінології – займатися розробкою драйверів пристроїв), а повинен мати просту високорівневу

абстракцію, скажемо представляючи інформаційний простір диска як набір файлів. Файл можна відкривати для читання або запису, використати для одержання або скидання інформації, а потім закривати. Це концептуально простіше, ніж піклуватися про деталі переміщення головок дисків або організації роботи мотора. Аналогічним образом, за допомогою простих й ясних абстракцій, ховаються від програміста всі непотрібні подробиці організації переривань, роботи таймера, керування пам'яттю й т.д. Більше того, на сучасних обчислювальних комплексах можна створити ілюзію необмеженого розміру оперативної пам'яті й числа процесорів. Всім цим займається операційна система. Таким чином, операційна система представляється користувачеві віртуальною машиною, з якої простіше мати справу, чим безпосередньо з устаткуванням комп'ютера.

2) Операційна система як менеджер ресурсів

Операційна система призначена для керування всіма частинами досить складної архітектури комп'ютера. Представимо, приміром, що відбудеться, якщо кілька програм, що працюють на одному комп'ютері, будуть намагатися одночасно здійснювати вивід на принтер. Ми одержали б мішанину рядків і сторінок, виведених різними програмами. Операційна система запобігає такого роду хаос за рахунок буферізації інформації, призначеної для друку на диску й організації черги на друк. Для багатокористувацьких комп'ютерів необхідність керування ресурсами і їхнього захисту ще більш очевидна. Отже, операційна система, як менеджер ресурсів, здійснює впорядкований і контрольований розподіл процесорів, пам'яті й інших ресурсів між різними програмами.

3) Операційна система як захисник користувачів і програм

Якщо обчислювальна система допускає спільну роботу декількох користувачів, то виникає проблема організації їхньої безпечної діяльності. Необхідно забезпечити схоронність інформації на диску, щоб ніхто не міг видалити або зашкодити чужі файли. Не можна дозволити програмам одних користувачів довільно втручатися в роботу програм інших користувачів. Потрібно припиняти спроби несанкціонованого використання обчислювальної системи. Всю цю діяльність здійснює операційна система як організатор безпечної роботи користувачів й їхніх програм. З такого погляду операційна система представляється системою безпеки держави, на яку покладені поліцейські й контррозвідницькі функції.

4) Операційна система як постійно функціонуюче ядро

Нарешті, можна дати й таке визначення: операційна система - це програма, що постійно працює на комп'ютері й взаємодіюча з усіма прикладними програмами. Здавалося б, це абсолютно правильне визначення, але, як ми побачимо далі, у багатьох сучасних операційних системах постійно працює на комп'ютері лише частина операційної системи, що прийнято називати її ядром.

Як ми бачимо, існує багато точок зору на те, що таке операційна система. Неможливо дати їй адекватне строге визначення. Нам простіше сказати не що є операційна система, а для чого вона потрібна й що вона робить.

Операційні системи виконують велику кількість *функцій* :

Головні функції:

- Виконання на вимогу програм користувача тих елементарних (низькорівневих) дій, які є спільними для більшості програмного забезпечення і часто зустрічаються майже у всіх програмах (ввід і вивід даних, запуск і зупинка інших програм, виділення та вивільнення додаткової пам'яті тощо).
- Стандартизований доступ до периферійних пристроїв (пристрої введення-виведення).
- Завантаження програм у оперативну пам'ять і їх виконання.
- Керування оперативною пам'яттю (розподіл між процесами, організація віртуальної пам'яті).
- Керування доступом до даних енергозалежних носіїв (твердий диск, оптичні диски тощо), організованим у тій чи іншій файловій системі.
- Забезпечення користувацького інтерфейсу.
- Мережеві операції, підтримка стеку мережевих протоколів.

Додаткові функції:

- Паралельне або псевдопаралельні виконання задач (багатозадачність).
- Розподіл ресурсів обчислювальної системи між процесами.
- Організація надійних обчислень (неможливість впливу процесу на перебіг інших), основана на розмежуванні доступу до ресурсів.
- Взаємодія між процесами: обмін даними, синхронізація.
- Захист самої системи, а також користувацьких даних і програм від дій користувача або програм.
- Багатокористувацький режим роботи та розділення прав доступу (автентифікація, авторизація).

Окрема операційна система зазвичай може виконуватись на обмеженому переліку обладнання, яке забезпечує потрібні їй механізми. Сучасні універсальні (і не тільки) операційні системи зазвичай вимагають апаратної підтримки наступних механізмів:

- підтримка сторінкового поділу оперативної пам'яті з можливістю апаратного захисту сторінок від модифікації даних окремими задачами (процесами);
- підтримка захищеного режиму виконання процесора (режиму ядра ОС), який передбачає можливість виконання операцій процесора по управлінню обладнанням системи, при цьому спроба виконати подібну операцію в прикладній програмі блокується апаратно.

Можуть існувати і інші вимоги.

У сучасному світі існує велика кількість операційних систем. На мою думку, найбільш цікавими для дослідження є операційні системи компанії “Apple Inc.”, а саме iOS. Я вважаю, що саме за продуктами цієї компанії майбутнє і хочу розповісти більш детально щодо неї.

iOS (відома як **iPhone OS** до червня 2010 року) — це власницька мобільна операційна система від Apple. Розроблена спочатку для iPhone, вона стала операційною системою також для iPod Touch,iPad і Apple TV. Apple не ліцензує встановлення iOC на мобільних пристроях інших фірм.

iOS є похідною від Mac OS X, отже, є за своєю природою Unix-подібною операційною системою.

Користувацький інтерфейс iOS заснований на концепції прямої маніпуляції з використанням жестів Multi-Touch. Елементи інтерфейсу управління складаються з повзунків, перемикачів і кнопок. Він

призначений для безпосереднього контакту користувача з екраном пристрою. Внутрішній акселерометр використовуються деякими програмами для реагування на струшування пристрою, яке є також загальною командою скасування, або обертати пристрій у трьох вимірах, що є загальною командою перемикання між книжковим та альбомним режимами.

В якості операційної системи iOS була представлена з iPhone на Macworld Conference & Expo 9 січня 2007 року і випущена в червні того ж року. Зпершу, Apple не вказувала її ім'я, просто заявивши, що "iPhone використовує OS X ". Спочатку, сторонні програми не підтримувалися. Стів Джобс заявив, що розробники можуть створювати веб-програми, що «будуть вести себе, як рідні програми на iPhone». 17 жовтня 2007 року Apple оголосила, що рідний SDK знаходиться в стадії розробки, і що вони планують поставити його «в руки розробників у лютому». 6 березня 2008 року Apple випустила першу бета-версію,

а також нове ім'я для операційної системи: iPhone OS. Продажі мобільних пристроїв Apple запалили інтерес до SDK. Apple також продала більше одного мільйона iPhones під час курортного сезону 2007. У червні 2010 року, Apple перейменувала iPhone OS як iOS. Назва iOS користувалася компанією Cisco вже більше десяти років на маршрутизаторах Cisco. Для того щоб уникнути будь-якого потенційної позову, Apple ліцензувала «iOS» торгової марки від Cisco.

Головний екран

Домашній екран з іконками програм, і дока в нижній частині екрана, де користувачі можуть пов'язують найбільш часто використовувані програми, представляється щоразу, коли пристрій увімкнений або натискається кнопка Home. Екран має статус-бар у верхній частині екрану для відображення даних, таких як: час, рівень заряду батареї, сили сигналу, стан інтернет-з'єднання та блютузу тощо.

Папки

З iOS 4 була введенна проста Файлова система. Якщо іконки знаходяться в «режимі погойдування», будь-які дві можна перетягувати одну на одну, щоб створити папку, і з тих пір, будь-яку іншу можна додавати у папку з використанням тих же процедур (максимальне у папці може бути 12 іконок на iPhone і IPod і 20 на iPAD). Назву папки автоматично вибирається за типом програм усередині, але ім'я може також бути змінене користувачем.

Вбудовані програми

iPhone екран містить ці стандартні «програми». Деякі з цих програм за замовчуванням приховані і доступні користувачеві через програму налаштування або іншим методом. (Наприклад, Nike + iPod активується через налаштування, а AirPrint активується, коли користувач друкує файл.

1.2 Ігрова індустрія та інтернет ринок

Індустрія відеоігор або індустрія інтерактивних розваг — економічний сектор, пов'язаний з розробкою, просуванням та продажем відеоігор. У неї входить велика кількість спеціальностей, за якими працюють тисячі людей по всьому світу.

Індустрія відеоігор зародилася в середині 1970-х років як рух ентузіастів і за кілька десятиліть виросла з невеликого ринку в мейнстрім з річним прибутком в 9.5 мільярдів доларів у США в 2007 році і 11.7 мільярдів у 2008 році (за звітамиESA).

Сучасні персональні комп'ютери дали безліч нововведень ігрової індустрії. До числа найбільш значимих відносять звукові та графічні карти, CD-і DVD-приводи, Unix та центральні процесори. Зокрема, операційна система Unix була розроблена для запуску комп'ютерної гри про дослідження космосу[1][2].

Звукові карти спочатку були розроблені для інтегрування якісного цифрового звуку в комп'ютерні ігри, і тільки потім звукове обладнання було вдосконалено під потреби меломанів[3].

Графічні карти, які на зорі комп'ютерної епохи еволюціонували у напрямі збільшення кількості підтримуваних кольорів, пізніше стали розвиватися для графічних інтерфейсів користувача (англ. *GUI*) та ігор. GUI зіграв роль у збільшенні дозволів екрану, а ігри - в 3D прискорення і винаході таких технологій, як SLI і CrossFire. Використання CD і DVD дисків для розповсюдження ігор породив необхідність у збільшенні обсягів і швидкостей читання.

Бен Сойєр (англ. *Ben Sawyer*) з Digitalmill розділив ціннісну мету ігрової індустрії на шість рівнів:

- **Рівень видавництва і капіталу**: видавництва беруть участь у фінансуванні розробки нових ігор і купівлі ліцензій.
- **Рівень умінь**: включає в себе розробників, дизайнерів та художників, які можуть працювати як за індивідуальними контрактами, так і як частина групи розробників.
- **Рівень виробництва та інструментів**: тут виробляються інструменти для створення ігрового контенту, що настроюються ігрові Пакети та інструменти для управління розробкою.
- **Рівень поширення**: випуск та просування ігор для продажу в роздрібних мережах та сервісах цифрової дистрибуції.
- **Рівень апаратного та програмного забезпечення**: сюди входять апаратні базиси-платформи, такі як консолі та мобільні пристрої. У цей рівень зараз входять і неаппаратние платформи, такі як віртуальні машини (наприклад Java абоFlash)

або програмні платформи, такі як браузери та (останнім часом) Facebook.

- **Рівень кінцевих користувачів** (інакше кажучи, споживачів ігор або геймерів)

В ігровій індустрії часто працюють ті, хто вже мав якийсь досвід у близьких до неї професіях (таких як програміст, художник, музикант), проте деякі мають досвід у роботі стосовно лише до ігрової індустрії: ігровий програміст, ігровий дизайнер, дизайнер рівнів, ігровий продюсер, ігровий художник і тестер ігор. Багато хто з них наймаються фірмами-розробниками або видавцями комп'ютерних ігор. Однак багато одинаки пишуть ігри самостійно, а потім продають їх.

Початком епохи комп’ютерних ігор можна вважати появу "Space War", розроблену С. Расселом 1962 року в Массачусетському технологічному інституті. Але висока вартість комп’ютерів не дозволяли тоді цій грі

стати масовим захопленням. Першу доступну усім комп’ютерну гру розробив Н. Бушель. Вона з’явилася в США 1972 року і називалася "Понг" (електронна версією пінг-понгу). Після цього з’явилося безліч ігор, які, власне кажучи, нічим не відрізнялися від "Понга", але мали інше зовнішнє оформлення, наприклад, електронний хокей, футбол тощо. Однак знадобилося ще близько десяти років, аби комерційні комп’ютерні ігри дійшли у своєму розвитку до рівня складності "Космічної війни".

Справа в тому, що гра є розвагою для користувача, а для виробника це необхідність використання новітніх досягнень у галузі техніки, зокрема, у графіці, програмуванні складних видів руху та спеціальних ефектів тощо.

Навіть сьогодні тільки деякі прикладні програми, що використовуються більшістю користувачів, є такими ж ресурсомісткими і вибагливими до

устаткування, як ігри. Особливо це стосується відеоплати, процесора, материнської плати і монітора. Для 97% завдань, що їх виконує домашній комп'ютер, немає жодної необхідності у надсучасних і дорогих системах. Найвимогливішими до устаткування залишаються саме ігри, хоча останнім часом розширення завдань, що ставляться перед домашнім комп'ютером, поступово додає в список програмного забезпечення і нові розділи, наприклад, відеофільми.

Існує кілька варіантів класифікації комп'ютерних ігор. Усі вони умовні, оскільки з'являється безліч ігор, що поєднують у собі елементи кожної категорії. Один з найпоширеніших варіантів класифікації має такий вигляд:

- ігри типу "action", у тому числі і "RPG";
- ігри пригодницькі, типу "quest";
- ігри стратегічні;
- ігри, що імітують транспорт;

- віртуальне казино.

Ігри типу "action". 1994 року, відразу після своєї появи, набула популярності гра "Doom". Вона посідала перші місця у рейтингах, одержувала нагороди і стала джерелом ідей для цілої низки подібних ігор. Багато з них у свою чергу стали бестселерами, наприклад, "Doom Ultimate", "Quake" (від 1 до 3), "Hexen", "Unreal", "Duke Nukem 3D" (усі випуски), "Half-Life", "Serious Sam" (1 і 2) тощо.

Існує ряд особливостей, характерних для ігор сімейства "Doom". Одна з основних – це наявність багаторівневого лабіринту з пастками. Інша – обов’язкова наявність небезпечних монстрів. І, звичайно ж, герой, з яким ідентифікується гравець. Основна ідея гри – припущення, що людина може пройти лабіринт, подолати монстрів і досягти мети.

"Doom"-подібні ігри постійно використовують архетип чудовиська. Їх світи населяють мерці, що

оживають, демони, інопланетні чудовиська, агресивні роботи. Взагалі, розумні нелюдські істоти – традиційний елемент культури. Він реалізовувався в усній і писемній творчості, пізніше – у кінематографі, зараз – у віртуальних світах. Знайомство з монстрами починається в дитинстві, при читанні казок, міфічних оповідей. Пізніше, коли людина дорослішає, чудовиська, якими вона себе лякає, стають небезпечнішими (наприклад у романах С. Кінга та фільмах жахів). Сьогодні це, все частіше, чудовиська з комп'ютерних ігор.

Крім подібних "Doom" ігор до цього розділу можна зарахувати ігри військового характеру. У них, як правило, використовується як основа реальний чи близький до реального історичний простір. Прикладом таких ігор можуть слугувати "Delta Force", "Rainbow" та її варіації, "Hitman" і багато інших. У них ворогами є люди, і завдання гравця, керуючи героєм, чи,

найчастіше, командою, перемогти всіх "поганих хлопців".

Комп’ютерна гра такого типу може формувати певні корисні навички. Гравець навчається швидким, точним і тонким рухам рук. Інша навичка, яку формують "Doom"-подібні ігри, це здатність працювати у ритмі, заданому особливостями процесу. З цього погляду "Doom"-подібна гра є засобом навчання швидкому опрацюванню інформації в умовах високої нервової напруги та дефіциту часу.

Ігри з елементами рольового моделювання дуже різноманітні. Більшість з них мають свій складний і захоплюючий світ. Наприклад, "Morrowind" (1, 2, 3) має свою історію, свої раси розумних істот, що дає можливість гравцеві прожити життя одного з персонажів з усіма його особливостями. Частина рольових ігор дуже схожа на звичайний "action", інші набагато ближчі до стратегій. Наприклад, серія "Heroes

of Might and Magic" (2, 3, 4), що захоплює вже третє покоління гравців, за багатьма параметрами близька до стратегій, а подібна до неї "Might and Magic" (6, 8, 9), як рольовий "action", перебуває на протилежному полюсі. "Aliens vs Predator" (1 і 2) містить у собі як рольові елементи, так і всі основні елементи "action".

Особливістю таких ігор є можливість обирати героя чи групу героїв, розвивати в них специфічні здібності, а іноді, як у "Aliens vs Predator", навіть бачити навколишній світ по-різному – залежно від героя, від імені якого ведеться гра. Цікаво, що в цьому випадку підлітки і діти набагато швидше адаптуються до зміни сприйняття. Багато дорослих людей, що спробували грати "Чужого" з його незвичайним зором, відчувають запаморочення і нудоту. Зате не менш незвичайний "потрійний" зір "Хижака" легко сприймається всіма. Скоріш за все, це пов’язано з особливостями програми, що описує рух цих істот.

Ігри подібного типу розвивають гнучкість мислення, здатність ставитися до будь-яких героїв та істот як до одного з варіантів різноманітного світу. Правота і неправота в цих іграх, як і в реальному світі, відносні. Усе залежить від того, на чиєму ти боці. Та навіть при відносності "правди" більшість з ігор цього типу має чітке протиставлення "добра" і "зла".

Для більшості подібних ігор не досить тільки швидкої реакції. Набагато частіше необхідне розвинене комбінаторне мислення та здатність прогнозувати неоднозначні ігрові ситуації. Кожна з ігор цього розділу формує внутрішню переконаність у тому, що немає непереможних монстрів. Монстри небезпечні, та герой може знайти засіб для перемоги. А для перемоги, на відміну від реального світу, є можливість зробити кілька спроб. Наявність функцій "autosave" та "save" значно розширює простір пошуку кращого рішення у кожній ситуації.

Стратегії. Стратегічні ігри неоднорідні: одні пропонують тільки "мир", інші – "війну", треті – те і те одночасно. У мирному жанрі до лідерства близьке сімейство ігор "SimCity": "SimCity-2000", "Sim Tower", "SimCity-3000". Для військового жанру найтиповіша гра "Warhammer: Dark Omen". Війна і мир, як жанр стратегічних ігор, найбільший за кількістю назв. Багато гравців вважають ігри "Age of Empire" і "Civilization", а також їх похідні еталоном сучасних стратегічних ігор.

Найближчим прототипом подібних стратегічних ігор є шахи. Ілюзія шахів, що ожили, реалізувалася у вигляді стратегічної комп'ютерної гри: людина дивиться на екран монітора і бачить у тривимірному просторі макет місцевості з живими істотами. Віртуальну країну можна наблизити, віддалити, оглянути під іншим кутом зору. Її розмір може сягати сотень квадратних кілометрів. На ній існують і переміщуються з волі гравця і за власним алгоритмом безліч фігур різних видів. Їх зовнішній вигляд постійно

змінюється. Військові загони зазнають втрат, але здобувають бойовий досвід, цивільні об’єкти нарощують чи втрачають міць. Часовий проміжок гри може досягати тисячоріч.

Як правило, ігрове поле є макетом місцевості. Іноді воно розмічене на клітинки, як у "SimCity", частіше – ні, але наявність структури визначає все, що відбувається у більшості ігор. Фігура – це об’єкт, яким гравець може керувати як цілим. Частина фігур статичні: наприклад, заводи, електростанції, будинки. Ступінь і швидкість їх розвитку залежать від усіх інших фігур дошки. Інші фігури рухливі, наприклад, смерч, тарілки, що літають, чудовиська. У військових іграх більшість фігур рухливі. Спочатку гравець розставляє свої фігури – загони кінноти, піхоти, лучників – на відведеному для цього місці, а потім може їх відправити в будь-яку зону дошки. Ходи супротивника розраховує комп’ютер.

Стратегічна гра складається з ряду рівнів. У мирних іграх основний показник якості гри – чисельність населення. Коли він перевищує заздалегідь задане для даного рівня число, у гравця з'являється можливість використовувати додаткові види фігур. У військових іграх головне – розгром ворога. Якщо гравцеві це вдається, то він одержує нову дошку з іншим рельєфом місцевості та потужніші фігури. Ігри, що поєднують війну і мир, вимагають спочатку побудувати інфраструктуру для армії, а вже потім знищувати ворога. Військова мета у цьому випадку тісно пов'язана з мирною.

Більшість ігор не дозволяє гравцеві бачити всю дошку відразу. Гра звичайно відбувається на дошці, більшій поля зору. Вірніше, дошку можна вмістити, але для цього доведеться її так зменшити, що стане неможливо розрізнити, що на ній відбувається. Велику частину ігрового часу гравець розглядає дошку у великому масштабі. Та щоб виграти, йому необхідно

точно уявляти, що і де на дошці відбувається. Для цього гравець повинен мати внутрішній образ дошки. З ним він співвідносить усі свої дії, постійно уточнюючи і корегуючи його в міру забування.

У процесі гри формуються навички системного аналізу**.** Гравцеві необхідно швидко і правильно опрацьовувати інформацію, а потім на її основі прогнозувати розвиток подій. Ще одна навичка, яку формують ігри подібного типу, це досвід роботи з "чорною скринькою" як пристроєм з невідомим принципом роботи, що має вхід і вихід.

Експериментуючи з введенням даних, гравцю необхідно домогтися потрібної реакції на виході. На початку гри майже усі фігури для гравця – "чорна скринька". Якусь інформацію про їх взаємозв’язки дають опис та система підказок. В основному гравець робить припущення, керуючись здоровим глуздом. Інколи вони виправдовуються, інколи – ні. В іграх

цього типу можливе формування навичок експериментування та пошуку оптимальних варіантів діяльності в умовах дефіциту інформації.

Квест. Слово "quest" означає пошук, предмет, що відшукується, пошуки пригод, дізнання. У цих іграх реалізується одне з занять, що захоплюють людей різного віку – розгадування загадок. Людську потребу розкривати таємниці давно використовують театр, кінематограф і література. Мистецтво та засоби масової інформації могли запропонувати тільки пасивну участь у таких іграх. Гравець у комп'ютерній грі, на відміну від читача чи глядача, розгадує загадку сам.

У віртуальному світі квесту воля гравця у пересуванні набагато менша, ніж у "Doom" -подібній грі. У визначеній зоні простору гравець обирає шлях, потім пасивно спостерігає переміщення до нового місця. За структурою квест схожий на більшість

комп’ютерних довідкових та інформаційних систем з гіпертекстом. Можливість перейти зі сторінки на сторінку в квесті залежить від попередніх дій гравця: поки він не виконає їх у мінімальному обсязі, це неможливо. У момент переходу гра демонструє мультфільм чи фрагмент відеофільму. На кожен свій крок гравець одержує реакцію. Квест вчить враховувати зворотний зв’язок та визначати, вірні припущення гравця чи ні. В іграх цього типу можливе формування навичок дедуктивного та індуктивного мислення.

Ігри, що імітують. Найчастіше комп’ютерні ігри цієї категорії імітують керування транспортним засобом. Вони дають можливість "приміряти" нову соціальну роль: пілота літака, вертольота, командира танка, водія реальних і фантастичних машин. Ігри, що імітують, більше за інші типи ігор використовують історичні факти, особливо з розвитку техніки. Ігри, що імітують, широко використовуються не тільки для гри, але й для

формування навичок керування реальними об’єктами і процесами. Завдяки цьому в іграх такого типу можливе формування навичок, необхідних при реальному управлінні технікою.

Віртуальне казино. Розроблено безліч ігор, що імітують реальні азартні ігри. Кожна з них будується на дотриманні тих же правил, що й у реальній грі. Тому основні прийоми цілком збігаються з грою в реальному світі. Основна різниця в тому, що не одержуєш реальних грошей при виграші. Хоча при використанні Інтернету ці ігри перетворюються на звичайні азартні. Сформовані навички не відрізняються від тих, що формуються у звичайних іграх цього типу. Для успішної гри в карти необхідна логіка, схильність до дедуктивного та індуктивного мислення, навички до рефлексії. В іграх з автоматами, де є вірогідність виграшу залежно від дій гравця чи результату якихось подій, наприклад, перегонів, може формуватися

навичка врахування у реальному житті положень теорії вірогідності.

Позитивний потенціал більшості комп'ютерних ігор реалізується далеко не завжди. І це залежить здебільш не від самої гри, а від людини, що грає, від того, який мотив переважає при включенні до гри. Крім основного мотиву розваги, гра може реалізувати інші мотиви. У залежності від мотивів другого плану можуть формуватися абсолютно різні для різних гравців навички та уміння. Реалізація мотиву тренінгу призводить до формування навичок у сфері, що тренується, а мотив компенсації внутрішніх проблем скоріш матиме результатом формування механізмів психологічного захисту. У зв'язку з розмаїттям мотивів гра може сприяти як підготовці до зустрічі з реальністю, так і втечі від неї.

У міру того, як захоплення комп'ютерними іграми стає все поширенішим, проблема їх впливу на психіку

людини та формування різних навичок потребує більшої уваги.

Для адекватного прогнозування соціальних та індивідуальних наслідків комп’ютеризації населення необхідне формування національної і міжнародної програм дослідження цього феномена.

Indie games

Незалежні ігри або **інді-ігри** (англ. indie games від independent — «незалежний») — комп'ютерні ігри, створені незалежно від фінансової підтримки великих видавців. Часто ці ігри дешеві або безкоштовні, багато незалежних ігри мають невеликий розмір і тому поширюються через інтернет за допомогою цифрової дистрибуції або як freeware. Більшість спочатку вільних ігор також належиаь до цієї категорії.

З 1998 року щорічно проводиться Independent Games Festival в рамках конференції Game Developers Conference. З 2005 року проводиться фестиваль

IndieCade. Деякі інді-ігри стали дуже успішним у фінансовому відношенні, наприклад, Braid, World of Goo і Minecraft.

Ринок коп'ютерних ігор в Україні

В Україні щороку зростає кількість людей, що купують комп'ютерні ігри. Якщо для гравців це просто забавка, то для розробників, виробників та розповсюджувачів — досить вигідний бізнес.

Світовий ринок комп'ютерних ігор оцінюють в сотні мільярдів доларів. На Заході один ліцензійний ігровий диск коштує $40–50. Популярну гру можуть продати накладом від мільйона до кількох десятків мільйонів примірників.

Не дивно, що в розробку гри там можуть легко вкласти кілька мільйонів доларів. Ця індустрія приносить великі прибутки і державній скарбниці. Комп'ютерні ігри стали вже й елементом політики. Парламенти західних країн дискутують щодо законодавчого обмеження насильства в комп'ютерних іграх або ж стимулювання виробництва ігор як такого.

Ще років 10 тому Україна своїми здобутками у світовій індустрії електронних розваг похвалитися не могла. Єдиною комерційно успішною грою від українських програмістів були ”Козаки”. Вона отримала нагороди у багатьох європейських країнах. Але сьогодні у нас є вже кілька успішних компаній, які виробляють комп’ютерні ігри і продають їх за кордоном.

В Києві пройшла виставка вітчизняних розробників під назвою ”Територія ігор”. Вона засвідчила: в нашої ігрової індустрії є великі перспективи стати потужною, орієнтованою на експорт галуззю економіки.

— Багато українських проектів уже на рівних змагаються із західними, — каже один з організаторів виставки й головний редактор журналу ”Gameplay” Сергій Гальонкін, 28 років. — Наприклад, ”В тылу врага-2” чи ”Герои уничтоженных империй”.

Нині в Україні діють 90 професійних студій, які готують близько 100 ігрових проектів міжнародного

масштабу. Торік було продано понад 10 млн ліцензійних копій комп'ютерних іграшок. На думку генпродюсера однієї з російських компаній Сергія Клімова, до 2008 року Україна й Росія разом випускатимуть по 10–12 великих і прибуткових ігор щороку.

Харківська команда CRAZY HOUSE зараз працює над національно-свідомим квестом - "Вій: історія, розказана знову", за Гоголем. У грі співіснуватимуть "живі" герої (для цього на відео знімали харківських акторів театру, серед яких, як гордо заявили представники "скаженої фірми", є кілька заслужених і, навіть, один народний артист) разом із тривимірними анімованими моделями нечисті. А, наприклад, IRBIS Development Group створює не менш патріотичний проект "Футбольний менеджер: серце тренера", присвячений пам'яті Валерія Лобановського...

До слова: Росія нас у ігровій індустрії поки що випереджає. Російські компанії приваблюють

іноземців. Ті або купують їх, або вкладають в них свої гроші. Наші ж поки що працюють переважно самотужки.

App Store

App Store — розділ онлайн супермаркету iTunes Store, що продає власникам мобільних телефонів iPhone, плеєрів iPod Touch та планшетних ПК iPad різні додатки. Українська версія відсутня.

App Store містить понад 550 тисяч додатків (станом на початок 2012 року).

Вартість цих програм/додатків складає від $0,99 до $9,99, за деякими винятками. В Україні можливе придбання програмного забезпечення за допомогою банківської картки Visa. В магазині App Store також є розділ з безкоштовними програмами, або пробними версіями звичайних.

Користувачам iPhone 3G доступ до магазину було відкрито відразу в момент початку продаж цієї моделі.

Але попередня модель потребувала оновлення ПЗ. Доступ до App Store також можливий з iTunes, починаючи з версії 7.7.

За магазином слідкують спеціалісти Apple і кожен застосунок проходить перевірку та отримує електронний сертифікат. Від продажу застосунків автор отримує 70 %, Apple 30 %, для підтримки магазину.

В iPod Touch сервіс App Store працює за наявності інтернету через Wi-Fi.

Додатки і контент в магазині App Store надаються у двадцяти категоріях:

1. **Книги** (англ. *Books*)
2. **Бізнес** (англ. *Business*)
3. **Освіта** (англ. *Education*)
4. **Розваги** (англ. *Entertainment*)
5. **Фінанси** (англ. *Finance*)
6. **Іграшки** (англ. *Games*).

6.1 **Підкатегоріїї**: Action; Adventure; Arcade; Board; Card; Casino; Dice; Educational; Family; Kids; Music; Puzzle; Racing; Role Playing; Simulation; Sports; Strategy; Trivia Word

7. **охорона здоров'я та фітнес** (англ. *Healthcare & Fitness*)
8. **Стиль життя** (англ. *Lifestyle*)
9. **Медицина** (англ. *Medical*)
10. **Музика** (англ. *Music*)
11. **Навігація** (англ. *Navigation*)
12. **Новини** (англ. *News*)
13. **Фотографія** (англ. *Photography*)
14. **Продуктивність** (англ. *Productivity*)
15. **Словники, довідники** (англ. *Reference*)
16. **Соціальні мережі** (англ. *Social Networking*)
17. **Спорт** (англ. *Sports*)
18. **Подорожі** (англ. *Travel*)
19. **Утиліти** (англ. *Utilities*)

20. **Погода** (англ. *Weather*)

1.3 Інструментал для розробника

Розробники, які програмують під девайси фірми Apple, мають не дуже широкий вибір інструменталу. Проте, ті засоби для написання програмного продукту що доступні на даний момент, цілком задовольняють потреби кодерів.

Одним із найпотужніших середовищ розробки можна виділити Xcode.

Xcode — інтегроване середовище розробки (IDE) виробництва Apple. Дозволяє створювати програмне забезпечення з використанням таких технологій як GCC, GDB, Java та ін. На сьогодні є єдиним засобом написання «універсальних»(Universal Binary) прикладних програм для Mac OS X.

Xcode є інтегрованою середовищемрозробки від Apple Computer (IDE), який використовується для запису, створення та тестуванняпрограм для роботи в операційній системі MacOS X. З використанням Xcode, ви можете створювати широкий спектр додатків, плагінів, драйверів і багато іншого з використанням різних технологій і мов. Він включає в себе інструменти для розробки, редагування, аналізу, налагодження, тестування, упакування і поширювання своїх проектів. Він може бути використаний як індивідуальний розробник або співпрацювати з групою розробників.

IDE є "клеєм", який збирає разом і дозволяє керувати всіма маленькими шматочками, які необхідні для створення сучасного програмного забезпечення. Цілком можливо, редагування, компіляції, посилання, розшарування та тестування програмного забезпечення без використання IDE. Ви можете редагувати файли за допомогою редактора, зберегти їх, запустити компілятор для компіляції вихідного коду, запустити

компонувальник, позв'язуватио б'єктні файли в програми, а потім запустити відладчик, щоб перевірити це. Додатки стають все більш складними, тому і інструменти, необхідні для їх виробництва теж стають більш складними. Навіть "Простий" проект може використовувати десятки різних інструментів для створення працюючого додатку.

Xcode не поставляється сам по собі. Він є лише складовою частиною цілого комплекту засобів розробника під назвою iOS SDK.

iOS SDK (раніше iPhone SDK) - комплект засобів розробки для iOS, випущений в березні 2008 року корпорацією Apple. iOS SDK(Software Development Kit) випускається тільки для Mac OS X.

Історія даного комплекту починається 17 жовтня 2007, коли у відкритому листі, опублікованими в блозі «Гарячі новини» компанії Apple, Стів Джобс, засновник компанії, анонсував SDK, який повинен був бути наданий стороннім розробникам в лютому 2008

року. Однак інструментарій вийшов лише 6 березня. Він дозволяє розробляти додатки під iPhone, iPod Touch і iPad, а також тестувати їх на емуляторі iPhone. Тим не менш, завантаження програми на пристрої можлива тільки після оплати ліцензії. Починаючи з Xcode 3.1 він є засобом розробки для iOS SDK.

Серед особливостей можна виділити те, що розробники можуть встановлювати будь-яку ціну, що перевищує мінімальну встановлену, за їх застосування, які будуть поширюватися через App Store, з якої вони будуть отримувати 70%. Крім того, вони можуть поширювати свій додаток безкоштовно, в цьому випадку вони повинні платити тільки членські внески, які складають 99 доларів на рік.

Склад SDK:

SDK розбита на наступні розділи:

Cocoa Touch

Мультитач управління

Підтримка акселерометра

Ієрархія видів

Локалізація

Підтримка камери

Мультимедіа

OpenAL

Мікшування і запис аудіо

Відтворення відео

Формати зображень

Кварц

Анімаційне ядро

OpenGL ES

Сервісне ядро

Мережа

Вбудована база даних SQLite

Локаційне ядро

Багатопоточність

CoreMotion

Ядро OS X

TCP/IP

Сокети

Управління живленням

Файлова система

Безпека

1.4 Objective–C як основна мова в середовищі Xcode

Objective-C - це є основна мова програмування для оперційної системи Mac і відповідно iOS. Якщо програміст знайомий з принципами ООП (об'єктно-орієнтованого програмування) і мовою програмування "C" то йому буде дуже легко освоїти Objective-C. Якщо ж ні, то не буде лишнім почитати інші допоміжні матеріали щодо азів програмування.

Objective-C. Основи

Мова програмування Objective-C є надбудовою чи краще сказати розширенням C і як наслідок

компілятор Objective-C вільно компілює C-код. Крім цього Objective-C також базується на приципах і запозичила основну семантику мови програмування SmallTalk, тобто замість виклику методів об’єкта ми відправляємо йому повідомлення. Наприклад, якщо об’єкт реалізовує метод doSmth, то вважаємо, що він відкликається на повідомлення doSmth.

```
[object doSmth]  // Приклад коду
```

Перевагами, є те що будь-якому об’єкту можна відправити будь-яке повідомлення, при чому цей об’єкт може проігнорувати це повідомлення, або переслати (делегувати) комусь іншому, на противагу мові програмування C++, де виклик методу, який не реалізовується викликаючим об’єктом приведе до генерації виключної ситуації (Exception) та до можливого краху програми.

Можна відправити повідомлення з параметрами

[object doSmthWithParams:data otherData: data1];

В цьому випадку іменем повідомлення є ” doSmthWithParams:otherData:”

Також об’єкт, який обробив повідомлення може повертати значення

output = [object doSmth];

Можна використовувати вкладені повідомлення, тобто результат одного повідомлення може використовуватись як параметр іншого

[object doSmthWithParams:data otherData: [object doSmth]];

Крім цього хотілося б згадати ще декілька особливостей мови програмування Objective-C. Це

використання універсального типу даних id. Змінна типу id фактично являється вказівником на будь-який об’єкт. Для нульових вказівників використовується константа nil. До речі, відправка повідомлення нульовому об’єкту є цілком законною операцією на відміну від C++. Результатом буде повернення теж нульового об’єкта.

Можна ще згадати відсутність вбудованого булєвого типу даних для логічних величин. Тому замість нього використовується штучний тип даних BOOL із двома можливими значеннями YES і NO.

Класи

Цікаво, що класи в Objective-C самі являються об’єктами (class objects). Їхня основна задача – це створення конкретних екземплярів об’єктів. Тобто з однієї сторони ім’я класу виступає типом даних, а з іншої - об’єктом, якому можна відправити повідомлення. Нижче описана структура класу. Як

правило класи описуються в заголовочних файлах з розширенням *.h

```
@interface HelloWorldViewController : UIViewController
{
UITextField* edText;
UILabel* lbText;
}

@property(nonatomic,retain) (IBOutlet) UITextField*
edText;
@property(nonatomic,retain) (IBOutlet) UILabel* lbText;

-(IBAction)start;
@end
```

Зверніть увагу, що всі директиви компілятору починаються із знаку @. Нехай вас не бентежить слово @interface перед назвою класу, це особливості мови програмування, але це все-таки клас. Класичний

інтерфейс, тобто клас, методи якого не можуть бути реалізовані, позначається як @protocol. Директива @class також використовується, але попередньої декларації класу. Для об'єктів дозволено наслідування, але тільки не множинне. Об'єкт може бути породженний від іншого об'єкту і одночасно реалізовувати багато протоколів. Варто звернути увагу, що всі методи класу є віртуальними, тому можуть бути перевизначені в нащадках. На відміну наприклад від C++, методи класу є завжди публічними (public). Хоча видимістю даних (полів) екземплярів класу можна керувати за допомогою аналогічних директив @private, @public, @protected. По замовчуванню, всі вони є @protected. Для доступу до полів об'єкта використовується таке поняття як властивості - @property

```
@property(nonatomic,retain) (IBOutlet)
UITextField* edText;
```

В дужках записані атрибути властивостей. Вони можуть бути наступними:

readwrite – поле доступне для запису і читання. Тобто є акцесор і мутатор (getter,setter)

readonly – тільки для читання (getter)

assign – використовується для присвоєння змінній значення стороннього об’єкту, яким ми не володіємо і не можемо керувати

retain – вказує на те, що об’єкту, який використовується в якості нового значення надсилається повідомлення retain (що воно означає буде більш детально описано, коли ми дійдем до управління пам’яттю)

copy – вказує на те, що для присвоєння буде використовуватись копія об’єкту

atomic, nonatomic – атрибути, які є важливими для багатопотокового програмування. Атрибут atomic гарантує, що ви отримаєте цілісні дані, навіть якщо під час доступу до даних інший потік теж намагається отримати доступ до них і модифікувати. Відповідно

nonatomic – навпаки, але його використання пришвидшує роботу. Тому коли немає доцільності використання атрибуту atomic (який присвоюється по замовчуванню), краще використовувати nonatomic.
Назва акцесора (getter) співпадає із назвою змінної – поля об'єкту. До назви мутатора (setter) додається префікс set. Так виглядають getter і setter для однієї з вищевказаних властивостей

```
-(void)setEdText:(UITextField*)value;
-(UITextField*)edText;
```

Якщо ви звернули увагу декларація методів починається із знака (-) або (+). Знак "+" означає, що це метод є виключно методом класу (не об'єкту) і його як повідомлення можна відправляти тільки class object. Можна провести аналогію із статичними методами наприклад в C++. Метод зі знаком "-" є стандартним методом об'єкта – екземпляра даного класу. Опис класу закінчується директивою @end.

Тепер мабуть варто перейти до реалізації класу. Як правило це робиться у файлі з розширенням *.m. Реалізація методів класу починається з директиви @implementation і закінчується @end.

```
@implementation HelloWorldViewController
@synthesize edText;
@synthesize lbText;
-(IBAction)start
{
lbText.text = edText.text;
}
@end
```

Ви мабуть вже звернули увагу на незрозумілу директиву @synthesize. Дуже корисна річ. Вона позбавляє розробника від нудної роботи генерувати код для setters і getters, тоді, коли він є однотипним і не потребує додаткової логіки. Тобто написавши строчку коду @synthesize edText, ви вже можете

використовувати властивість в межах, описаних її атрибутами. Все просто.

Далі хотілося б згадати ще одну корисну річ, яка досить широко використовується – це @selector. Селектор дозволяє однозначно ідентифікувати метод об'єкта і передавати його як параметр при відправці повідомлення. Це чимось схоже на вказівник на функцію в C++. Типом селектора є SEL. Отримати селектор по імені метода можна за допомогою наступної конструкції

SEL doSmth = @selector(doSmth); //метод doSmth
або

SEL doSmthWithParams = @selector(doSmth:) //метод doSmth:(id)data

Крім цього, використовуючи те, що в Objective-C реалізована потужна підтримка метаінформації, можна прямо на етапі виконання перевірити чи підтримує

об’єкт певний метод за допомогою відправки об’єкту повідомлення respondsToSelector:

```
if ( [object respondsToSelector: @selector(doSmth:)] )
  [object doSmth: data];
```

Також можна відправити об’єкту повідомлення, яке відповідає вибраному селектору за допомогою performSelector: чи performSelector:withObject:

```
[anyObject performSelector:@selector(doSmth:)
withObject: nil];
```

І ще одна річ, яка мені дуже подобається, це використання селекторів для відправки відкладених в часі повідомлень performSelector: withObject: afterDelay: Тобто із затримкою, яка вимірюється в секундах. Досить корисна особливість, яка надає гнучкості вашій програмі без реалізації додаткових потоків, які б значно ускладнювали програму.

[anyObject performSelector:@selector(doSmth:) withObject: nil afterDelay: 30.0];

Але використання цих особливостей вимагає, щоб ваші об’єкти були породжені від NSObject – кореневий клас для класів Cocoa Framework (основне середовище розробки продуктів Apple). Про це пізніше, оскільки обговорення конкретних інструментів розробки та API виходить за рамки даної статті.

Протоколи

Протоколи, це ще одна важлива особливість мови програмування Objective-C, яка була згадана раніше. Протокол – це повністю абстрактний клас, який ще деколи називають інтерфейсом, але так як назва “інтерфейс” вже занята будемо використовувати @protocol. Протокол містить тільки опис методів і можна сказати, що об’єкт реалізовує певний протокол, якщо він містить реалізацю методів описаних протоколом. При чому методи, які описує протокол можуть бути як обов’язкові для реалізації так і

реалізовуватись по потребі. Для цього слугують директиви відповідно @required і @optional. По замовчуванню використовується @optional. Нижче приведено приклад опису протоколу.

```
@protocol Serializable
@required
  - (id) initWithCoder: (NSCoder *) coder;
@optional
  - (void) encodeWithCoder: (NSCoder *) coder;
@end
```

Протокол може наслідувати довільну кількість протоколів. Для перевірки чи підтримує об’єкт певний протокол, можна скористатись повідомленням conformsToProtocol

```
if ( [obj conformsToProtocol: @protocol (Serializable)] )
  [obj encodeWithCoder: myCoder];
```

Для явного вказання компілятору, що об’єкт підтримує певні протоколи можна використати наступну форму декларації змінної, вказавши в її імені назву протоколу:

id <Serializable,Drawable> obj;

Зверніть увагу, що тут описана змінна невідомого на етапі компіляції типу. Якщо ж тип змінної відомий, то опис буде виглядати так:

anyOject<Serializable,Drawable> *obj;

Категорії

Категорії – ще одна корисна особливість Objective-C. Аналогом є Extensions (розширення) наприклад в C#. Інколи виникає питання як розширити функціональні можливості класу, до реалізації якого ви не маєте доступу. Одним із способів є наслідування, коли можна в породженому класі реалізувати необхідну функціональність. Іншим більш елегантним способом в даному випадку є використання категорій, що дозволяє

описати і реалізувати необхідні додаткові методи для існуючого класу. Нижче поданий приклад опису і реалізації методів описаних категоріями.

```
#import "HelloWorldViewController.h"
@interface HelloWorldViewController ( CategoryName )
  -(void) doMore;
@end
#import "HelloWorldViewController.h"
@implementation HelloWorldViewController (
CategoryName )
-(void) doMore{
  NSLog(@" We are doing more!");
}
@end
```

Обробка виключних ситуацій

Обробка виключних ситуацій аналогічна як в інших мовах програмування. Все практично ідентично, навіть директиви: @try, @catch, @finally і @throw.

Наприклад:

```
@try
{
  [obj doSmth];
}
@catch ( NSException * e )
{
  NSLog ( @"Exception caught: %@", e );
}
@finally
{
  [obj release];
}
```

І відповідно для генерації виключної ситуації використовується @throw

```
NSException * e = [NSException exceptionWithName:
@"myException" reason:@" an_error" userInfo: nil];
@throw e;
```

Управління пам’яттю

На відміну від інших високорівневих мов програмування, в яких для полегшення роботи програміста реалізовано автоматичне вивільнення використаної пам’яті (т.з. Garbage Collector), програмування для iPhone вимагає робити це вручну. Хоча використання Garbage Collector дозволено для програмування під Mac OS, але використання такої ресурсоємкої особливості є розкішшю для мобільного пристрою, навіть такого як iPhone

Виділення пам’яті для нового об’єкту відбувається за допомогою відправки йому повідомлення alloc або copy. В першому випадку повідомлення відправляється клас-об’єкту (class object), який відповідно створює новий об’єкт. Повідомлення copy, як ви вже мабуть здогадались, копіює вже існуючий об’єкт в нову область пам’яті. Крім цього, існує що одна важлива особливість

управління пам’яттю – це принцип “володіння об’єктом”. Тобто можна стати повноцінним власником об’єкта і використовувати його для власних потреб, не створюючи і не копіюючи його. При цьому, будучи впевненим, що він ніким і ніде в програмі не буде неочікувано видалений (звичайно, при дотриманні цих правил). Для цього слугує механізм підрахунку посилань на цей об’єкт. Тобто кожен новостворений об’єкт має лічильник посилань на нього, значення якого при створенні стає рівним 1. Отримати об’єкт у власність можна відправивши йому повідомлення retain, яке інкрементує лічильник посилань. Відповідно цей об’єкт не буде знищений до того часу поки значення його лічильника буде більше нуля.

Тепер відносно вивільнення зайнятої пам’ятті. Звільнити виділену пам’ять можна за допомогою відправки об’єкту повідомлень dealloc, release, autorelease. Зверніть увагу, що повідомлення dealloc ніколи напряму об’єкту не

повинно відправлятися, оскільки це приводить до негайного вивільнення пам’яті, незважаючи на те, що об’єкт ще може десь використовуватись. Це повідомлення відправляється об’єкту автоматично системою тоді, коли, як я вже казав, його лічильник посилань стає рівним нулю. Замість нього використовується метод release, який служить для декрементації згаданого лічильника посилань. Це дуже важливе правило, яке ви повинні завжди пам’ятати для успішної роботи.

Отже, підсумуємо. Якщо вам потрібно створити об’єкт ви відправляєте його клас-об’єкту повідомлення alloc. Якщо ж потрібно використати об’єкт, який ви не створювали – ви відправляєте йому повідомлення retain, що, можна сказати, резервує його за вами. Коли ж він стає вам непотрібний – завжди відправляєте повідомленняrelease. Якщо ж кількість викликів retain більша за кількість викликів release, пам’ять зайнята об’єктом не вивільниться і можуть виникнути проблеми – т.з. memory leaks. Тому завжди

слідкуйте за цим.

Ще хотів би відмітити, що коректний дизайн класу і його подальша реалізація вимагає перевизначення методу dealloc, який є деструктором об'єкта. Відповідно реалізація цього методу повинна містити необхідні дії щодо вивільнення пам'яті іншими об'єктами – членами даного класу. Тобто, як я вже згадував, коли лічильник посилань на об'єкт стає рівний нулю, йому системою автоматично відправляється повідомлення dealloc. Зверніть увагу, що при реалізації цього методу після всіх необхідних дій, необхідно передати управління батьківському класу, який має продовжити вивільнення пам'яті. Це робиться за допомогою відправки повідомлення [super dealloc]. Це єдиний випадок коли dealloc викликається напряму. Нижче наведено приклад.

```
- (void)dealloc {

        [edText release];
        [lbText release];
```

```
    [super dealloc];
}
```

Autorelease pool

Ви мабуть помітили, що вказано ще один метод, який використовується для вивільнення пам’ятті. Це – autorelease. По назві можна здогадатись, що щось схоже на release, тільки автоматично. Часто при реалізації певного методу необхідно повернути посилання на створений об’єкт.

```
//некоректна реалізація!!!
-(NSString*) getString
{
        NSString* str= [NSString alloc];

        return str;
}
```

Але не відомо, де, коли, ким і чи взагалі він буде знищений. Звідси виходить, що ми самі повинні подбати про його знищення, викликавши release. Виникає питання – де? Якщо до виклику return, то об’єкт буде знищений ще до його повернення. В цьому випадку на допомогу приходить autorelease. Цей механізм ще можна назвати “відкладеним звільненням”. Тобто, відправивши об’єкту повідомлення autorelease ми кладемо його в спеціальний пул, де він зберігається до того моменту, коли цей пул також буде знищений.

```
-(NSString*) getString
{
        NSString* str= [[NSString alloc] autorelease];

        return str;
}
```

Що ж таке Autorelease pool? Це спеціальний об’єкт – екземпляр класуNSAutoreleasePool. Якщо глянути на файл проекту main.m, який створюється по замовчуванню можна побачити, що весь процес запуску програми відбувається між створенням і знищенням Autorelease pool.

```
int main(int argc, char *argv[]) {

   NSAutoreleasePool * pool = [[NSAutoreleasePool alloc] init];
   int retVal = UIApplicationMain(argc, argv, nil, nil);
   [pool release];
   return retVal;
}
```

Може спочатку здатись, що знищення об’єктів в пулі відбувається тільки після закриття програми, але це не так… Взагалі-то кожен цикл обробки події (event loop) має власний autorelease pool, який знищується після

закінчення цього циклу. Відповідно всім об’єктам, які знаходяться в цьому пулі відправляються повідомленняrelease стільки раз, скільки їм перед тим було відправлено повідомлення autorelease. Незважаючи на зручність такого підходу все-таки рекомендується де це можливо уникати його і використовувати release. Це пов’язано з тим, що іноді цикли обробки подій бувають досить довгі (залежить від реалізованої логіки) і за цей час в пулі може назбиратись багато непотрібних об’єктів, які ще займають виділену пам’ять, і для подальшої роботи програми пам’яті може просто не залишитись…

1.5 Фреймворки. cocos2d

Фреймворк (англ. Framework) — основна концептуальна система або структура для вирішення комплексних задач.

Програмний фреймворком (англ. software framework) є готовим до використання комплекс програмних рішень, включаючи, дизайн, логіку та базову функціональність системи або підсистеми. Відповідно програмний фреймворк може містити в собі також допоміжні програми, якісь бібліотеки коду, скрипти та загалом все, що полегшує створення та поєднання різних компонентів великого програмного забезпечення чи швидке створення готового і не обов'язково великого програмного продукту. Побудова кінцевого продукту відбувається зазвичай на базі єдиного API.

Каркас застосунку

Одна з головних переваг, при використанні каркасних застосунків, полягає в тому, що такі програми мають стандартну структуру. Каркаси застосунків стали популярними з появою елементів інтерфейсу, які мали тенденцію до реалізації стандартної структури для додатків. З їх використанням стало набагато простіше створювати засоби для автоматичного створення графічних інтерфейсів, так як структура внутрішньої реалізації коду програми стала відома заздалегідь. Для забезпечення каркасу зазвичай використовують підходи об'єктно-орієнтованого програмування, наприклад, частини програми можуть успадковуватися від базових класів фреймворка.

Концептуальний фреймворк (англ. Conceptual Framework) — це абстрактна структура чи система, що використовується в дослідженнях для визначення можливих способів вирішення проблеми, задачі чи представлення ідеї.

Багато інших англомовних термінів де присутнє слово framework легко замінюється в українському варіанті загалом на слова: основа, несуча конструкція, структура, каркас, в автоматиці та аудиті може бути концепція, в механіці стержнева система, або це може бути просто механізм.

Отже, якщо ми визначилися з тим, що ж таке фреймворк, то нам потрібно обрати той, працювати з яким буде найзручніше(звісно можна обійтися взагалі без нього, але ми такий варіант не розглядаєм).

Існує багато фреймворків під iOS: Box2D, Bullet, Chipmunk, iTorque 2D, Oolong Engine, ShiVa3D, SIO2, Unity, cocos2d, Sparrow, Wibble Quest, RestKit, LRResty, HTTPRiot, FlowCover, OpenFlow, iCarousel, Tapku, Core Plot, iVisualization, PowerPlot та багато інших.

Ми зупинимося на фреймворку cocos2d.

cocos2d написаний для вирішення різних задач у двохвимірному просторі. Він є основою для створення 2D-ігор, демонстрацій та інших графічних/інтерактивних додатків. Движок дозволяє

швидке і легке управління спрайтами. Є такі дії, як переміщення, обертання, масштабування і багато іншого. Безліч графічних ефектів. Можна створювати як 2D ігри видом зверху чи знизу, так і гри в 2D ізометрії. Стильний перехід від сцени до сцени. Вбудовані класи для створення меню. Писати на ньому можна використовуючи Objective-C. Розробники, що використовують cocos2d мають можливість використовувати величезну різноманітність конструкцій. Також він є простим у використанні, використовує знайомий інтерфейс API, і поставляється з великою кількістю прикладів. Інтегрується з фізичними движками типу Box2d і Chipmunk. Графіка використовує OpenGL ES 1.1 передового досвіду і оптимізованої структури даних. Є система часток, підтримка TMX для карт тайлів, 2D, паралакс. Є безкоштовний, з відкритим вихідним кодом, доступні відкриті і закриті ігри-приклади. AppStore

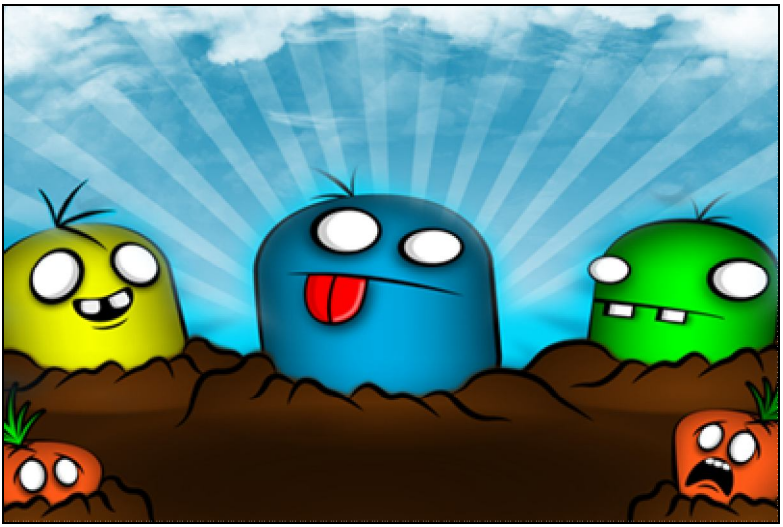
затверджено більше 2500 ігор, серед яких багато хітових ігрор.

cocos2d для iPhone підтримує: IPod Touch, iPhone, IPad і OS X.

Розділ 2. Розробка власного продукту

2.1 Основна ідея, та концепт

Ідея даного продукту полягає в тому, щоб гравець міг азартно зануритись в логічно-аркадну задачу поставлену перед ним. Сворений продукт складається з головного меню, та безпосередньо ігрової сцени.



Меню містить кнопку PLAY – старт гри, та SKINS – яка дозволяє вибрати ігровий мод: кроти чи інопланетяни.

Після запуску гравець бачить перед собою поле 6x4, тобто 24 лунки з яких в хаотичному порядку появляються ігрові персонажі. Метою є попасти пальцем по персонажу доки він не сховався. Звісно що гра продовжується не довго. У верхньому правому кутку екрану міститься три індикатори які по черзі зникають. Після того як останній індикатор зник гра завершується, а кількість попадань зберігається. При чому гравцеві дається на вибір: попробувати свої сили ще раз, чи повернутися до головного меню.

2.2 Виложення продукту на ринок AppStore

Як влаштований Apple App Store?

App Store є одним з розділів онлайн магазину ITunes Store, який займається продажем різних додатків власникам плеєрів IPod Touch, мобільних телефонів iPhone і планшетів IPad.

Отже, що ж собою представляє цей магазин? Як тільки ви зайдете в App Store (з iPhone, Mac, PC, iPhone або IPad сенсорний), ви побачите вкладку компоненти. Це ті ігри та програми, які вам рекомендує Apple. Як ви самі розумієте, передбачається, що там будуть зібрані самі краще програми. Однак це не зовсім так і ви можете натрапити на не самі якісні програми. У верхній частині екрану ви побачите три меню.

New - тут ви знайдете програми, які були випущені зовсім недавно. Так би мовити, останні новинки.

What's Hot - тут представлені найбільш популярні додатки, які скачують найчастіше.

Release Date - додатки розсортовані по датам. Наприклад, ви не пам'ятаєте, як точно називається додаток, але вам, відомо, що воно вийшло певного числа. Тоді просто знаходите це число і намагаєтеся згадати, яке з представлених там додатків вас цікавило.

У другій закладці - Top Charts будуть показані рейтинги програм по популярності.

Дана вкладка є однією з найпопулярніших в App Store.

Причому, варто зазначити, що тут складається два рейтинги - один з них стосується платних програм, а інший - безкоштовних. Тому, якщо ви реєструвалися без кредитної картки, то сміливо можна дивитися рейтинг безкоштовних програм. Щоб побачити кожен рейтинг цілком, а не тільки представлені на екрані програми, то потрібно натиснути – See All.

Можна спочатку почитати описи для кожної програми, перш ніж купувати його і встановлювати на свій пристрій. Під картинкою програми буде відповідна кнопка. У вас буде також можливість залишити свій відгук на конкретний додаток і поставити йому оцінку. Деякі відгуки навіть здатні дати вам корисну інформацію про ту чи іншу гру або програму. І допоможуть знайти щось цікаве, щоб полегшити життя або скрасити дозвілля.

Вкладка Categories - тут всі програми розсортовані по категоріям. Спрощує вам пошук відповідного додатку. Дуже зручно.

Остання вкладка називається Updates. Якщо доступна нова версія програми, яка вже встановлено на вашому пристрої, то тут з'явиться нагадування. Якщо ви встановлювали тільки безкоштовні програми, то потрібно буде вручну сортувати список оновлюваних додатків.

Публікуємо додаток в AppStore

Щоб викладати програми в AppStore, необхідною умовою є наявність статусу iPhone Developer. Весь процес можна розділити на два етапи: підготовка і публікація. Підготовка (Provisioning). Найскладніший,на мою думку, етап. Він включає в себе створення ідентифікатора додатка App Id та сертифіката для підпису програми. Все це об'єднується під загальним поняттям профілю (Provisioning profle). Але все по порядку. Найскладніше у всьому цьому - створення сертифіката. Сертифікат необхідний для цифрового підпису програми. Існує два види сертифікатів - сертифікат розробника (Development certificate) та сертифікат дистрибуції (Distribution certificate). З назви не важко здогадатися, що перший тип потрібен для розробки, другий - для розповсюдження. Першим можна підписувати програми та запускати на своєму гаджеті (з метою налагодження), другий потрібен, щоб збирати додаток для публікації в аппсторе. Щоб мати можливість запускати додаток на своєму гаджеті, необхідно

зареєструвати пристрій у Provisioning Portal. Робиться це в такий спосіб. У середовищі Xcode відкриваємо Organizer (Window -> Organizer), підключаємо свій гаджет до комп'ютера (по USB, наприклад), бачимо, що він з'являється в розділі Devices. Копіюємо його DeviceID (довгий HEX-рядок) в буфер і йдемо в Provisioning Portal в розділ Devices. Там натискаєм Add Device, далі все просто

Створення сертифікатів. Основні кроки:

- Створення запиту на підпис сертифіката (включає в себе створення пари відкритого і приватного ключа). В результаті отримуємо на диску файл CSR.

- Сабміт запиту на підпис сертифіката. В Provisioning Portal заходимо в секцію Сертифікати, вказуємо файл CSR, тиснемо Submit, сертифікат створюється і переводиться в стан Pending Approval**.**

- В Provisioning Portal заходимо в секцію Certificates і підтверджуємо сертифікат (тиснемо Approve)

- Викачуємо і встановлюємо сертифікат в систему. В Provisioning Portal заходимо в секцію Certifiates. Тиснем на посилання WWDR Intermediate Certificate, на що нам пропонують зберегти файл .cer. Погоджуємося, і по завершенню скачування клікаєм на ньому. Запускається програма Keychain Access, яка пропонує встановити сертифікат в систему. Погоджуємося, і ось у нас є сертифікат для розробки.

Важливий етап - Apple настійно рекомендує зберегти згенерований приватний ключ. При його втраті всі праці виявляться марними, ми нічого більше не зможемо підписати. Добре, що я згадав це, так як сам до цих пір не забекапіть свій ключ. Для сертифіката дистрибуції все те ж саме, за винятком того, що ключі більше не треба генерувати.

Тепер потрібно створити Provisioning Profiles. Підготовчий профіль (назвемо його так) являє собою сукупність сертифікату та ідентифікатора додатка. Сертифікат у нас вже є, створимо ідентифікатор додатки. Для цього йдемо на Provisioning Portal в

секцію AppIDs. Якщо не брати до уваги «пакетну» концепцію ідентифікації додатків, то все просто: натискаємо Новий App ID, вводимо

- Description (напр, My Awesome App)
- Bundle Seed ID (App ID Prefix) - якщо це наше перше додаток, то в списку буде всього один пункт - Generate New, в іншому випадку в ньому будуть ідентифікатори раніше створених додатків (за задумом, додатки можна об'єднувати в пакети (suite))
- Bundle Identifier (App ID Suffix) - рекомендується вводити в так званому reverse domain style. Тобто домен в зворотному порядку. Тиснемо Sumbit. Все, ідентифікатор програми створено. Далі заходимо в розлом Provisioning і створюємо профілі нашого додатки (тиснемо New Profile). Для девелопмент-профілю вводимо
- Profile Name - під цим ім'ям профіль буде відображатися в xCode. Приклад:
- Certificates - ставимо мітку на нашому сертифікаті
- App ID - вибираємо наш додаток

- Devices - ставимо мітку на тих девайсах, які зареєстровані, їх може бути більше одного. Створили профіль, дочекалися його аппрува, завантажуємо файл профілю (. Provisionprofile), відкриваємо вікно Organizer і кидаємо туди драг-енд-дропом файл профілю. Він успішно інсталюється. Тепер відкриваємо Project Settings, шукаємо опцію Code Signing підрозділ Any iPhone OS, розгортаємо список, вибираємо свій профайл, закриваємо опції проекту. Далі важливий момент. У структурі проекту (в деревоподібному сайдбарі) розгортаємо гілку Targets, клацаємо правою кнопкою на Таргет, вибираємо Get Info. З'являються знову опції проекту, але вже з секцією Properties. Заходимо туди і вписуємо в поле Identifier те значення, яке ми вводили при створенні App ID в поле Bundle Identifier. Переконуємося, що девайс підключений, вибираємо в головному вікні xCode конфігурацію Device | Debug і тиснемо Build And Go.

Після того, як ми налагодили додаток на девайсі, настає пора збирати дистрибутив для публікації. Для цього повторюємо процес створення профілю дистрибуції. Він схожий з створенням девелопмент-профілю. Інсталюємо профіль в xCode, а далі потрібно здійснити наступне. Необхідно створити нову конфігурацію Distribution. Відкриваємо опції проекту, секцію Configuration, вибираємо там конфігурацію Release і натискаємо Duplicate. Вводимо назву Distribution. Далі у вікні Target в секції Build вибираємо настройки для конфігурації Distribution і в поле Code Signing Identity - Any iPhone OS Device вибираємо наш Distribution Profile. На закладці Properties не забуваємо вписати Bundle Identifier, якщо ще не зробили. Закриваємо опції, і в головному вікні xCode вибираємо конфігурацію Device | Distribution. Далі непогано б простежити за процесом складання. Для цього йдемо в меню Build - Build Results. Там налаштовуємо щоб показувався лог компіляції. Натискаємо Build. З'являється багато рядків, в кінці яких шукаємо слова

ProcessingProductPackaging ... embedded.mobileprovision і CodeSign. Пакет програми являє собою просту папку з розширенням. App, яка містить виконуваний файл і всі нутрощі програми (ресурси). Власне, додатки Mac OS X є рівно те ж саме, тому їх можна просто перетягнути куди-небудь в інше місце, і воно від туди запуститься, тому що містить всередині папки все необхідне. У цьому звичайно явна перевага non-registry концепції. І так, зіпуем наш застосунок тим самим готуємося до наступного етапу. Публікація:

Подальші дії проводяться на порталі iTunes Connect (itunesconnect.apple.com) Заходимо в розділ Manage Your Applications і тиснемо Add New Application. Вводимо

- App Name - ім'я програми, то, як воно буде виглядати в AppStore. При публікації другий додаток я зіткнувся з тим, що ця назва має бути унікальним для всього AppStore. Тобто, якщо ми задумали назвати додаток якимось чином, а додаток з такою назвою вже існує,

нам доведеться придумувати щось інше. Також слід врахувати вимоги самого Apple до назви програми http://www.apple.com/legal/trademark/guidelinesfor3rdparties.html

- SKU Number - унікальний ідентифікатор програми.
- Bundle ID вибираємо наш додаток

Далі ніби все зрозуміло, потрібно заповнити мета-інформацію про програму: опис, категорії, локалізацію, скріншоти, іконку. Після всього цього додаток переходить в стан Waiting for upload. Для завантаження бінарники потрібно програмка Application Loader з пакету Developer. Якщо такої немає, потрібно доустановити. Запускаємо її, вона запитує наш Apple ID і пароль, далі показує список програм, які очікують аплоаду. Вибираємо, завантажуємо. Після завантаження додаток переводиться в стан Waiting for Review - очікування розгляду. Розгляд відбувається фахівцями Apple на предмет відповідності вимогам. Ці вимоги описані в мануалі iPhone Application Development Guide, і включають в себе такі аспекти, як

відповідність загальному дизайну, грамотне використання ресурсів пристрою, таких як пам'яті, енергоспоживання, продуктивності ну і просто user-friendly. Apple дуже відповідально до цього підходить. Можуть загорнути, наприклад, якщо додаток дуже довго закривається, або є витік пам'яті. Щоб уникнути цих неприємностей, додаток перед складанням дистрибутива слід ретельно протестувати. У пакеті Developer є набір інструментів (додаток Instruments). За допомогою нього можна дізнатися багато нового про свій додаток. Наприклад, написавши перший додаток і прогнавши його через програму моніторингу виділення пам'яті, і витоків можна виявити, що є втечі пам’яті.

Висновки

Гра має важливе значення для людини. Безперечно, гра навчає і виховує. Вона є життєвою потребою багатьох. Комп'ютерні іграшки є не відмінною частиною дозвілля багатьох людей. Сучасна комп'ютерна гра – це багатофункціональна програма, яку використовують не тільки з розважальними, а й із навчальними та пропагандистськими цілями.

Зі всього вищесказаного можна зробити висновок, що комп'ютерна гра оцінюється двояко ,як така, що вчить, або як така, що розвиває концентрацію, увагу. Без сумніву,комп'ютерні ігри є квінтесенцією(багатозначністю) сучасних знань і технологій, проте найбільшу дію вони можуть надати не в духовній, не в інтелектуальній, а в емоційно-тілесній сфері.

Позитивний потенціал більшості комп'ютерних ігор реалізується далеко не завжди. І це залежить вирішальною мірою не від самої гри, а від людини, що грає, від того, який мотив переважає при включенні до гри. Крім основного мотиву розваги, гра може реалізувати інші мотиви. У залежності від мотивів

другого плану можуть формуватися абсолютно відмінні для різних гравців навички та вміння. Реалізація мотиву тренінгу призводить до формування навичок у сфері, що тренується, а мотив компенсації внутрішніх проблем скоріше матиме результатом формування механізмів психологічного захисту. У зв'язку з розмаїттям мотивів гра може сприяти як підготовці до зіткнення з реальністю, так і втечі від неї.

Гра заявлена в даній роботі має розвиваючий характер. Адже в ній гравець підвищує свою реакцію, за рахунок швидкої зміни цільових об'єктів. Також за рахунок цікавого дизайну, має більш привабливий сприйняття вигляд.

Метою було створити повноцінну iPhone гру, і розказати про засоби які необхідні розробнику.

На мою думку робота виконана відмінно.

Література

1. Програмирование для iPhone / Махер Али ; [пер. с англ.]. – М.: Эксмо, 2010. – 368 с.
2. More iPhone Cool Projects: Cool Developers Reveal the Details of Their Cooler Apps and Discuss Their iPad Development Experiences / Danton Chin, Claus Нцfele, Ben Kazez, Saul Mora, Leon Palm, Scott Penberthy, Ben Britten Smith, Chuck Smith, David Smith, Arne de Vries, and Joost van de Wijgerd; 2010, – 345 p.
3. Learn cocos2d Game Development with iOS 5 / Steffen Itterheim; 2011. – 677 p.
4. Learn iPhone and iPad cocos2d Game Development / Steffen Itterheim; 2010. - 397 p.
5. Objective-C 2.0 и програмирование под Mac / Далримпл, Марк, Киастер, Скотт.: Перевод с англ. – М.:ООО «И.Д.Вильямс», 2010. – 320 с.: ил. – Парал. тит. англ.

6. Начни программировать под Mac OS X используя Objective-C / Bert Altenberg, Alex Clarke, Philippe Mougin.: - 2008ю – 84 с.
7. iPhone SDK 3 Programming Advanced Mobile Development for Apple iPhone and iPod touch / Maher Ali; 2009, - 621 p.
8. Objective-C for Absolute Beginners: iPhone, iPad, and Mac Programming Made Easy / Gary Bennett, Mitch Fisher, Brad Lees; 2010. – 263 p.
9. iOS Programming: The Big Nerd Ranch Guide / Joe Conway, Aaron Hillegass; 2011. – 505 p.
10. iPhone Programming: The Big Nerd Ranch Guide / Joe Conway, Aaron Hillegass; 2010. – 464 p.
11. iPhone. Разработка приложений / Здзиарски Дж.: . 2009.: - 360 с.
12. Programming iOS 4 / Matt Neuburg; 2011.: - 789 p.
13. iOS 4 Programming Cookbook / Vandad Nahavandipoor; 2011.: - 595 p.
14. Beginning iPhone SDK Programming with Objective-C / Wiley Publishing, Inc. ; 2010.: - 499 p.

15. Xcode 4 iOS Development Beginner's Guide / Packt Publishing; 2011.: - 395 p.
16. Head First iPhone Development / Dan Pilone, Tracey Pilone; 2009.: - 560 p.
17. Building iPhone Apps with HTML, CSS, and JavaScript / Making App Store Apps Without Objective-C or Cocoa; Jonathan Stark; O'Reilly Media; 2010.: - 186 p.
18. Learning iPhone Programming / From Xcode to App Store; Alasdair Allan; O'Reilly Media; 2010.: - 384 p.
19. iOS 5 Programming Cookbook / Vandad Nahavandipoor; 2012.: - 876 p.
20. Concurrent Programming in Mac OS X and iOS / Vandad Nahavandipoor; 2011.: - 60 p.
21. Writing Game Center Apps in iOS / Vandad Nahavandipoor; 2011.: - 80 p.
22. Tap, Move, Shake - Turning Your Game Ideas into iPhone & iPad Apps / Todd Moore; 2011.: - 272 p.

23. Network Your Computers & Devices Step by Step / Ciprian Adrian Rusen, 7 Tutorials; Microsoft Press; 2010.; - 560 p.
24. Программируем для iPhone и iPad. 2-е изд. / Пайлон Д., Пайлон Т.; 2012.: - 624 p.
25. iPhone. Разработка приложений с открытым кодом. .: Пер. с англ. – 2-е изд., перераб. и доп. – СПб.: БХВ-Петербург, 2009. – 368 с.

Додатки

Код програми:

AppDelegate.h

```
#import <UIKit/UIKit.h>
#import "GKWizard.h"
@class RootViewController;


@interface AppDelegate : NSObject
<UIAlertViewDelegate,
UIApplicationDelegate,GKLeaderboardView
ControllerDelegate> {
    UIWindow         *window;
    RootViewController
    *viewController;
    BOOL
hasPlayedBefore,whacked3,whacked4;
    NSString *currentSkin;
    GKWizard *wiz;
    int
timesPlayed,currentAction,greenMolesWha
cked,blueMolesWhacked,yellowMolesWhacke
d;
}

@property (nonatomic, retain) UIWindow
*window;
```

```
-(void)finishedWithScore: (int)score;
-(int)getHighScore;
-(void)pause;
-(void)resume;
-(BOOL)isGameScene;
-(NSString *)getCurrentSkin;
-(void)setCurrentSkin:(NSString *)skin;
-(UIViewController *)getViewController;
-(void)showLeaderboard;

@end
```

AppDelegate.m

```
#import "cocos2d.h"

#import "AppDelegate.h"
#import "GameConfig.h"
#import "Game.h"
#import "RootViewController.h"
#import "MainMenu.h"
#import "SimpleAudioEngine.h"
#import "Constants.h"

@implementation AppDelegate

@synthesize window;
```

```
- (void) removeStartupFlicker
{
    //
    // THIS CODE REMOVES THE STARTUP
FLICKER
    //
    // Uncomment the following code if
you Application only supports landscape
mode
    //
#if GAME_AUTOROTATION ==
kGameAutorotationUIViewController

//  CC_ENABLE_DEFAULT_GL_STATES();
//  CCDirector *director = [CCDirector
sharedDirector];
//  CGSize size = [director winSize];
//  CCSprite *sprite = [CCSprite
spriteWithFile:@"Default.png"];
//  sprite.position = ccp(size.width/2,
size.height/2);
//  sprite.rotation = -90;
//  [sprite visit];
//  [[director openGLView]
swapBuffers];
//  CC_ENABLE_DEFAULT_GL_STATES();

#endif // GAME_AUTOROTATION ==
kGameAutorotationUIViewController
```

```
}
- (void)
applicationDidFinishLaunching:(UIApplic
ation*)application
{
    // Init the window
    window = [[UIWindow alloc]
initWithFrame:[[UIScreen mainScreen]
bounds]];
    timesPlayed = [[NSUserDefaults
standardUserDefaults]
integerForKey:kTimesPlayed];
    currentSkin = SKIN_MOLE;
    // Try to use CADisplayLink
director
    // if it fails (SDK < 3.1) use the
default director
    if( ! [CCDirector
setDirectorType:kCCDirectorTypeDisplayL
ink] )
        [CCDirector
setDirectorType:kCCDirectorTypeDefault]
;

    wiz = [[GKWizard alloc] init];
    CCDirector *director = [CCDirector
sharedDirector];

    // Init the View Controller
```

```
    viewController =
[[RootViewController alloc]
initWithNibName:nil bundle:nil];
    viewController.wantsFullScreenLayou
t = YES;

    //
    // Create the EAGLView manually
    //  1. Create a RGB565 format.
Alternative: RGBA8
    //  2. depth format of 0 bit. Use
16 or 24 bit for 3d effects, like
CCPageTurnTransition
    //
    //
    EAGLView *glView = [EAGLView
viewWithFrame:[window bounds]

pixelFormat:kEAGLColorFormatRGB565
    // kEAGLColorFormatRGBA8

depthFormat:0
    // GL_DEPTH_COMPONENT16_OES
                          ];

    // attach the openglView to the
director
    [director setOpenGLView:glView];
```

```
    //
    // VERY IMPORTANT:
    // If the rotation is going to be
controlled by a UIViewController
    // then the device orientation
should be "Portrait".
    //
    // IMPORTANT:
    // By default, this template only
supports Landscape orientations.
    // Edit the RootViewController.m
file to edit the supported
orientations.
    //
#if GAME_AUTOROTATION ==
kGameAutorotationUIViewController
    [director
setDeviceOrientation:kCCDeviceOrientati
onPortrait];
#else
    [director
setDeviceOrientation:kCCDeviceOrientati
onLandscapeRight];
#endif

    [director
setAnimationInterval:1.0/60];
```

```
    //[director setDisplayFPS:YES];

    [CCTexture2D
PVRImagesHavePremultipliedAlpha:YES];

    [viewController setView:glView];
    // make the OpenGLView a child of
the view controller


    //  // Enables High Res mode
(Retina Display) on iPhone 4 and
maintains low res on all other devices
    //[director
enableRetinaDisplay:YES]

    // make the View Controller a child
of the main window
    [window addSubview:
viewController.view];

    [window makeKeyAndVisible];

    // Default texture format for
PNG/BMP/TIFF/JPEG/GIF images
    // It can be RGBA8888, RGBA4444,
RGB5_A1, RGB565
    // You can change anytime.
```

```
    [CCTexture2D
setDefaultAlphaPixelFormat:kCCTexture2D
PixelFormat_RGBA4444];


    // Removes the startup flicker
    [self removeStartupFlicker];

    // Run the intro Scene
    [[CCDirector sharedDirector]
runWithScene: [MainMenu node]];
}


-
(void)applicationWillResignActive:(UIAp
plication *)application {
    if([self isGameScene])
    {
        [(Game *)[[CCDirector
sharedDirector] runningScene]
pauseGame];
    }
    else
    {
        [[CCDirector sharedDirector]
pause];
    }
```

```
}

-
(void)applicationDidBecomeActive:(UIApp
lication *)application {
    if (![self isGameScene]) {
        [[CCDirector sharedDirector]
resume];
    }

}

-
(void)applicationDidReceiveMemoryWarnin
g:(UIApplication *)application {
    [[CCDirector sharedDirector]
purgeCachedData];
}

-(void)
applicationDidEnterBackground:(UIApplic
ation*)application {
    [[CCDirector sharedDirector]
stopAnimation];
}

-(void)
applicationWillEnterForeground:(UIAppli
cation*)application {
```

```
    [[CCDirector sharedDirector]
startAnimation];
}

-
(void)applicationWillTerminate:(UIAppli
cation *)application {
    CCDirector *director = [CCDirector
sharedDirector];

    [[director openGLView]
removeFromSuperview];

    [viewController release];

    [window release];

    [director end];
}

-
(void)applicationSignificantTimeChange:
(UIApplication *)application {
    [[CCDirector sharedDirector]
setNextDeltaTimeZero:YES];
}
```

```
-(void)finishedWithScore: (int)score
{
    /*if(score > [self getHighScore])
    {
        [[NSUserDefaults
standardUserDefaults] setInteger:score
forKey:kHighScoreKey];
        [wiz reportScore:score
forLeaderboard:kHighScoreKey];
    }
    timesPlayed++;
    [[NSUserDefaults
standardUserDefaults]
setInteger:timesPlayed
forKey:kTimesPlayed];
    if (timesPlayed % 10 == 0 &&
![[NSUserDefaults standardUserDefaults]
boolForKey:kDidRate]) {
        UIAlertView *alert =
[[UIAlertView alloc]
initWithTitle:@"Like Mole It?"
message:@"If you like Mole It, please
rate it to show your support."
delegate:self
cancelButtonTitle:@"Cancel"
otherButtonTitles:@"Rate", nil];
        [alert show];
    }*/
}
```

```
-(int)getHighScore
{
    return [[NSUserDefaults
standardUserDefaults]
integerForKey:kHighScoreKey];
}
#pragma mark -------

-(void)pause
{
    if (![self isGameScene]) {
        [[CCDirector sharedDirector]
pause];
    }
}
-(void)resume
{
    if (![self isGameScene]) {
        [[CCDirector sharedDirector]
resume];
    }
}


-(BOOL)isGameScene
{
```

```
    return [[[CCDirector
sharedDirector] runningScene]
isKindOfClass:[Game class]];
}


-(NSString *)getCurrentSkin
{
    return currentSkin;
}
-(void)setCurrentSkin:(NSString *)skin
{
    currentSkin = skin;
}

-(UIViewController *)getViewController
{
    return viewController;
}


-(void)alertView:(UIAlertView
*)alertView
clickedButtonAtIndex:(NSInteger)buttonI
ndex
{
    if(buttonIndex == 0)
    {
        return;
```

```
    }
    [[UIApplication sharedApplication]
openURL: [NSURL
URLWithString:@"http://itunes.apple.com
/us/app/mole-it!-
free/id464362476?ls=1&mt=8"]];
    [[NSUserDefaults
standardUserDefaults] setBool:YES
forKey:kDidRate];

}

-(void)showLeaderboard
{
    GKLeaderboardViewController *lb =
[[[GKLeaderboardViewController alloc]
init] autorelease];
    lb.leaderboardDelegate = self;
    [viewController
presentModalViewController:lb
animated:YES];
}

-
(void)leaderboardViewControllerDidFinis
h:(GKLeaderboardViewController
*)viewController
{
```

```
    [viewController
dismissModalViewControllerAnimated:YES]
;
}

- (void)dealloc {
    [[CCDirector sharedDirector] end];
    [window release];
    [wiz release];
    [super dealloc];
}

@end
```

MainMenu.h

```
#import "cocos2d.h"

@class AppDelegate;

@interface MainMenu : CCScene {
    AppDelegate *delegate;

}

-(void)playGame;

@end
```

MainMenu.m

```
#import "MainMenu.h"
```

```
#import "Game.h"
#import "AppDelegate.h"
#import "SimpleAudioEngine.h"
#import "Constants.h"
#import "PopUp.h"
#import "GameButton.h"
#import "CCMenuPopup.h"

@implementation MainMenu

- (id)init
{
    self = [super init];
    if (self) {
        // Initialization code here.
        CGSize s = [[CCDirector
sharedDirector] winSize];

        [[SimpleAudioEngine
sharedEngine] stopBackgroundMusic];
        delegate = (AppDelegate
*)[[UIApplication sharedApplication]
delegate];

        NSString *fileName =  [NSString
stringWithFormat: @"%@.plist",
[delegate getCurrentSkin]];
```

```
        CCLabelTTF *autor = [CCLabelTTF
labelWithString:@"РњР°РєСЃРёРјС‡СѓРєР
РЎРµРіРёРЅ" fontName:@"Arial"
fontSize:18];
        [self addChild:autor];
        autor.position =
ccp(s.width/6,s.height/32);

        [[CCSpriteFrameCache
sharedSpriteFrameCache]
addSpriteFramesWithFile:fileName];
        int fSize = 24;
        CCLabelTTF *highScore =
[CCLabelTTF labelWithString:[NSString
stringWithFormat:@"High Score: %i",
[delegate getHighScore]]
fontName:@"Arial" fontSize:fSize];
        highScore.anchorPoint =
ccp(1,1);
        highScore.color = ccBLACK;
        highScore.position =
ccp(s.width,s.height);
        [self addChild:highScore];
        [CCMenuItemFont
setFontName:@"TOONISH.ttf"];//@"TOONISH
.ttf"];
        fSize = [CCMenuItemFont
fontSize];
```

```
        [CCMenuItemFont
setFontSize:48];

        CCMenuItemSprite *playButton =
[CCMenuItemSprite
itemFromNormalSprite:[GameButton
buttonWithText:@"PLAYP°Pє" isBig:YES]
selectedSprite:NULL target:self
selector:@selector(playGame)];

        [CCMenuItemFont
setFontSize:fSize/1.5];
        //CCMenuItemSprite
*leaderboardsButton = [CCMenuItemSprite
itemFromNormalSprite:[GameButton
buttonWithText:@"Game Center1"]
selectedSprite:NULL target:self
selector:@selector(showLeaderboard)];
        CCMenuItemSprite
*selectSkinButton = [CCMenuItemSprite
itemFromNormalSprite:[GameButton
buttonWithText:@"SKINS"]
selectedSprite:NULL target:self
selector:@selector(selectSkin)];
        //CCMenuItemSprite
*otherGamesButton = [CCMenuItemSprite
itemFromNormalSprite:[GameButton
buttonWithText:@"more games1"]
```

```
selectedSprite:NULL target:self
selector:@selector(otherGames)];
        //[CCMenuItemFont
setFontSize:fSize];
        CCMenu *menu = [CCMenu
menuWithItems:
/*leaderboardsButton,otherGamesButton,*
/ selectSkinButton, nil];

        [menu
alignItemsHorizontallyWithPadding:20];
        menu.position = ccp(s.width/2,
45);
        [self addChild:menu];

        CCMenu *mainPlay = [CCMenu
menuWithItems:playButton, nil];
        mainPlay.position =
ccp(s.width/2,s.height/2 +
s.height/3.5f);
        [self addChild:mainPlay];

        fileName = @"title123.png";

        [CCTexture2D
setDefaultAlphaPixelFormat:kCCTexture2D
PixelFormat_RGB565];
        CCSprite *bg = [CCSprite
spriteWithFile:fileName];
```

```
        bg.anchorPoint = ccp(0,0);
        [self addChild:bg z:-1];
        [CCTexture2D
setDefaultAlphaPixelFormat:kCCTexture2D
PixelFormat_RGBA4444];
    }

    return self;
}

-(void)playGame
{
    [[CCDirector sharedDirector]
replaceScene:[Game node]];
}


-(void)selectSkin
{
    CCMenuItemSprite *jetIcon =
[CCMenuItemSprite
itemFromNormalSprite:[CCSprite
spriteWithFile:@"jetpack_icon.png"]
selectedSprite:NULL target:self
selector:@selector(jSkin)];
    CCMenuItemSprite *moleIcon =
[CCMenuItemSprite
itemFromNormalSprite:[CCSprite
spriteWithFile:@"moles_icon.png"]
```

```
selectedSprite:NULL target:self
selector:@selector(mSkin)];

    CCMenuPopup *menu = [CCMenuPopup
menuWithItems:jetIcon,moleIcon, nil];
    [menu
alignItemsHorizontallyWithPadding:20];

    CCMenuItemSprite *btnCancel =
[CCMenuItemSprite
itemFromNormalSprite:[GameButton
buttonWithText:@"Cancel"]
selectedSprite:NULL];

    CGSize s = [[CCDirector
sharedDirector] winSize];

    CCMenuPopup *cancelMenu =
[CCMenuPopup menuWithItems:btnCancel,
nil];
    btnCancel.position = ccp(0,-
s.height/3);
    CCSprite *container = [CCSprite
node];
    [container addChild:menu];
    [container addChild:cancelMenu];

    PopUp *pop = [PopUp
popUpWithTitle:@"-SKINS!-"
```

```
description:@"Select a skin to use or
preview" sprite:container];
    [self addChild:pop z:1000];
}

-(void)otherGames
{

    NSString *search = @"wedgekase
games";
    NSString *sstring = [NSString
stringWithFormat:@"http://phobos.apple.
com/WebObjects/MZSearch.woa/wa/search?W
OURLEncoding=ISO8859_1&lang=1&output=lm
&country=US&term=%@&media=software",
[search
stringByAddingPercentEscapesUsingEncodi
ng:NSUTF8StringEncoding]];
    [[UIApplication sharedApplication]
openURL: [NSURL
URLWithString:sstring]];
}

-(void)jSkin
{
    [delegate
setCurrentSkin:SKIN_JETPACK];
}
```

```
-(void)mSkin
{
    [delegate
setCurrentSkin:SKIN_MOLE];
}

-(void)showLeaderboard
{
    [delegate showLeaderboard];
}

- (void)dealloc
{
    CCLOG(@"dealloc: %@", self);
    [super dealloc];
}

@end
```

GameButton.h

```
#import "cocos2d.h"


@interface GameButton: CCSprite
{

}

+(id)buttonWithText: (NSString
*)buttonText isBig:(BOOL)big;
```

```
+(id)buttonWithText: (NSString
*)buttonText;


-(id)initWithText: (NSString
*)buttonText isBig:(BOOL)big;


@end
```

GameButton.m

```
#import "GameButton.h"

@implementation GameButton

+(id)buttonWithText: (NSString
*)buttonText isBig:(BOOL)big
{
    return [[[GameButton alloc]
initWithText:buttonText isBig:big]
autorelease];
}

+(id)buttonWithText: (NSString
*)buttonText
{
    return [[[GameButton alloc]
initWithText:buttonText isBig:NO]
autorelease];
}
```

```
-(id)initWithText: (NSString
*)buttonText isBig:(BOOL)big
{
    if((self == [super init]))
    {
        NSString *btnFrame = (big) ?
@"button_big.png" :
@"button_small.png";
        int fSize = 12;
        [self
setDisplayFrame:[[CCSpriteFrameCache
sharedSpriteFrameCache]
spriteFrameByName:btnFrame]];
        CCLabelTTF *label = [CCLabelTTF
labelWithString:buttonText
fontName:@"TOONISH" fontSize:fSize +
big * fSize];
        label.position =
ccp(self.contentSize.width/2,
self.contentSize.height/2);
        [self addChild:label z:1];
        CCLabelTTF *labelShadow =
[CCLabelTTF labelWithString:buttonText
fontName:@"TOONISH" fontSize:fSize +
big * fSize];
        labelShadow.position =
ccp(self.contentSize.width/2 - (2 + big
* 2), self.contentSize.height/2);
```

```
        labelShadow.color = ccBLACK;
        labelShadow.opacity = 150;
        [self addChild:labelShadow];
    }
    return self;
}

@end
```

Game.h

```
#import "cocos2d.h"

@class AppDelegate,Mole;

@interface Game : CCScene
<CCStandardTouchDelegate> {
    CCArray *moles, *carrots;
    float timeBetweenMoles,timeElapsed,
increaseMolesAtTime, increaseElapsed,
lastMoleHitTime,totalTime;
    CCLabelBMFont *scoreLabel;
    int carrotsLeft, score,
molesAtOnce;
    CGSize s;
    AppDelegate *delegate;
    CCMenuItemSprite *pauseButton;
    NSString *nextMoleType, *missSound,
*hitSound;
```

```
    bool
canShowYellowMoles,canShowBlueMoles,isP
aused;
}

-(void)showMole;
-(void)didScore;
-(void)missedMole;
-(int)getMolesUp;
-(void)pauseGame;
-(void)resumeGame;
-(NSArray *)getUpMoles;
-(NSArray *)getDownMoles;
-(void)startGame;
-(void)initializeGame;
-(void)chooseWhichMoleToMake;
-(void)mainMenu;
-(void)playAgain;
-(void)gameOver;

@end
```

Game.m

```
#import "Game.h"
#import "Mole.h"
#import "MainMenu.h"
#import "AppDelegate.h"
#import "SimpleAudioEngine.h"
#import "Constants.h"
#import "PopUp.h"
```

```
#import "GameButton.h"
#import "CCMenuPopup.h"

@implementation Game

- (id)init
{
    self = [super init];
    if (self) {
        carrotsLeft = 3;
        molesAtOnce = 3;
        timeBetweenMoles = 0.5f;
        increaseMolesAtTime = 10.0f;
    }

    return self;
}

-(void)initializeGame
{
    delegate = (AppDelegate
*)[[UIApplication sharedApplication]
delegate];
    [[CCDirector sharedDirector]
resume];
    s = [[CCDirector sharedDirector]
winSize];
```

```
    missSound = [NSString
stringWithFormat:@"%@_miss.wav",
[delegate getCurrentSkin]];
    hitSound = [NSString
stringWithFormat:@"%@_ouch.wav",
[delegate getCurrentSkin]];

    [[SimpleAudioEngine sharedEngine]
preloadEffect:missSound];
    [[SimpleAudioEngine sharedEngine]
preloadEffect:hitSound];
    [[SimpleAudioEngine sharedEngine]
preloadEffect:@"splat.wav"];
    [[SimpleAudioEngine sharedEngine]
preloadBackgroundMusic:[NSString
stringWithFormat:@"%@_bg.mp3",
[delegate getCurrentSkin]]];

    NSString *fileName = [NSString
stringWithFormat:@"%@.plist", [delegate
getCurrentSkin]];

    [[CCSpriteFrameCache
sharedSpriteFrameCache]
addSpriteFramesWithFile:fileName];

    int fSize = 24;
    scoreLabel = [CCLabelBMFont
labelWithString:@"score:0"
```

```
fntFile:[NSString
stringWithFormat:@"%i.fnt", fSize]];
    scoreLabel.anchorPoint = ccp(0,1);
    scoreLabel.position =
ccp(0,s.height);
    [self addChild:scoreLabel z:10];

    carrots = [[CCArray alloc] init];
    moles = [[CCArray alloc] init];

    float hPad = 20;
    float vPad = 25;

    for (int i = 1; i <= 4; i++) {
        for (int j = 1; j <= 6; j++) {
            Mole *mole = [Mole
spriteWithSpriteFrameName:@"a0001.png"]
;
            mole.position = ccp(j *
mole.contentSize.width + hPad, i *
mole.contentSize.height + vPad);
            [moles addObject:mole];
            [self addChild:mole z:1];
        }
    }

    [CCTexture2D
setDefaultAlphaPixelFormat:kCCTexture2D
PixelFormat_RGB565];
```

```
    CCSprite *bg = [CCSprite
spriteWithFile:[NSString
stringWithFormat:@"%@_bg.png",
[delegate getCurrentSkin]]];
    bg.anchorPoint = ccp(0,0);
    [self addChild:bg z:-1];
    [CCTexture2D
setDefaultAlphaPixelFormat:kCCTexture2D
PixelFormat_RGBA4444];

    for (int i = 0; i < carrotsLeft;
i++) {
        CCSprite *c = [CCSprite
spriteWithSpriteFrameName:@"life.png"];
        c.anchorPoint = ccp(1,1);
        c.position = ccp(s.width - i *
c.contentSize.width, s.height);
        [carrots addObject:c];
        [self addChild:c z:10];
    }

    pauseButton = [CCMenuItemSprite
itemFromNormalSprite:[CCSprite
spriteWithSpriteFrameName:@"pause_butto
n.png"] selectedSprite:NULL target:self
selector:@selector(pauseGame)];
    CCMenu *menu = [CCMenu
menuWithItems:pauseButton, nil];
```

```
    pauseButton.position =
ccp(s.width/2 -
pauseButton.contentSize.width/2, (-
s.height/2) +
pauseButton.contentSize.height/2);
    [self addChild:menu z:100];

    [self startGame];
}

-(void)startGame
{
    [[SimpleAudioEngine sharedEngine]
playBackgroundMusic:[NSString
stringWithFormat:@"%@_bg.mp3",
[delegate getCurrentSkin]] loop:YES];
    [self schedule:@selector(tick:)];
}

-(void)chooseWhichMoleToMake
{
    if ([self getMolesUp] >=
molesAtOnce) {
        return;
    }
    nextMoleType = (CCRANDOM_0_1() <
.15 && canShowBlueMoles) ? MOLE_TYPE_B
: MOLE_TYPE_A;
```

```
    if ([self getMolesUp] < molesAtOnce
- 1 && CCRANDOM_0_1() < .1 &&
canShowYellowMoles) {
        nextMoleType = MOLE_TYPE_C;
        [self showMole];
    }

    [self showMole];
}

-(void)tick: (ccTime)dt
{
    timeElapsed += dt;
    increaseElapsed += dt;
    if(timeElapsed >= timeBetweenMoles)
    {
        [self chooseWhichMoleToMake];
        timeElapsed = 0;
    }
    if (increaseElapsed >=
increaseMolesAtTime) {
        int maxMolesAtOnce = 18;
        if (molesAtOnce <
maxMolesAtOnce) {
            molesAtOnce++;
            float minMoleTime = .1f;
            timeBetweenMoles -=
(timeBetweenMoles > minMoleTime) ?
0.05f : 0;
```

```
            increaseMolesAtTime +=
10.0f;
        }
        if (canShowBlueMoles) {
            canShowYellowMoles = YES;
        }
        else
        {
            canShowBlueMoles = YES;
        }
    }


}

-(void)showMole
{
    Mole *mole = [[CCArray
arrayWithNSArray:[self getDownMoles]]
randomObject];
    [mole startWithType:nextMoleType];
}

-(NSArray *)getDownMoles
{
    NSMutableArray *downMoles =
[[NSMutableArray alloc] init];
    for (Mole *m in moles) {
        if (![m getIsUp]) {
```

```
            [downMoles addObject:m];
        }
    }
    return downMoles;
}
-(NSArray *)getUpMoles
{
    NSMutableArray *upMoles =
[[NSMutableArray alloc] init];
    for (Mole *m in moles) {
        if ([m getIsUp]) {
            [upMoles addObject:m];
        }
    }
    return upMoles;
}



-(void)ccTouchesBegan:(NSSet *)touches
withEvent:(UIEvent *)event
{
    if (isPaused) {
        return;
    }
    NSMutableArray *molesTappedAtOnce =
[[NSMutableArray alloc] init];
    for (UITouch *touch in [event
allTouches]) {
```

```
        for (Mole *mole in moles) {
            CGPoint location = [touch
locationInView:touch.view];
            location = [[CCDirector
sharedDirector] convertToGL:location];
            if
(CGRectContainsPoint([mole
boundingBox], location)) {
                if(![mole getIsUp] ||
[molesTappedAtOnce
containsObject:mole])
                {
                    continue;
                }
                if([[mole getType]
isEqualToString:MOLE_TYPE_C])
                {
                    [molesTappedAtOnce
addObject:mole];
                }

                bool
greenMoleWasWhacked = ([[mole getType]
isEqualToString:MOLE_TYPE_A]);
                bool blueMoleWasWhacked
= ([[mole getType]
isEqualToString:MOLE_TYPE_B] && [touch
tapCount] > 1);
```

```
                if(greenMoleWasWhacked
|| blueMoleWasWhacked)
                {
                    [mole wasTapped];
                    [self didScore];
                    [[SimpleAudioEngine
sharedEngine] playEffect:hitSound
pitch:1 pan:1 gain:.25];
                    [[SimpleAudioEngine
sharedEngine] playEffect:@"splat.wav"];
                }

            }
        }
    }

    if ([molesTappedAtOnce count] > 1)
{
        for (Mole *m in
molesTappedAtOnce) {
            [m wasTapped];
            [self didScore];
        }
        [[SimpleAudioEngine
sharedEngine] playEffect:hitSound
pitch:1 pan:1 gain:.25];
        [[SimpleAudioEngine
sharedEngine] playEffect:@"splat.wav"];
    }
```

```
    [molesTappedAtOnce
removeAllObjects];
}

-(void)didScore
{
    score++;
    [scoreLabel setString:[NSString
stringWithFormat:@"score:%i",score]];
}


-(void)missedMole
{
    carrotsLeft--;
    [[SimpleAudioEngine sharedEngine]
playEffect:missSound pitch:1 pan:1
gain:.2];
    if([carrots count] > 0)
    {
        [self removeChild:[carrots
objectAtIndex:[carrots count] - 1]
cleanup:YES];
        [carrots removeLastObject];
    }
    if(carrotsLeft <= 0)
    {
        [self gameOver];
    }
```

```
    else
    {
        for (Mole *m in [self
getUpMoles]) {
            [m stopEarly];
        }
    }
}

-(void)gameOver
{
    for (Mole *m in moles) {
        [m stopAllActions];
        [m unscheduleAllSelectors];
    }
    [delegate finishedWithScore:score];
    [self unscheduleAllSelectors];
    [[SimpleAudioEngine sharedEngine]
stopBackgroundMusic];

    CCMenuItemSprite *playAgainButton =
[CCMenuItemSprite
itemFromNormalSprite:[GameButton
buttonWithText:@"play again"]
selectedSprite:NULL target:self
selector:@selector(playAgain)];
    CCMenuItemSprite *mainButton =
[CCMenuItemSprite
itemFromNormalSprite:[GameButton
```

```
buttonWithText:@"main menu"]
selectedSprite:NULL target:self
selector:@selector(mainMenu)];
    CCMenuPopup *menu = [CCMenuPopup
menuWithItems:playAgainButton,mainButto
n, nil];
    [menu
alignItemsHorizontallyWithPadding:10];
    PopUp *pop = [PopUp
popUpWithTitle:@"-game over-"
description:@"" sprite:menu];
    [self addChild:pop z:1000];

}

-(int)getMolesUp
{
    int upMoles = 0;
    for (Mole *m in moles) {
        if ([m getIsUp]) {
            upMoles++;
        }
    }
    return upMoles;
}

-(void)pauseGame
{
    if(isPaused)
```

```
    {
        return;
    }
    CCMenuItemSprite *resumeButton =
[CCMenuItemSprite
itemFromNormalSprite:[GameButton
buttonWithText:@"resume"]
selectedSprite:NULL target:self
selector:@selector(resumeGame)];
    CCMenuItemSprite *mainButton =
[CCMenuItemSprite
itemFromNormalSprite:[GameButton
buttonWithText:@"main menu"]
selectedSprite:NULL target:self
selector:@selector(mainMenu)];
    CCMenuPopup *menu = [CCMenuPopup
menuWithItems:resumeButton,mainButton,
nil];
    [menu
alignItemsHorizontallyWithPadding:10];
    PopUp *pop = [PopUp
popUpWithTitle:@"-pause-"
description:@"" sprite:menu];
    [self addChild:pop z:1000];
    pauseButton.visible = NO;
    for (Mole *m in [self getUpMoles])
{
        [m stopEarly];
    }
```

```
    [self unschedule:@selector(tick:)];
    [[SimpleAudioEngine sharedEngine]
pauseBackgroundMusic];
    isPaused = YES;
}
-(void)resumeGame
{
    pauseButton.visible = YES;
    [self schedule:@selector(tick:)];
    [[SimpleAudioEngine sharedEngine]
resumeBackgroundMusic];
    isPaused = NO;
}

-(void)mainMenu
{
    [[CCDirector sharedDirector]
resume];
    [[CCDirector sharedDirector]
replaceScene:[MainMenu node]];
}

-(void)playAgain
{
    [[CCDirector sharedDirector]
resume];
    [[CCDirector sharedDirector]
replaceScene:[[self class] node]];
}
```

```
-(void)onEnterTransitionDidFinish
{
    [[CCTouchDispatcher
sharedDispatcher]
addStandardDelegate:self priority:0];
    [[[CCDirector sharedDirector]
openGLView]
setMultipleTouchEnabled:YES];
    [self initializeGame];
}

-(void)onExit
{
    [carrots release];
    [moles release];
    [[SimpleAudioEngine sharedEngine]
unloadEffect:hitSound];
    [[SimpleAudioEngine sharedEngine]
unloadEffect:missSound];
    [[SimpleAudioEngine sharedEngine]
unloadEffect:@"splat.wav"];
    [[CCTouchDispatcher
sharedDispatcher] removeAllDelegates];
}

- (void)dealloc
{
    CCLOG(@"dealloc: %@", self);
```

```
    [super dealloc];
}

@end
```

CCMenuPopup.h

```
#import "cocos2d.h"


@interface CCMenuPopup : CCMenu {

}

@end
```

CCMenuPopup.m

```
#import "CCMenuPopup.h"
#import "PopUp.h"

@implementation CCMenuPopup

-(void)registerWithTouchDispatcher
{
    [[CCTouchDispatcher
sharedDispatcher]
addTargetedDelegate:self priority:-1001
swallowsTouches:YES];
}

-(BOOL)ccTouchBegan:(UITouch *)touch
withEvent:(UIEvent *)event
```

```
{
    if (![super itemForTouch:touch]) {
        return NO;
    }
    NSArray *ancestors = [NSArray
arrayWithObjects:self.parent,self.paren
t.parent,self.parent.parent.parent,
nil];
    for (CCNode *n in ancestors) {
        if ([n isKindOfClass:[PopUp
class]]) {
            [(PopUp *)n closePopUp];
        }
    }

    return [super ccTouchBegan:touch
withEvent:event];
}

@end
```

GKWizard.h

```
#import <Foundation/Foundation.h>
#import <GameKit/GameKit.h>

@interface GKWizard : NSObject {
    BOOL isLoggedinToGC;
    int highScore;
}
```

```
-(void)reportScore: (int)score
forLeaderboard:(NSString *)leaderboard;
-(bool) isGameCenterAvailable;
- (void) authenticateLocalPlayer;
-(int)getHighScore;

@end
```

GKWizard.m

```
#import "GKWizard.h"
#import "Constants.h"
#import "cocos2d.h"

@implementation GKWizard

- (id)init
{
    self = [super init];
    if (self) {
        // Initialization code here.
        highScore = [[NSUserDefaults
standardUserDefaults]
integerForKey:kHighScoreKey];
        [self authenticateLocalPlayer];
    }

    return self;
}
```

```
-(void)reportScore: (int)score
forLeaderboard:(NSString *)leaderboard
{
    if (![self isGameCenterAvailable])
{
        return;
    }
    GKScore *scoreReporter = [[[GKScore
alloc] initWithCategory:leaderboard]
autorelease];
    if (scoreReporter.value < score) {
        scoreReporter.value = score;
    }
    else
    {
        return;
    }


    [scoreReporter
reportScoreWithCompletionHandler:^(NSEr
ror *error) {
        if (error != nil)
        {
            // handle the reporting
error
        }
    }];
}
```

```
- (void) authenticateLocalPlayer
{
    /*if (![self
isGameCenterAvailable]) {
        UIAlertView *alert =
[[UIAlertView alloc]
initWithTitle:@"Feature Not Available"
message:@"Game Center features require
supported devices/operating systems."
delegate:NULL cancelButtonTitle:@"OK"
otherButtonTitles:nil];
        [alert show];
        [alert release];
    }
    [[GKLocalPlayer localPlayer]
authenticateWithCompletionHandler:^(NSE
rror *error) {
        if (error == nil)
        {
            // Insert code here to
handle a successful authentication.

            isLoggedinToGC = YES;
        }
        else
        {
```

```
            // Your application can
process the error parameter to report
the error to the player.
            //NSLog(@"error logging
in");
        }
    }];*/
}

- (void) reportScore: (int64_t) score
forCategory: (NSString*) category
{

}

-(bool) isGameCenterAvailable
{
    // Check for presence of
GKLocalPlayer API.
    Class gcClass =
(NSClassFromString(@"GKLocalPlayer"));

    // The device must be running
running iOS 4.1 or later.
    bool isIPAD = [[UIDevice
currentDevice] userInterfaceIdiom] ==
UIUserInterfaceIdiomPad;
    NSString *reqSysVer = (isIPAD) ?
@"4.2" : @"4.1";
```

```
    NSString *currSysVer = [[UIDevice
currentDevice] systemVersion];
    BOOL osVersionSupported =
([currSysVer compare:reqSysVer
options:NSNumericSearch] !=
NSOrderedAscending);

    return (gcClass &&
osVersionSupported);
}

-(int)getHighScore
{
    return highScore;
}

- (void)dealloc
{
    [super dealloc];
}

@end
```

Сергій Валентинович Максимчук
магістрант інформаційних технологій

Розробка гри під платформу iOS для iPHONE

ІН 21М

Комп'ютерний набір, верстка і макетування та дизайн в редакторі Microsoft®Office® Word 2003 С.В. Максимчук
Науковий керівник: Р.М.Літнарович, канд. техн. наук, доцент

Міжнародний Економіко-Гуманітарний Університет ім. акад. Степана Дем'янчука
Факультет Кібернетики
Кафедра математичного моделювання
33027,м.Рівне,Україна
Вул.акад. С.Дем'янчука,4, корпус 1
Телефон:(+00380) 362 23-73-09
Факс:(+00380) 362 23-01-86
E-mail:mail@regi.rovno.ua
E-mail: litnarovich@windowslive.com