Co-funded by the
Erasmus+ Programme
of the European Union

GAMEHUB University Enterprises Cooperation In Game Industry In Ukraine

ПРОЕКТ ЕРАЗМУС+
GAMEHUB: «СПІВРОБІТНИЦТВО МІЖ УНІВЕРСИТЕТАМИ
ТА ПІДПРИЄМСТВАМИ В СФЕРІ ГРАЛЬНОЇ ІНДУСТРІЇ В УКРАЇНІ»

ERASMUS+ PROJECT
GAMEHUB: «UNIVERSITY-ENTERPRISES COOPERATION
IN GAME INDUSTRY IN UKRAINE»

ТЕХНОЛОГІЇ РОЗРОБКИ КОМП'ЮТЕРНИХ ІГОР

Co-funded by the
Erasmus+ Programme
of the European Union

GAMEHUB: «СПІВРОБІТНИЦТВО МІЖ УНІВЕРСИТЕТАМИ
ТА ПІДПРИЄМСТВАМИ В СФЕРІ ГРАЛЬНОЇ ІНДУСТРІЇ В УКРАЇНІ»

ТЕХНОЛОГІЇ РОЗРОБКИ КОМП'ЮТЕРНИХ ІГОР

Харків
«Друкарня Мадрид»
2018

УДК 004.9
ББК 32.973-018.2
в6
Б 87

Проект ЕРАЗМУС+
GameHub: «Співробітництво між університетами та підприємствами
в сфері гральної індустрії в Україні»
№ 561728-EPP-1-2015-1- ES-EPPKA2-CBHE-JP

Бреславець В.С.

Б 87 Технології розробки комп'ютерних ігор: довідник модуля. / В.С. Бреславець. - Х.: «Друкарня Мадрид», 2018. - 162 с.
ISBN 978-617-7683-04-8

Довідник модуля «Технології розробки комп'ютерних ігор» написаний згідно з останніми освітньо — кваліфікаційними вимогами до підготовки бакалаврів зі спеціальності «Інформаційні технології». З урахуванням кредитно-модульної системи навчальна дисципліна формується як система змістовних модулів. Довідник модуля складається з інформації стосовно модуля та критеріїв оцінювання, лекційних та лабораторних робіт.
Призначено для студентів всіх спеціальностей вузів, викладачів курсу «Технології розробки комп'ютерних ігор», фахівців у галузі «Інформаційні технології», які підвищують кваліфікацію.

Бібліографічні описи здійснено відповідно до існуючих державних та міждержавних стандартів: ДСТУ ГОСТ 7.1:2006 «Бібліографічний запис. Бібліографічний опис. Загальні вимоги та правила складання».

The "Technology of Computer Games Development" module manual is written in accordance with the latest educational and qualification requirements for the preparation of bachelors in the "Information Technologies" specialty. Taking into account the credit-module system, the discipline is formed as a system of content modules. The module manual consists of information on the module and criteria for evaluation, lecture and laboratory work.
It is intended for students of all specialties of higher educational establishments, lecturers of the "Technologies of development of computer games" course, specialists in the field of "Information technologies", who improve qualification.

This publication has been funded with support from the European Union. The publication reflects the views only of the authors, and the Union cannot be held responsible for any use which may be made of the information contained therein.

УДК 004.9
ББК 32.973-018.2

ISBN 978-617-7683-04-8

ЗМІСТ

ДОВІДНИК МОДУЛЯ

ЛЕКЦІЇ

ЛАБОРАТОРНІ РОБОТИ

Довідник модуля

1 ВСТУП

Предмет передбачає ознайомити студентів з основними концептуальними поняттями, що використовуються при реалізації підходів до проектування комп'ютерних ігор, познайомити з технологією розробки комп'ютерних ігор, з програмним забезпеченням що застосовується для розробки окремих модулів комп'ютерних ігор, також досліджуються інфологічні, даталогічні та фізичні моделі, які використовуються в комп'ютерних іграх та інших технічних системах.

Мета дисципліни

Метою викладання навчальної дисципліни «Проектування інформаційних систем» є наступне:

- освоїти майбутнім фахівцям в розробці комп'ютерних ігор теоретичні знання і сформувати у них практичні навички в застосуванні інформаційних систем для вирішення завдань розробки різноманітних типів ігор;

- створення у студентів впорядкованої системи знань про реальні можливості систем розробки ігор, їх типах, архітектурі, складові частини, методах і засобах проектування систем розробки ігор, основних технологічних підходах до проектування;

- формування бази для прийняття рішення про оцінку необхідності і доцільності впровадження тих чи інших систем розробки в практику;

- ознайомлення студентів з практикою застосування новітніх інфор-маційних технологій в області проектування сучасних комп'ютерних ігор, застосування сучасних методів і засобів проектування, заснованих на використанні CASE-технології, а також навичок самостійного практичного проектування ігор для різних предметних областей.

Очікувані результати

Основними завданнями вивчення дисципліни «Проектування інформаційних систем» є оволодіння

- здатністю розробляти, втілювати та адаптувати прикладне програмне забезпечення;

- здатністю проектувати ІС в відповідності із профілем підготовки по видах забезпечення;

- здатністю програмувати додатки та створювати програмні прототипи рішення прикладних задач;

- здатністю проводити тестування компонентів програмного забезпечення ІС;

- здатністю виконувати тестування компонентів інформаційних систем за заданими сценаріями.

Змістовний модуль «Технології розробки комп'ютерних ігор» дисципліни «Проектування інформаційних систем» орієнтовно на оволодіння студентами практичними

- навичками розробки, впровадження і адаптації ПО;

- навичками написання коду з використанням мов програмування/розмітки, визначення та маніпулювання даними;

- навичками проектування і створення користувацьких інтерфейсів;

- навичками проведення інтеграційного і модульного тестування ІС на основі тест-планів (сценаріїв).

2 ОПИС МОДУЛЯ

Галузь знань: 12 "Інформаційні технології".

Спеціальність: 122 "Комп'ютерні науки та інформаційні технології"

Рівень: бакалавр.

Назва дисципліни: Проектування інформаційних систем

Назва змістовного модуля: Технології розробки комп'ютерних ігор

Семестр: 7

Кількість кредитних одиниць: дисципліна - 4,0, модуль - 1,0

Орієнтовна кількість годин: дисципліна - 120, модуль - 30

Викладачі: доцент, к.т.н. Бреславець В.С.; старший викладач, Під'ячий Г.Ю.

3 ПЕРЕЛІК КОМПЕТЕНТНОСТЕЙ ТА РЕЗУЛЬТАТИ НАВЧАННЯ[1]

ЗАГАЛЬНІ (УНІВЕРСАЛЬНІ) КОМПЕТЕНТНОСТІ

ЗК-3	Здатність і готовність володіти основними методами, способами та засобами одержання, оцінювання, збереження, переробки та використання інформації з різних джерел, які необхідні для рішення наукових і професійних завдань.

СПЕЦІАЛЬНІ (ФАХОВІ) КОМПЕТЕНТНОСТІ

ПК-1	Здатність до математичного та логічного мислення, формулювання та досліджування математичних моделей, зокрема дискретних математичних моделей, обґрунтовування вибору методів і підходів для розв'язування теоретичних і прикладних задач в галузі комп'ютерних наук, І інтерпретування отриманих результатів
ПК-2	Здатність до виявлення закономірностей випадкових явищ, застосування методів статистичної обробки даних та оцінювання стохастичних процесів реального світу.
ПКс8-1	Здатність до математичного та логічного мислення, формулювання та досліджування математичних моделей, зокрема дискретних математичних моделей, обґрунтовування вибору методів і підходів для розв'язування теоретичних і прикладних задач в галузі комп'ютерних наук, інтерпретування отриманих результатів.

[1] Профіль програми освітнього ступенів бакалавра. Спеціальність 122 Комп'ютерні науки та інформаційні технології, спеціалізація 122-08 Системи штучного інтелекту

ПКс8-4	Здатність застосовувати теоретичні та практичні основи методології та технології моделювання, реалізовувати алгоритми моделювання для дослідження характеристик і поведінки складних об'єктів і систем, проводити експерименти за програмою моделювання з обробкою й аналізом результатів.

ПРОГРАМНІ РЕЗУЛЬТАТИ НАВЧАННЯ

ЗАГАЛЬНА ПІДГОТОВКА

РНз-3	Знання методів, способів та технологій збору інформації з різних джерел, контент-аналізу документів, аналізу та обробки даних.

ПРОФЕСІЙНА ПІДГОТОВКА

РН-1	Знання теоретичних і прикладних положень неперервного та дискретного аналізу, включаючи аналіз нескінченно малих, інтегральне числення, лінійну алгебру, аналітичну геометрію, диференційні рівняння, функціональний аналіз, комбінаторику, теорію графів, булеву алгебру.
РН-2	Знання закономірностей випадкових явищ, їх властивостей та операцій над ними, теорем і законів розподілу випадкових величин, ймовірнісні методи дослідження складних систем, базові поняття математичної статистики, методи опрацювання емпіричних даних, перевірки статистичних гіпотез на основі вибіркових даних, елементи теорії регресії і кореляції.
РНс8-1	Знання теоретичних і прикладних положень неперервного та дискретного аналізу, включаючи аналіз нескінченно малих, інтегральне числення, лінійну алгебру, аналітичну геометрію, диференційні рівняння, функціональний аналіз, комбінаторику, теорію графів, больову алгебру.
РНс8-4	Знання моделей систем масового обслуговування, мереж Петрі; методології ймовірнісного та імітаційного моделювання об'єктів, процесів і систем; планування та проведення експериментів з моделями, прийняття рішень щодо досягнення мети за результатами моделювання.

ПЕРЕЛІК КОМПЕТЕНТНОСТЕЙ ТА РЕЗУЛЬТАТИ НАВЧАННЯ МОДУЛЬ «ТЕХНОЛОГІЇ РОЗРОБКИ КОМП'ЮТЕРНИХ ІГОР»

Загальні (універсальні) компетентності	ЗК-3
Спеціальні (фахові) компетентності	ПК-1, ПК-2, ПКс8-1, ПКс8-4
Загальна підготовка	РНз-3
Професійна підготовка	РН-1, РН-1, ПНс8-1,ПНс-8-4

4 МІЖДИСЦИПЛІНАРНІ ЗВ'ЯЗКИ

Для засвоєння матеріалу використовується такий перелік забезпечуючих дисциплін:

- Дискретна математика
- Основи баз даних та знань (Моделі даних та знань, системи підтримки прийняття рішень, видобування даних, інтелектуальний аналіз даних)
- Основи програмування та алгоритмічні мови
- Об'єктно-орієнтоване програмування

5 МЕТА ТА ПЕРЕДБАЧУВАНІ РЕЗУЛЬТАТИ ВИВЧАННЯ МОДУЛЯ

5.1 МЕТА МОДУЛЯ

Мета модуля – формування у студентів системи понять, знань, умінь і навичок в галузі об'єктно - орієнтованого програмування, яке включає до себе методи проектування, аналізу і створення ігрових продуктів і їх супроводження, розвиток логічного мислення, формування наукового світогляду, прищеплення схильності до творчості.

5.2 РЕЗУЛЬТАТИ НАВЧАННЯ

ЗНАННЯ ТА ЇХ ВИКОРИСТАННЯ

У разі успішного оволодіння матеріалами модуля студент буде вміти використовувати основні поняття проектування інформаційних систем по видах забезпечення, сучасні методики тестування ІС, що розробляються, сучасну теорію побудови комп'ютерних ігор та програмних продуктів, методи та технології їх розробки, а саме: концепцію та технологію побудови інформаційних систем; логічний рівень опису інформаційних систем, моделі, класи, моделі поведінки інформаційних систем та вмінь проектувати інформаційні системи, реалізовувати технологію розробки комп'ютерних ігор сучасними програмними засобами; використовувати технологію розробки комп'ютерних ігор для роботи в інформаційних системах підтримки прийняття рішень.

ДОСЛІДНИЦЬКІ НАВИЧКИ

У разі успішного вивчення модуля студент буде вміти розробляти і налагоджувати ефективні алгоритми та програми з використанням сучасних технологій програмування; застосовувати сучасні розробки та тенденції в галузі створення програмних продуктів в професійній діяльності; шукати дефекти системи в процесі тестування, брати участь в їх виправленні і модернізації додатку, який тестується.

СПЕЦІАЛЬНІ ВМІННЯ

У разі успішного вивчення модуля студент буде вміти:
- визначати необхідні інструментальні засоби для створення програмних додатків та ігор;
- використовувати можливості програмного середовища розробки додатків Eclips IDE, Android SDK та JDK для створення програмних додатків та ігр;
- приймати участь у розробці комп'ютерних програмних додатків, ігрових додатків та інформаційних систем підтримки прийняття рішень.

СОЦІАЛЬНІ ВМІННЯ

У разі успішного вивчення модуля студент буде вміти приймати участь в командній роботі: обговорювати в групі шляхи побудови програмних додатків та ігрових додатків.

ОСОБИСТІ ЯКОСТІ

У разі успішного вивчення модуля студент буде вміти:
- розробляти, впроваджувати та адаптувати ПЗ;
- мати навички написання коду з використанням мов програмування/розмітки, визначення та маніпулювання даними;
- проектувати та створювати користувацькі інтерфейси;
- проводити інтеграційне та модульне тестування ІС на основі тест- планів (сценаріїв).
- аналізувати сучасну науково-технічну літературу з питань створення програмних додатків та їх застосування, в тому числі у комп'ютерних ігрових додатках.

6 КАЛЕНДАРНИЙ ПЛАН СЕМЕСТРУ І СТРУКТУРА МОДУЛЯ

ІНФОРМАЦІЙНЕ НАПОВНЕННЯ ЗМІСТОВНОГО МОДУЛЯ

Номер	Номер тижня	Зміст
1	10-11	Основи розробки комп'ютерних ігор. Нариси історії комп'ютерних ігор. Етапи розробки комп'ютерної гри. Ігрові професії. Перспективи програміста-розробника комп'ютерних ігор. Введення до XNA Game Studio
2	12-13	Інструментарій розробника комп'ютерних ігор. Графічні файли. Файли тривимірних моделей. Файли шрифтів. Звукові файли. Файли ефектів. Ігрова термінологія. Огляд XNA Game Studio - історія, розвиток, особливості застосування. 2D-графіка в XNA Game Studio
3	14-15	Психологія комп'ютерних ігор. Жанри комп'ютерних ігор. Ігри та навчання. Ігри та формування професійних компетенцій. Пристрої введення, переміщення об'єктів.

Номер	Номер тижня	Зміст
4	16	Тривимірна графіка. Створення ігрового проекту. Розробка ігрової документації. Концепт-документ. Дизайн-документ. План розробки гри. Розбір коду стандартного ігрового проекту. Система координат. Розробка класу для зберігання графічної інформації. Розробка ігрового компонента. Система координат. Перетворення в тривимірному просторі. Об'єкти XNA для роботи з тривимірною графікою. Взаімодія об'єктів

7 ФОРМИ НАВЧАННЯ

Навчальне навантаження модуля складається з аудиторної та самостійної роботи. Аудиторна робота включає 4 лекції та 4 лабораторні роботи. Самостійна робота студентів передбачає підготовку до аудиторних занять і лабораторних робіт, а також підготовку до тесту та оформлення залікового звіту з лабораторних робіт.

Підготовка до поточних аудиторних занять є аналіз літератури, інтернет-матеріалів по темам лекцій і лабораторних робіт, підготовка до тестів.

Контактні години передбачають індивідуальні консультації та контроль студентів в он-лайн режимі.

Заліковий звіт – опис, презентація та ігровий проект за результатами виконання лабораторних робіт 1 - 4. – Технології розробки комп'ютерних ігор.

8 ПОРЯДОК ПРОВЕДЕННЯ АТЕСТАЦІЇ

Загальний принцип оцінювання підсумкових знань студента з курсу "Технології розробки комп'ютерних ігор" полягає в тестуванні студентів на лекціях, оцінці поточної практичної роботи студента у навчальному семестрі на лабораторних роботах та оцінки контрольного заходу у формі іспиту, у результаті яких студент має сумарну оцінку в балах.

За результатами вивчення кожного змістовного модуля передбачено оцінка виконання залікового завдання. Оцінка включає: результати тестування студентів на лекціях, результати поточного опитування при виконанні лабораторних робіт модуля, оцінку за виконання залікового завдання. Сумарно оцінка кожного змістовного модуля не може бути більш ніж 30 балів. Максимальна оцінка іспиту 40 балів.

ОЦІНКА РЕЗУЛЬТАТІВ ВИВЧЕННЯ ДИСЦИПЛІНИ В ЦІЛОМУ

Поточне тестування	Балів
Змістовний модуль 1	до 30
Змістовний модуль 2	до 30
Іспит	до 40
Разом	до 100

ГРАФІК ПРОВЕДЕННЯ ПОТОЧНОГО ОЦІНЮВАННЯ МОДУЛЯ

Номер тижня	Оцінювання
13	Оцінка виконання лабораторної роботи 1
14	Оцінка виконання лабораторної роботи 2
15	Оцінка виконання лабораторної роботи 3
16	Оцінка виконання лабораторної роботи 4 Оцінка залікового звіту

ПРЕДСТАВЛЕННЯ ЗВІТУ ЩОДО ВИКОНАННЯ ЛАБОРАТОРНИХ РОБІТ

При представленні звіту з лабораторних робіт необхідно виконувати наступне. При умові виконання кожної окремої лабораторної роботи єдиний звіт надається студентом на 16 тижні семестру. Продовження терміну можливе лише при наявності поважної причини, передбаченої порядком навчання студентів у вищій школі. При цьому:

електронна версія єдиного звіту за результатами проходження модуля надається викладачеві через систему Інтернет на 9 та 16 тижні;

під час виконання 4-ї лабораторної роботи на 15 тижні студент демонструє закінчений варіант проекту комп'ютерної гри;

за кожний день прострочки представлення та здачі єдиного звіту по модулю знімається 1 бал (не більш ніж 5 днів – 5 балів).

МЕТОД ОЦІНКИ ЗМІСТОВНОГО МОДУЛЯ

Кількість балів в загальній оцінці змістовного модулю відповідно така:

Тестування по модулю	максимально 2 бал.
Виконання лабораторної роботи 1	максимально 6 бали.
Виконання лабораторної роботи 2	максимально 6 бали.
Виконання лабораторної роботи 3	максимально 6 бали.
Виконання лабораторної роботи 4	максимально 6 бали.
Оцінка залікового Звіту	максимально 4 бали.

Усі набрані бали підсумовуються (максимально 15 балів), штрафні бали за запізнення в представленні єдиного звіту (максимально мінус 5 балів) віднімаються. Сумарна оцінка (від 0 до 15 балів) є індивідуальна оцінка студента освоєння змістовного модуля 1.

МЕТОД ОЦІНКИ ДИСЦИПЛІНИ В ЦІЛОМУ

Оцінки студентів за результатами вивчення змістовних модулів 1 – 4 підсумовуються. Оцінка іспиту (максимально 40 балів) додається. Таким чином розраховується сумарна оцінка студента в балах за дисципліною.

Сумарна оцінка в балах переводиться за нижченаведеною шкалою оцінювання в національну та ЄКТС- оцінку:

Сума балів за всі види навчальної діяльності	Оцінка ECTS	Оцінка за національною шкалою
90 – 100	A	Відмінно
82 – 89	B	Дуже добре
74 – 81	C	Добре
64 – 73	D	Задовільно
60 – 63	E	Посередньо
35 – 59	FX	Не зараховано з можливістю повторного складання
0 – 34	F	Не зараховано з обов'язковим повторним вивченням дисципліни

КРИТЕРІЇ ОЦІНЮВАННЯ ПРЕЗЕНТАЦІЇ РЕЗУЛЬТАТІВ РОБОТИ

	Високий	Середній		Низький
	4	3	2	1
організація	Організаційна структура (конкретне введення і висновок, послідовність викладу матеріалу всередині тіла і переходи) чітко і послідовно спостерігалася, вміло представлена і робить зміст презентації цілісним.	Організаційна структура (конкретне введення і висновок, послідовність викладу матеріалу всередині тіла і переходи) чітко і послідовно проглядається в презентації.	Організаційна структура (специфічне введення і висновок, послідовність викладу матеріалу всередині тіла і переходи) періодично проглядається в презентації.	Організаційна структура (специфічне введення і висновок, послідовність викладу матеріалу всередині тіла і переходи) не спостерігається в презентації.
Мова	Вибір мови є вражаючим, що запам'ятовується, є привабливим і підвищує ефективність презентації. Мова в презентації підходить аудиторії.	Вибір мови є продуманим і в цілому підтримує ефективність презентації. Мова в презентації підходить аудиторії.	Вибір мови є повсякденним і звичайним і частково підтримує ефектність презентації. Мова в презентації підходить аудиторії.	Вибір мови незрозумілий і мінімально підтримує ефективність презентації. Мова в презентації не підходить аудиторії.

	Високий	Середній		Низький
	4	3	2	1
Подача	Вміння подачі (поза, жест, зоровий контакт і голосова виразність) роблять презентацію переконливою, а доповідач виглядає підготовленим, впевненим.	Вміння подачі (поза, жест, зоровий контакт і голосова виразність) роблять презентацію цікавою, і динаміка доповідача виглядає впевнено.	Вміння подачі (поза, жест, зоровий контакт і голосова виразність) роблять презентацію зрозумілою, а доповідач виглядає не дуже впевнено.	Вміння подачі (поза, жест, зоровий контакт і голосова виразність) роблять презентацію незрозумілою, а доповідь виглядає невпевнено.
Допоміжні матеріали	Різноманітні типи допоміжних матеріалів (пояснення, приклади, ілюстрації, статистика, аналоги, цитати з відповідних джерел) дають посилання на інформацію або аналіз, які в значній мірі підтримують презентацію і підтверджують авторитет доповідача в представленій темі.	Допоміжні матеріали (пояснення, приклади, ілюстрації, статистика, аналогії, цитати з відповідних джерел) містять відповідні посилання на інформацію або аналіз, які в цілому підтримують презентацію і підтверджують авторитет доповідача в представленій темі.	Допоміжні матеріали (пояснення, приклади, ілюстрації, статистичні дані, аналоги, цитати з відповідних джерел) містять відповідні посилання на інформацію або аналіз, які частково підтримують презентацію і авторитет доповідача в представленій темі.	Недостатні допоміжні матеріали (пояснення, приклади, ілюстрації, статистика, аналоги, цитати з відповідних джерел) містять відповідні посилання на інформацію або аналіз, які мінімально підтримують презентацію і авторитет доповідача в представленій темі.
Головна думка	Головна думка є переконливою (точно сформульована, відповідним чином повторюється, запам'ятовується і міцно підтримується).	Головна думка є ясною і узгоджується з допоміжним матеріалом.	Головна думка є в принципі зрозумілою, але не часто повторюється і не запам'ятовується.	Головна думка може бути виведена, але явно не вказана в презентації.

9 ЗВОРОТНІЙ ЗВ'ЯЗОК

Інформація щодо результатів тестування, виконання лабораторних робіт та загальна оцінка змістовного модулю надається кожному студенту як індивідуально, так і групі в цілому.

Інформація щодо результатів тестування надається студентам по завершенню тестування. Загальні результаті надаються на 16 тижні навчання.

Інформація щодо оцінки виконання лабораторної роботи надається студентові під час заняття.

Інформація щодо оцінки змістовного модуля в цілому надається студентам на 9 та 16 тижні навчання.

Контактні дані для on-line допомоги та консультування:

викладачі: доцент, к.т.н. Бреславець В.С., e-mail: bres7272@gmail.com

старший викладач, Під'ячий Г.Ю., e-mail: glebpod@gmail.com

10 ВИКЛАДАЦЬКИЙ СКЛАД ТА ДОПОМІЖНІ ДЖЕРЕЛА

ОБОВ'ЯЗКИ ВИКЛАДАЧІВ

- подача матеріалів модуля згідно з програмою;
- оцінка результатів тестування та виконання лабораторних робіт.

ОБОВ'ЯЗКИ КООРДИНАТОРА ДИСЦИПЛІНИ

- планування та внесення змін до модулю;
- координація і управління професорсько-викладацьким складом;
- координація проведення тестування, лабораторних робіт та іспиту.

ОБОВ'ЯЗКИ ДОПОМІЖНОГО ПЕРСОНАЛУ

Допоміжний персонал здійснює підготовку комп'ютерної техніки до виконання лабораторних робіт студентами та надає технічну підтримку студентів під час виконання лабораторних робіт.

КОНТАКТНІ ДАНІ ВИКЛАДАЧІВ

доцент, к.т.н. Бреславець В.С., e-mail: bres7272@gmail.com

старший викладач, Під'ячий Г.Ю., e-mail: glebpod@gmail.com

КОНТАКТНІ ДАНІ КУРАТОРА

доцент, к.т.н. Бреславець В.С., e-mail: bres7272@gmail.com

11 НАВЧАЛЬНА ПРОГРАМА І МАТЕРІАЛИ

11.1 ТЕМА 1: ОСНОВИ РОЗРОБКИ КОМП'ЮТЕРНИХ ІГОР

АНОТАЦІЯ

Лекція знайомить з такими поняттями як: Історія комп'ютерних ігор. Етапи розробки комп'ютерної гри. Ігрові професії. Перспективи програміста-розробника комп'ютерних ігор.

МЕТА ЛЕКЦІЇ

Ознайомити студентів з історією створення комп'ютерних ігор, етапами розробки комп'ютерної гри, типами ігрових професій, перспективами програміста-розробника комп'ютерних ігор.

ОЧІКУВАНІ РЕЗУЛЬТАТИ

Студент оволодіє знаннями з історія комп'ютерних ігор, етапи розробки комп'ютерної гри, ігрових професій, перспектив програміста-розробника комп'ютерних ігор.

КОНТРОЛЬНІ ЗАПИТАННЯ

- Які етапи розвитку комп'ютерних ігор Ви знаєте?.
- Які етапи розробки комп'ютерної гри Ви знаєте?
- Що таке підготовка виробництва комп'ютерної гри?
- Що таке виробництво комп'ютерної гри?
- Що таке випуск комп'ютерної гри?
- Що таке підтримка комп'ютерної гри?
- Які ігрові професії Ви знаєте?
- Які функції виконують програмісти комп'ютерної гри?
- Які функції виконують художники комп'ютерної гри?
- Які функції виконують музики комп'ютерної гри?
- Які функції виконують письменники комп'ютерної гри?
- Які функції виконують дизайнери рівнів комп'ютерної гри?
- Які функції виконують тестувальники комп'ютерної гри?
- Які перспективи є у програміста – розробника комп'ютерних ігор?

МЕТОДИЧНІ МАТЕРІАЛИ ТА ВКАЗІВКИ

HTTP://BLOGS.KPI.KHARKOV.UA/V2/GAMEHUB/DOWNLOAD

11.2 ЛАБОРАТОРНА РОБОТА № 1.

РОЗРОБКА ІГРОВОГО ПРОЕКТУ З ВИКОРИСТАННЯМ ВІЗУАЛЬНИХ ЗАСОБІВ РОЗРОБКИ: SCRATCH, MICROSOFT KODU GAME LAB

АНОТАЦІЯ

Лабораторна робота орієнтована на ознайомлення студентів з основними елементами середовища розробки додатків для ОС Windows і Android. Середовище розробки Scratch, Microsoft Kodu Game Lab.

МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Освоїти основні прийоми розробки додатків для ОС Windows і Android, вивчити процес встановлення і налаштування середовища розробки, знайомство з Android Virtual Device та призначеним для користувача інтерфейсом (UI), проведення сеансу налагодження.

ОЧІКУВАНІ РЕЗУЛЬТАТИ

У разі успішного виконання лабораторної роботи студент буде вміти використовувати середовища розробки додатків Scratch, Microsoft Kodu Game Lab для ОС Windows і Android, зберігати та завантажувати проект розробки, буде знати та вміти застосовувати основні прийоми роботи з інструментами що входять до складу цих додатків.

КОНТРОЛЬНІ ЗАПИТАННЯ

- Засоби та платформи розробки мобільних додатків, основні функціональні можливості.
- Призначення середовища розробки Scratch.
- Перерахуйте основні елементи інтерфейсу що застосовуються в мобільних додатках.
- Призначення середовища розробки Microsoft Kodu Game Lab.
- Життєвий цикл додатка.

МЕТОДИЧНІ МАТЕРІАЛИ ТА ВКАЗІВКИ

HTTP://BLOGS.KPI.KHARKOV.UA/V2/GAMEHUB/DOWNLOAD

11.3 ТЕМА 2. ІГРОВІ РЕСУРСИ

АНОТАЦІЯ

Лекція знайомить з основами типами ігрових ресурсів, такими як графічні файли, файли тривимірних моделей, файли шрифтів, звукові файли, файли ефектів. Також розглядаються ігрова термінологія, огляд XNA Game Studio - історія, розвиток, особливості застосування.

МЕТА ЛЕКЦІЇ

Ознайомити студентів з основами типами ігрових ресурсів, архітектурою та побудовою цих ресурсів, формами взаємодії між ними, сучасними міжнародними стандартами

створення різноманітних типів ігрових файлів та платформи для їх розробки.

ОЧІКУВАНІ РЕЗУЛЬТАТИ

Сформувати у студентів знання щодо основних принципів побудови типів ігрових ресурсів, форм взаємодії між ними. Навчити використовувати сучасні міжнародні стандарти створення різноманітних типів ігрових файлів, платформи для розробки ігрових ресурсів та області застосування цих платформ.

КОНТРОЛЬНІ ЗАПИТАННЯ

- З чого складаються ігрові ресурси?
- Взаємодії між ресурсами.
- Структура типового графічного файлу.
- Структура типового файлу тривимірної моделі.
- Структура типового файлу шрифтів.
- Які бувають типи звукових файлів?
- Структура типового файлу ефектів.
- Ігрова термінологія.
- Що таке платформа XNA Game Studio 2.0. – історія, розвиток, особливості застосування?

МЕТОДИЧНІ МАТЕРІАЛИ ТА ВКАЗІВКИ

HTTP://BLOGS.KPI.KHARKOV.UA/V2/GAMEHUB/DOWNLOAD

11.4 ЛАБОРАТОРНА РОБОТА № 2.

РОЗРОБКА ІГРОВОГО ПРОЕКТУ З ВИКОРИСТАННЯМ ВІЗУАЛЬНИХ ЗАСОБІВ РОЗРОБКИ: GAME MAKER.

АНОТАЦІЯ

Лабораторна робота орієнтована на оволодіння студентами навичок розробки ігрових додатків.

МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Розробка ігрового додатка з використанням середовища розробки Game Maker.

ОЧІКУВАНІ РЕЗУЛЬТАТИ

У разі успішного виконання лабораторної роботи студент буде вміти створювати ігрові додатки з деяким набором функцій.

КОНТРОЛЬНІ ЗАПИТАННЯ

- Створення нового проекту в Game Maker.
- Структура файлу проекту Game Maker.
- Які функції задіюються при управлінні об'єктами.
- Функції для роботи з файлами.

МЕТОДИЧНІ МАТЕРІАЛИ ТА ВКАЗІВКИ
HTTP://BLOGS.KPI.KHARKOV.UA/V2/GAMEHUB/DOWNLOAD

11.5 ТЕМА 3: ПСИХОЛОГІЯ КОМП'ЮТЕРНИХ ІГОР

АНОТАЦІЯ
Лекція знайомить з жанрами комп'ютерних ігор, як взаємодіють ігри та навчання, ігри та формування професійних компетенцій.

МЕТА ЛЕКЦІЇ
Ознайомити студентів з жанрами комп'ютерних ігор, показати, як взаємодіють ігри та процес навчання, як використовуються ігри при формуванні професійних компетенцій.

ОЧІКУВАНІ РЕЗУЛЬТАТИ
Сформувати у студентів знання щодо основних жанрів комп'ютерних ігор, сформувати компетентності для взаємодії ігри та процесу навчання, сформувати підходи для використання ігор при формуванні професійних компетенцій.

КОНТРОЛЬНІ ЗАПИТАННЯ
- Основні жанри комп'ютерних ігор.
- Що таке ігри змішаних жанрів.
- Як використовуються ігри при навчанні.
- Використання ігор при формуванні професійних компетенцій.

МЕТОДИЧНІ МАТЕРІАЛИ ТА ВКАЗІВКИ
HTTP://BLOGS.KPI.KHARKOV.UA/V2/GAMEHUB/DOWNLOAD

11.6 ЛАБОРАТОРНА РОБОТА №3. РОЗРОБКА И СТВОРЕННЯ FLASH-ІГОР

АНОТАЦІЯ
Лабораторна робота орієнтована на оволодіння студентами навичок роботи з платформою Flash.

МЕТА ЛАБОРАТОРНОЇ РОБОТИ
Знайомство з платформою розробки додатків Flash, встановити платформу на робочий комп'ютер, освоїти основні прийоми роботи з нею.

ОЧІКУВАНІ РЕЗУЛЬТАТИ
У разі успішного виконання лабораторної роботи студент буде вміти створювати ігрові додатки з використанням платформи Flash.

КОНТРОЛЬНІ ЗАПИТАННЯ
- Процес установки JDK та платформи Flash.
- Призначення і властивості контейнера в Flash.

- Створення нового проекту в середовищі Flash.
- Конфігурація запуску додатку в середовищі.

МЕТОДИЧНІ МАТЕРІАЛИ ТА ВКАЗІВКИ

HTTP://BLOGS.KPI.KHARKOV.UA/V2/GAMEHUB/DOWNLOAD

11.7 ТЕМА 4: ІГРОВИЙ ПРОЕКТ. 2D-ГРАФІКА В XNA GAME STUDIO. ТРИВИМІРНА ГРАФІКА

АНОТАЦІЯ

Лекція знайомить з процесом створення ігрового проекту, з тим, як розробляється ігрова документація, концепт-документ, дизайн-документ, план розробки гри, розбір коду стандартного ігрового проекту; що таке система координат, розробка класу для зберігання графічної інформації, розробка ігрового компонента, система координат. Як відбуваються перетворення в тривимірному просторі, що таке об'єкти XNA для роботи з тривимірною графікою.

МЕТА ЛЕКЦІЇ

Ознайомити студентів з процесом створення ігрового проекту, з тим, як розробляється ігрова документація, які основні типи документації, яка повинна бути підготована розробником. Дати поняття, що таке система координат, показати, як відбувається розробка класу для зберігання графічної інформації, розробка ігрового компонента, що таке система координат. Як відбуваються перетворення в тривимірному просторі, що таке об'єкти XNA для роботи з тривимірною графікою.

ОЧІКУВАНІ РЕЗУЛЬТАТИ

Сформувати у студентів знання щодо реалізації самостійного ігрового проекту. Навчити самостійно вирішувати питання підготовки основних типів ігрової документації, навчити роботі з тривимірними об'єктами та об'єктами в форматі.

КОНТРОЛЬНІ ЗАПИТАННЯ

- Які типи ігрової документації Ви знаєте?
- Що таке концепт – документ?
- Що таке дизайн – документ?
- Як створюється план розробки гри?
- Як відбуваються перетворення в тривимірному просторі?
- Що таке об'єкти XNA для роботи з тривимірною графікою?

МЕТОДИЧНІ МАТЕРІАЛИ ТА ВКАЗІВКИ

HTTP://BLOGS.KPI.KHARKOV.UA/V2/GAMEHUB/DOWNLOAD

11.8 ЛАБОРАТОРНА РОБОТА № 4. ІГРОВИЙ ПРОЕКТ. 2D - ГРАФІКА В XNA GAME STUDIO. ТРИВИМІРНА ГРАФІКА

АНОТАЦІЯ

Лабораторна робота орієнтована на оволодіння студентами навичками створення ігрового проекту з використанням 2D - графіки в XNA Game Studio та тривимірної графіки.

МЕТА ЛАБОРАТОРНОЇ РОБОТИ

Створити робочий ігровий проект з використання пакета 2D - графіки в XNA Game Studio та тривимірної графіки.

ОЧІКУВАНІ РЕЗУЛЬТАТИ

У разі успішного виконання лабораторної роботи студент буде вміти створювати ігрові додатки для ОС Windows з використанням спеціалізованої платформи XNA Game Studio.

КОНТРОЛЬНІ ЗАПИТАННЯ

- Налаштування XNA Game Studio.
- Етапи та налаштування ігрового проекту.
- Запуск проекту.
- Формування ігрової документації до проекту.

МЕТОДИЧНІ МАТЕРІАЛИ ТА ВКАЗІВКИ

HTTP://BLOGS.KPI.KHARKOV.UA/V2/GAMEHUB/DOWNLOAD

12 СПИСОК ДЖЕРЕЛ ІНФОРМАЦІЇ

1. Мазалов В. В. Математическая теория игр и приложения : учеб. пособие / В. В. Мазалов. — Санкт–Петербург : Лань, 2010. — 448 с. — ISBN 978–5–8114–1025–5.
2. Стиллмен Э. Изучаем C# / Э. Стиллмен, Дж. Грин. — 3–е изд. — Санкт–Петербург : Питер, 2012. — 696 с. : ил.
3. Смирнова Е. О. Психология и педагогика игры : учебник и практикум / Е. О. Смирнова, И. А. Рябкова. — Москва : Изд–во Юрайт, 2016. — 223 с. — ISBN 978–5–9916–6807–1.
4. Хортон А. Microsoft Visual C++ 2005: базовый курс / Айвор Хортон. — Москва : Диалектика, 2007. — 1152 с. — ISBN 0–7645–7197–4.
5. Bates B. Game Design / Bob Bates. — 2nd ed. — Course Technology PTR, 2004. — 377 p. — ISBN 1–59200–493–8.
6. Bethke E. Game development and production / Erik Bethke. — Texas : Wordware Publishing, Inc., 2003. — 432 p. — ISBN 1–55622–951–8.
7. Brathwaite B. Challenges for Game Designers / Brenda Brathwaite, Ian Schreiber. — Charles River Media, 2009. — 347 p. — ISBN 1–58450–580–X.
8. Chandler H. M. The Game Production Handbook / Chandler Heather Maxwell. — 2nd ed. — Hingham, Massachusetts : Infinity Science Press, 2009. — 442 p. — ISBN 978–1–934015–40–7.
9. McGuire M. Creating Games: Mechanics, Content, and Technology / Morgan McGuire. — Wellesley, Massachusetts : A K Peters, 2009. — 500 p. — ISBN 978–1–56881–305–9.
10. McShaffry M. Game Coding Complete / McShaffry Mike. — Hingham, Massachusetts : Charles River Media, 2009. — 754 p. — ISBN 978–1–58450–680–5.
11. Moore M. E. Game Industry Career Guide / Michael E. Moore, Jeannie Novak. — Delmar Cengage Learning, 2010. — 108 p. — ISBN 978–1–4283–7647–2.
12. Oxland K. Gameplay and design / Oxland Kevin. — Addison Wesley, 2004. — 349 p. — ISBN 0–321–20467–0.
13. Rollings A. On game design / Andrew Rollings, Ernest Adams. — New Riders Publishing, 2003. — 648 p. — ISBN 1–59273–001–9.
14. Salen K. Rules of Play / Katie Salen, Eric Zimmerman. — MIT Press, 2003. — 688 p. — ISBN 0–262–24045–9.
15. Salen K. The Game Design Reader: A Rules of Play Anthology / Katie Salen, Eric Zimmerman. — MIT Press, 2005. — ISBN 0–262–19536–4.

Лекції

ЛЕКЦІЯ 1. ОСНОВИ РОЗРОБКИ КОМП'ЮТЕРНИХ ІГОР.

Анотація: Ця лекція присвячена основам розробки комп'ютерних ігор. Тут ми будемо говорити про ігри та про їх створення без розгляду конкретних інструментів. Багато курси починаються з історичної довідки - ми не будемо відходити від цієї традиції.

Ключові слова: video game, Multics, computation space, flight simulation, PAC, entertainment system, 'fantasy', craft, 3dfx, empirical, Інтернет, developer, видавець, безлічі, команда, група

1.1. НАРИСИ ІСТОРІЇ КОМП'ЮТЕРНИХ ІГОР

Історія відеоігор починається в п'ятдесятих роках минулого століття. Тоді з'явилися перші пристрої, які пізніше перетворилися в ігрові приставки, портативні ігрові консолі та персональні комп'ютери, які сьогодні є найстаршою ігровою платформою.

Термін «відеоігри» (video games) сьогодні використовується для позначення всіх видів ігор, взаємодія з якими організовано за допомогою різних пристроїв для відображення відеоінформації. У цю категорію потрапляють комп'ютерні ігри, ігри для ігрових приставок, для мобільних пристроїв.

ОСНОВНІ ЕТАПИ РОЗВИТКУ ВІДЕОІГОР.

1940-І РОКИ

Комп'ютерні ігри ведуть свою історію з 1947 року. Тоді була створена перша гра - ракетний симулятор. Ця гра навряд чи змогла б стати масовою - а масовість - це одна з ознак сучасних ігор. Вона була реалізована за допомогою запатентованої в 1947 році катодно-променевої трубки. Гравець міг управляти світловим плямою - ракетою, якій потрібно було уразити ціль. Причому, так як в ті часи катодно-променеві трубки були надзвичайно обмежені в можливостях, мета була просто намальована і прикріплена до поверхні екрану.

Створення ігрових пристроїв і програм приваблювало багатьох людей, які працювали в високотехнологічній сфері, людей, пов'язаних з комп'ютерами, тому вже в 50-х роках ми можемо спостерігати розвиток цієї сфери.

1950-ТІ РОКИ

Розвиток комп'ютерів, засобів відображення інформації, призвело до створення кількох помітних ігрових проектів. Як і раніше, вони розглядалися скоріше як досягнення мистецтва програміста і сучасної в той період часу техніки, а не як гри, в які можуть грати всі охочі.

Так, в 1952 році Олександр Дуглас розробив програму OXO - комп'ютерний варіант гри в хрестики-нулики. Ця програма була створена в рамках його наукової роботи, яка була присвячена людино-машинного взаємодії. Програма містила модуль штучного інтелекту - саме він дозволяв комп'ютера грати з людиною.

У 1958 році Вільям Хігінботен розробив гру Tennis for Two (Теніс для двох) - простий симулятор гри в теніс. Нерідко саме цю гру вважають першою цієї відеогрою. Для управління в грі використовувалися контролери, оснащені джойстиком і кнопкою. Як пристрій відображення інформації гра використовувала осцилограф. Пристрій служило для розваги відвідувачів Брукхевенської Національної Лабораторії в Нью-Йорку.

1960-І РОКИ

У 60-х роках з'являються проекти, які цілком можна порівняти з сучасними комп'ютерними іграми. Однак, реалізовувалися вони, в основному, на ЕОМ, які були в розпорядженні навчальних закладів.

У 1961 році Стів Рассел і його товариші з Массачусетського університету (США) написали гру Spacewar! (Саме так - зі знаком оклику) - вона працювала на комп'ютері PDP-1. Мета гри - вразити космічний корабель противника і при цьому не потрапити під його вогонь і не зіткнутися із зіркою.

У 1966 Ральф Баєр створив комп'ютерну гру Chase, яка примітна тим, що вперше використовувала в якості пристрою для виведення інформації звичайний телевізор.

У 1969 Кен Томпсон з компанії AT & T написав гру Space Travel для ОС MULTICS. Надалі гра була перенесена на інші ОС, в результаті вона стала першою програмою для ОС Unix.

1970-ТІ РОКИ

Якщо раніше відеоігри були чимось рідкісним, то 70-ті роки можна вважати справжнім ігровим бумом. Ігри розвивалися в кількох напрямках - це були ігрові автомати, ігри для великих комп'ютерів, встановлених в навчальних закладах, ігри для домашніх комп'ютерів, а так само - консольні ігри. Зокрема, популярністю користувалися консолі від Atari і Magnavox.

Цікаво розвивалася ситуація серед користувачів домашніх комп'ютерів. Наприклад, в ті часи деякі ігри поширювалися у вигляді текстів, надрукованих в журналах або книгах. І домашній користувач ПК повинен був вручну набирати текст гри, налагоджувати, компілювати програму, і, в результаті - грати.

У 1971 з'явився перший ігровий автомат - він був запущений в Стенфордському університеті. На автомат була встановлена гра Galaxy (вона була заснована на концепції Spacewar!). В цьому ж році Нолен Басшнелл і Тед Дабні створили ігровий автомат, заснований на тій же Spacewar !. Їх гра називалася Computer Space, всього було вироблено 1500 екземплярів цього автомата. В цьому ж році Дон Деглоу написав першу комп'ютерну реалізацію бейсболу на комп'ютері DEC PDP-10. В цьому ж році Майк Мейфіл з Массачусетського технологічного університету розробив гру Star Trek

У 1972 заснована компанія Atari, в цьому ж році вийшла гра Atari PONG і відповідний пристрій. Було продано 19000 копій цієї гри.

У 1974 з'явилися гри Maze War і Spasim - перші спроби створення багатокористувацьких

тривимірних FPS. В цьому ж році Бред Фортнера розробив гру Airflight - авіасимулятор. Ця гра стала прообразом для Microsoft Flight Simulator.

У 1975 році була написана текстова комп'ютерна гра Adventure. Комп'ютери тих часів мали скромні обчислювальні ресурси, тому текстові ігри були популярними. До того ж, багато ігор «спілкувалися» з гравцями, розкриваючи інформацію на принтері.

У 1978 вийшла гра Space Invaders від Taito - її успіх активізував інших виробників ігрових автоматів.

У 1979 році, випущена гра Pac Man. Ця гра актуальна до сих пір. Вона занесена в Книгу Рекордів Гіннеса як найпоширеніша гра в світі.

1980-і

У 80-х роках ігрові автомати, які були популярні в минулому десятилітті, втрачали позиції. Однак початок 80-х можна вважати золотою ерою ігрових автоматів.Теперь на первый план выходят игры для ПК, а так же – для игровых консолей. Среди компьютеров тех времен можно отметить ZX Spectrum, Apple II, Apple Macintosh, Commodore 64, IBM PC.

В это же время появляются многие компании-издатели и компании-разработчики игр. Некоторые из них, например - Electronic Arts – существуют и по сей день.

У 1980-ті роки були закладені основи багатьох популярних сьогодні ігрових жанрів і способів побудови ігрового світу. Наприклад - це скроллінговие гри, квести, мережеві ігри, тривимірні ігри. Завдяки розробці нових апаратних засобів в іграх з'явилося непогане звукове оформлення. У 1985 році була розроблена гра, яка не втрачає популярності і в наші дні. Йдеться про всесвітньо відомому Тетріс, який створив наш співвітчизник Олексій Пажитнов. В цьому ж році випущена Nintendo Entertainment System - ігрова консоль, яка набрала більшу популярність до початку 1990-х років. До речі, навіть сьогодні можна пограти в NES-ігри. Nintendo і деякі інші компанії випускають ігрові консолі, або, як їх прийнято називати, ігрові приставки, до сих пір.

У 1987 була розроблена рольова гра Final Fantasy - вона стала однією з найпопулярніших рольових ігор і до сих пір виходять її нові релізи.

У 1989 компанія Nintendo випустила кишенькову ігрову консоль Game Boy.

1990-і

У 90-х роках індустрія комп'ютерних ігор зміцніла, ігри розроблялися вже не одинаками, як це було раніше, а величезними командами професіоналів.

Сучасними уявленнями ми зобов'язані розробкам, які зроблені в 1990-і роки. А багато ігор тих років все ще популярні. Наприклад - Star Craft (1998), чемпіонати по якому проводяться до сих пір, Duke Nukem 3D, продовження якої очікується вже багато років, але сама гра все не втрачає актуальності, або Counter Strike - мод до Half Life, що вийшов в 1999 році. Перша версія Alone in the Dark вийшла в 1992 році.

Одна з перших 3D-ігор - Quake - побачила світ у 1996 році. Почасти ця гра зобов'язана своїй появі набору мікросхем Voodoo, який випустила компанія 3dfx. Цей набір

мікросхем значно прискорював можливості ПК по обробці тривимірної графіки.

У 90-е почали розвиток Інтернет-ігри. Так, уже згадана Quake, дозволяла влаштовувати розраховані на багато користувачів бої в Інтернеті, то ж саме стосується Starcraft'a, Age of Empires, Ultima Online, EverQuest. Стали з'являтися гри, засновані на Macromedia Flash - такі ігри - зазвичай різні реалізації досить простих, але цікавих концепцій - популярні і сьогодні. Головна особливість Flash-ігор полягає в тому, що для їх виконання потрібен лише WEB-браузер, оснащений відповідними надбудовами.

Що стосується ігрових консолей, то вони розвивалися в 90-х роках. Мабуть, найбільш помітними серед них були SNES і Nintendo 64 від Nintendo, Sega Mega Drive, Sony PlayStation

НАШІ ДНІ

В наші дні гри продовжують розвиватися. На ринку ігрових консолей є три сильних конкурента. Це Microsoft Xbox 360 (її попередниця - Xbox - з'явилася в 2001 році), Sony PlayStation III і Nintendo Wii.

На сьогоднішній день створення комп'ютерних ігор - це величезна індустрія. Бюджети ігрових проектів досягають десятків мільйонів доларів, а обсяг ринків сучасних ігор - комп'ютерних, консольних, мобільних, оцінюється десятками мільярдів доларів.

Можна помітити, що розробкою комп'ютерних ігор на самому початку їх виникнення займалися одинаки. Пізніше, на початку 80-х років, почали виникати компанії-розробники і компанії-видавці ігор. У 1980-90-х роках помітні ігрові проекти зазвичай створювалися працями великих компаній. Однак, останні кілька років роль невеликих груп розробників і навіть одинаків зростає. Цей висновок можна зробити, наприклад, якщо проаналізувати такий феномен, як аматорські модифікації (або, як прийнято говорити, моди) популярних ігор. Наприклад, популярна гра Counter Strike з'явилася у вигляді аматорського мода до Half Life.

Поговоривши про історію ігор, перейдемо до опису етапів створення гри і до характеристики «ігрових» професій.

1.2. ЕТАПИ РОЗРОБКИ КОМП'ЮТЕРНОЇ ГРИ

Розробка комп'ютерних ігор - це досить чітко налагоджений процес, який має певні етапи, так чи інакше прохідні іграми при їх створенні. Однак життя зазвичай вносить свої корективи навіть у самі чіткі плани.

Дуже часто розробники ігор не можуть встигнути доробити гру в скільки-небудь прийнятний термін - яскравий приклад - Duke Nukem Forever, випуску якого весь ігровий світ чекав уже багато років. Практично завжди після виходу комп'ютерної гри за нею йдуть виправлення - вся справа в тому, що розробники, знову ж таки, не вкладаються у відведені їм терміни.

Тексти програм ігор нерідко «йдуть» в Інтернет і всі грають в новітню гру задовго до її офіційного релізу. Причому, тут не можна однозначно сказати, чи шкідливо це для ігрових компаній. З одного боку - шкідливо - адже копії гри потрапляють до

користувачів абсолютно безкоштовно (не рахуючи витрат на трафік в Інтернеті). Однак, з іншого - часто «пішли» коди далекі від досконалості і «витік» лише розігріває інтерес до фінальної версії гри.

Треба врахувати, що в ігровому бізнесі існує два типи компаній - розробник (developer) і видавець (publisher). Якщо розробник і видавець збігаються - процес розробки гри лише виграє - розробнику немає потреби переконувати стороннього видавця в доцільності капіталовкладень в розробку.

ЕТАПИ РОЗРОБКИ ТИПОВОЇ КОМП'ЮТЕРНОЇ ГРИ.

ПІДГОТОВКА ДО ВИРОБНИЦТВА

Підготовка до виробництва гри - це перший етап роботи над грою. Завдання розробників на цьому етапі - розробити концепцію гри, дизайн персонажів, вибрати засоби для реалізації проекту, створити прототип гри, підготувати план, за яким буде створюватися гра і узгодити цей план з начальством, або - з компанією, яка планує видавати гру. Як правило, всі сучасні ігри пишуться під конкретного видавця, який часто вкладає в розробку чималі кошти.

Коли всі адміністративні питання вирішені, гра набуває на етап виробництва.

ВИРОБНИЦТВО

Виробництво - це ключовий етап у створенні гри. Розробники займаються реалізацією раніше створеного плану. Однак початковий план гри піддається змінам - іноді ці зміни відбуваються дуже часто - аж до щоденних коригувань.

В ході виробництва гри - особливо це стосується комерційних версій - періодично влаштовується розгляд поточних результатів розробки, до яких команда повинна представити проект, який досяг певного рівня розвитку. Тобто, наприклад, до одного з таких моментів повинна бути готова працює демо-версія гри, до іншого - перший рівень і так далі. Як правило, ці проміжні результати служать відмінною рекламою нових ігрових проектів - демо-версії публікують на ігрових сайтах, геймери «приміряють» до цих версій можливості свого обладнання.

ВИПУСК

Після того, як гра створена, протестована і налагоджена, настає час її випуску. Як правило, інтерес до цієї події посилено підігрівається видавцем гри - адже не варто забувати, що головна мета видавця - прибуток. Як правило, найбільш успішні гри з лишком виправдовують очікування видавців.

ПІДТРИМКА

Ігри для ПК часто виходять з помилками - вся справа в тому, що розробникам вічно не вистачає часу щоб все як слід налагодити. Благо, є можливість виправляти помилки на вже встановлених іграх, встановлюючи патчі (від англійського patch - латка). Цим користуються розробники, випускаючи сируватий гру і, після цього, цілу низку латочок

для неї. Така практика не поширена для консольних ігор - тут розробники змушені відповідальніше підходити до своєї роботи і випускати повністю робочу гру, яка потребує втручань.

Як бачите, розробка ігор - справа нелегка. Звичайно, вище приведена лише приблизна схема роботи над грою, однак практично всі ігри проходять через однакові етапи.

1.3. ІГРОВІ ПРОФЕСІЇ

В процесі виробництва гри в справу вступають представники безлічі «ігрових» професій - поговоримо про основні з них. До речі, при розробці невеликого проекту в рамках обмеженого бюджету одна людина може поєднувати в собі обов'язки цілої команди розробників.

ПРОГРАМІСТИ

Програмісти зайняті роботою з написання програмного коду гри. Їх зусиллями реалізується ігрова фізика, штучний інтелект, з яким належить битися гравцеві при грі «проти комп'ютера» та багато іншого. Що цікаво, багато ігрові програмісти стали такими після того, як почали програмувати самостійно, у вигляді хобі.

Якщо говорити про інструменти програміста - то практично всі комерційні ігри написані на мові C ++ або C, деякі, особливо відповідальні частини гри, пишуть на мові Assembler. Останнім часом набирає популярність мова програмування C #. Строго кажучи, гру реально написати на практично будь-якій мові програмування - наприклад, прості ігри можна створювати в Microsoft Word або Microsoft Excel, використовуючи вбудований в ці продукти Visual Basic For Applications.

ХУДОЖНИКИ

Роль художників і взагалі всіх, хто працює з графікою, в сучасному ігростроенія важко переоцінити. У всі часи одним з критеріїв оцінки гри була її графічна складова - а сучасні засоби роботи з графікою дозволяють створювати красиві ігрові світи, в основі яких лежить копітка робота художників у всіх її проявах.

До того ж, аніматори оснащені сьогодні передовим апаратним і програмним забезпеченням (зокрема - технологією Motion Capture) яке, наприклад, дозволяє записувати реальні руху людини і переносити їх потім на ігрових персонажів.

МУЗИКАНТИ

Композитори, музиканти, актори, звукорежисери працюють над звуковим оформленням гри. Вони пишуть і виконують музику, читають тексти персонажів. Без гідної музики і якісного озвучування, як і без гарної графіки, сучасна гра навряд чи буде успішною. Хоча, звуки і музика, звичайно, не головне в більшості ігор, але, наприклад, пограйте в Need For Speed: Underground 2 без звуків і музики - чимала частина чарівності гри зникне без сліду.

ПИСЬМЕННИКИ

Як правило, рідкісна сучасна гра не побудована навколо якогось сценарію. Особливо сильна роль сценаріїв в рольових, пригодницьких іграх і в іграх змішаних жанрів, однак навіть якщо робота ведеться над черговою FPS-грою - якісні тексти їй не завадять. Всім цим займаються сценаристи, режисери, письменники.

ДИЗАЙНЕРИ РІВНІВ

Коли програмісти, художники, музиканти виконають основну роботу зі створення персонажів гри, ігрових інтер'єрів, звуків, за справу беруться дизайнери рівнів. Якщо навіть гра представляє собою один великий «рівень» без явного розбиття на частини, дизайнерам доводиться чимало попрацювати над створенням ігровий всесвіту.

ТЕСТЕРИ

Гра - це величезна програма, яка перед виходом у світ повинна бути протестована. Тестування займає дуже важливу роль в процесі створення ігор - іноді тестерів набирають з числа добровольців. Охочих взяти участь в попередньому тестуванні гри, як правило, більше, ніж потрібно - багато хто хоче спробувати новинку першими. Якщо в ході тестування виникають помилки, тестер повідомляє про них розробникам.

1.4. ПЕРСПЕКТИВИ ПРОГРАМІСТА-РОЗРОБНИКА КОМП'ЮТЕРНИХ ІГОР

Тепер, коли ми познайомилися з етапами розробки ігор і ігровими професіями, подумаємо про можливості початківців розробників.

Багато починаючі розробники, які хочуть самі розробляти гри, задаються питанням: «Чи може одна людина, або, в крайньому випадку, команда з декількох ентузіастів, створити сьогодні гру, яка принесе розробникам славу і багатство і стане бажаним гостем на ігрових пристроях гравців усього світу «? Це складне питання.

Історія знає безліч прикладів, коли біля витоків будь-якої гри, що стала шалено популярною у всьому світі, стояла невелика група ентузіастів. Наприклад - це всім відомий Tetris, який в практично незмінному вигляді існує вже не одне десятиліття. Або ігрова серія Final Fantasy - перша версія цієї гри з'явилася в кінці 1980-х років, а тепер - це ціла ігрова всесвіт, над новими версіями якої працює величезна кількість людей.

Потрібно визнати, що поодинці зараз набагато складніше, ніж, скажімо, 20 років тому, створити значну гру. Але ігровий бізнес як ніякий інший заснований на інтуїції і фантазії окремих людей - і якщо в вашій голові народиться геніальна ігрова ідея і ви зможете донести її до інших людей - цілком можливо, що світ стане свідком народження чергової великої гри.

ЛЕКЦИЯ 2. ІНСТРУМЕНТАРІЙ РОЗРОБНИКА КОМП'ЮТЕРНИХ ІГОР.

Анотація: У цій лекції ми поговоримо про ігрові ресурсах, про редактори для створення ігрових ресурсів, а так само обговоримо історію, розвиток і особливості застосування XNA Game Studio 2.0

Ключові слова: гра, безлічі, TGA, graphics adapter, TARGA, rasterization, DDS, ATI, cross-platform, Sprite, спрайт, текстура, об'єкт, анімація, слово, texel, voxel, volumetric, стратегія гри, логотип, акронім, NOT , acronym, FAQ, Windows, Visual Studio 2005, visual, інструментарій, Zune, мінімум, Графічний редактор

З чого складається гра: ігрові ресурси

Будь-яка гра складається з безлічі частин. Але серед них можна виділити дві основні. Перша - це ігрові ресурси, і друга - це програмний код.

Ігрові ресурси - це графічні, музичні та інші ресурси, які використовуються для оформлення гри. Розглянемо різні формати, в яких зберігаються ігрові ресурси, з якими можна працювати в XNA Game Studio 2.0.

ГРАФІЧНІ ФАЙЛИ

Графічні файли використовуються для зберігання зображень. Як правило, це - графічні уявлення ігрових об'єктів, які застосовуються в двовимірних або в псевдотривимірного іграх, коли тривимірний вигляд ігрових об'єктів досягається лише художніми засобами, без використання технологій тривимірної графіки. Так само ці файли можуть бути використані для створення елементів оформлення гри і як текстури для накладення на тривимірні моделі.

JPEG

JPEG - це стандарт зберігання файлів зображень, розроблений спеціально для зберігання цифрових фотографій. JPEG розшифровується як Joint Photographic Experts Group - саме так називалася робоча група, яка розробила цей стандарт. Особливістю JPEG є можливість стиснення зображення з втратами якості. Як правило, в JPEG виділяють 10 рівнів стиснення (від 1 до 10), проте ці зображення можна стискати і з більш точним зазначенням рівня стиснення. Чим сильніше стиск - тим сильніше втрати якості. На невисоких рівнях стиснення JPEG-файли практично не містять так званих артефактів стиснення. У випадку з JPEG це проявляється в помітному спотворенні зображення, особливо - що містить чіткі межі між квітами, чіткі лінії. Справа в тому, що при стисненні за алгоритмом JPEG зображення розбивається на фрагменти 8x8 пікселів. Після цього колірна інформація про пікселі одного квадрата стискається, яркостная ж інформація залишається в більш зберіганню вигляді.

Як правило, в JPEG зберігають різні двовимірні зображення, які можна використовувати для організації елементів інтерфейсу користувача, для створення різних екранів гри (екран вітання, екрани з інформацією про набраних очках і так далі). В JPEG можна зберігати зображення (їх називають спрайтами), які використовуються в двовимірних

іграх в якості ігрових об'єктів (переважно - прямокутної форми), в якості фонових зображень.

Найкраще використовувати JPEG для зберігання кольорових зображень з плавними колірними переходами. Зокрема, це можуть бути фотографії, намальовані ігрові об'єкти і так далі.

JPEG-файли мають розширення JPG або JPEG.

Для створення JPEG-файлів з успіхом можна використовувати популярні графічні редактори - такі, як Adobe Photoshop різних версій.

PNG

Формат PNG (Portable Network Graphics) відрізняється від формату JPG тим, що використовує алгоритми стиснення, що стискають зображення без втрати якості. В результаті виявляється, що зображення, які мають чіткі колірні переходи (такі, як схеми, графіки, зображення для простих елементів управління) можуть бути досить сильно стиснуті в PNG-файлах. Якщо зображення має плавні колірні переходи - кращим вибором для його стиснення буде JPG, хоча і PNG цілком можна використовувати. Як і JPG, даний формат підтримує повнокольорові зображення, проте зображення можна створити з використанням декількох фіксованих кольорів, що йде на користь розміром графічного файлу, і, природно, при правильному підборі кольорів, не позначається на його якості.

Формат PNG був розроблений для заміни застарілого, але все ще популярного формату GIF. Основне призначення GIF-файлів - проста інтернет-графіка.

PNG-файли мають розширення .PNG. Для їх створення можна використовувати растрові графічні редактори, такі, як Adobe Photoshop.

Ще одне важливе особливість формату PNG - підтримка прозорості. Тобто при створенні файлу можна вказати, які з пікселів зображення вважати прозорими. В результаті PNG ідеально підходить для зберігання графічного представлення двовимірних ігрових об'єктів складної форми.

TGA

TGA (Truevision Graphic Adapter), TARGA - Truevision Advanced Raster Graphics Adapter - це графічний формат, який використовується переважно для зберігання ігрових текстур. Цей формат був спеціально створений для зберігання текстур, він використовується в багатьох существущую іграх саме для цих цілей. Він підтримує повнокольорові зображення (8, 16, 24, 32-бітові), підтримують 8-ми бітний альфа-канал для вказівки прозорих ділянок зображення.

Текстура - це растрове (точкове) зображення, яке накладається на поверхню полігону, що є частиною тривимірної моделі. Текстури дозволяють надавати тривимірним моделям кольору, створювати видимість рельєфу, дрібних деталей моделі. При проектуванні 3D-моделі створення дрібних деталей зазвичай виявляється занадто ресурсномістких, використання ж текстур дозволяє отримувати привабливо виглядають моделі,

побудовані з використанням порівняно невеликої кількості тривимірних елементів. Сьогодні тривимірні моделі створюються з високим рівнем деталізації - потужності сучасних комп'ютерів вистачає для реалістичною обробки таких моделей. В іграх минулих років моделі зазвичай виглядають схематично (наприклад, профіль колеса автомобіля в іграх, які використовують нізкополігональних моделі, може виглядати як багатокутник, але не як окружність), проте текстурирование дозволяє навіть таким моделям виглядати досить реалістично.

Текстури розрізняються глибиною кольору і дозволом - чим вище і те й інше - тим вище якість зображення. У застосуванні до текстур існує таке поняття, як тексель - число пікселів, що припадає на мінімальну одиницю текстури.

TGA-файли мають розширення .TGA

Для створення TGA-файлів можна застосовувати всі той же Adobe Photoshop та інші редактори. Наприклад, безкоштовний редактор Paint.NET (http://www.getpaint.net/)

DDS

DDS (Direct Draw Surface) - це графічний формат, розроблений спеціально для використання в DirectX SDK (http://msdn.microsoft.com/en-us/library/bb943990(VS.85).aspx). Він призначений переважно для зберігання текстур. Завдяки особливостям формату - підтримки збереження зріджених і незжатих текстур, апаратної підтримки, цей формат ідеальний для зберігання ігрових текстур.

DDS-файли мають розширення DDS. Для їх створення можна використовувати плагіни для звичайних графічних редакторів, для роботи з такими файлами створені спеціальні набори утиліт і плагінів. Наприклад, це NVIDIA Texture Tools (http://developer.nvidia.com/object/nv_texture_tools.html), The Compressonator від ATI / AMD (http://developer.amd.com/gpu/compressonator/Pages/default.aspx). Плагін для редагування DDS-файлів в Photoshop можна завантажити на http://developer.nvidia.com/object/photoshop_dds_plugins.html).

BMP

BMP (Bitmap) - це графічний формат, який зберігає зображення в стислому вигляді. Він підтримує алгоритм стиснення, який, проте, використовується досить рідко. Створювати BMP-файли можна практично у всіх растрових редакторах. Особливість BMP - широке поширення при створенні графічних інтерфейсів користувача в різних Windows-програмах. BMP-файли зазвичай мають порівняно великий обсяг.

ФАЙЛИ ТРИВИМІРНИХ МОДЕЛЕЙ

Тривимірні моделі використовуються в тривимірних, об'ємних іграх. Для додання їм реалістичного виду, на такі моделі зазвичай накладають файли текстур.

FBX, X

FBX (Autodesk FBX), - це універсальний формат для зберігання тривимірних моделей і супутньої інформації. Він використовується в безлічі тривимірних додатків, зокрема,

вельми популярний в комп'ютерних іграх.

Існують плагіни для популярних програм створення тривимірної графіки, що дозволяють конвертувати створені в них моделі в цей формат. Знайти ПО для роботи з FBX-файлами можна на http://www.autodesk.ru/

Формат X - це формат тривимірних файлів, який використовується DirectX.

Файли шрифтів

SPRITEFONT

Це xml-файл з настройками шрифту. Він містить інструкції, що стосуються візуалізації шрифту в ігровому вікні. Файли цього типу мають розширення .SPRITEFONT.

ЗВУКОВІ ФАЙЛИ

XAP

XNA має спеціальний звуковий редактор - XACT - Microsoft Cross-Platform Audio Creation Tool. Це - основа XNA Framework Audio API. Редактор дозволяє створювати файли у форматі .XAP. Основою для цих файлів є стандартні звукові .WAV-файли. У XACT здійснюються операції по підготовці звуків перед імпортом їх в гру. Після того, як .XAP-проект імпортований в гру, звуками, включеними в нього, можна управляти. Зокрема, звуки можна відтворювати, впорядковувати параметрами відтворення, звуковими ефектами.

ФАЙЛИ ЕФЕКТІВ

FX

FX-файли - це файли ефектів, які застосовуються при рендерінге тривимірних зображень. У форматі FX зберігаються шейдерниє програми. Для розробки FX-файлів можна застосовувати програму FX Composer від nVidia (на http://developer.nvidia.com/object/ fx_composer_home.html).

Всі перераховані вище формати файлів - це ігрові ресурси, які можуть бути включені в XNA-проект.

Ігрова термінологія

Ігрова термінологія складається під сильним впливом англомовних назв. Часто в українській мові досить складно знайти гарну аналогію прийнятим англійським термінам, тому в сфері ігрової термінології існує подвійність - досить часто творці ігор користуються англомовними термінами в той час, як і російськомовні назви так само знаходять застосування. Розглянемо деякі терміни, які використовуються в ігровій індустрії.

Sprite - цей термін часто замінюють україномовним неологізмом спрайт - словник Lingvo 12 визначає поняття «спрайт» як «елемент динамічного графічного відображення». В ігровій індустрії синонімами слова спрайт є такі слова, як «зображення», «картинка», іноді користуються словом «текстура», проте зазвичай це поняття має дещо інше смислове навантаження. Як правило спрайт - це двовимірне зображення, причому, у

вузькому сенсі слова це - лише зображення, а в широкому - це ігровий об'єкт, який має набагато більш широким набором можливостей, ніж звичайне зображення. Спрайт мають прямокутну форму, проте в комп'ютерних іграх часто зустрічаються непрямокутні об'єкти. Це досягається за рахунок здавна прозорих областей при малюванні спрайту. Термін «анімований спрайт» відноситься до спрайту, який виводиться з використанням анімації, що створює ілюзію руху, переміщення будь-яких частин зображення - рух рук і ніг персонажа при переміщенні, рух коліс автомобіля, лопатей пропелерів літака і т.д. Анімація зазвичай реалізується почергової зміною декількох статичних зображень, спеціально підготовлених для того, щоб створити ілюзію руху.

Texture - текстура. Зазвичай текстурами називають двовимірні зображення, які «накладають» на тривимірні моделі. У термінології XNA поняття текстури і спрайту при розмові про двовимірних ігрових об'єктах збігається.

Background - фон. Так називається зображення, зазвичай - відповідне розмірам ігрового поля, яке є фоном для інших зображень. Фон може бути нерухомим і рухомим. Рухомий фон (scrolling background) використовується в іграх, які називаються Скролери (scrollers). Скролінг - це один з прийнятих ігрових термінів. Він означає прокрутку, переміщення вмісту вікна. Скроллінговие гри надзвичайно поширені серед двовимірних ігор. Наприклад, використання скроллінгового фону дозволяє створити ілюзію руху в двовимірному гоночному симуляторі. Фон в двовимірної грі може складатися з декількох частин, що рухаються з різною швидкістю - це дозволяє створити ефект тривимірності ігрового світу.

2D-game - двовимірна гра - гра, в якій використані двовимірні зображення.

3D-game - тривимірна гра - використовує тривимірні моделі і тривимірний ігровий світ.

Tile - тайл - невелике зображення, яке використовується для конструювання рівнів в іграх. Tile можна перевести як «мозаїка» або «черепиця».

Polygone (полігон, багатокутник) - просторовий багатокутник, який використовується для створення тривимірних об'єктів. Як правило, в комп'ютерній графіці використовуються трикутники

Pixel (піксель) - найменший елемент растрового зображення, точка, яка відображається на екрані. Зазвичай в пікселях вимірюють дозвіл текстур (наприклад - 1024x768), екранне дозвіл монітора, розміри ігрових вікон. Слово Pixel - це аббревітура від Picture's ELement.

Texel (тексель) - точка текстури в тривимірному пространстве.Слово Texel - це скорочення від Texture Element.

Voxel (воксель) - точка тривимірного зображення. Це слово - абревіатура від Volumetric Pixel - об'ємний піксель.

Texture Filtering (фільтрація текстур) - зменшення спотворень при накладенні текстур на тривимірний об'єкт.

Camera (камера) - так називають точку в ігровому просторі, з якою гравець бачить ігровий світ. З точки зору положення камери гри можна поділити на ігри від першої

особи (камера розташована так, що реалізує вид як би «з очей» персонажа), ігри з видом зліва-зверху (стратегії), ігри з видом ззаду - камера розташовується зазвичай позаду гравця і трохи вище його. Існують і гри, де камера може приймати різні позиції, наприклад, реалізують вид з очей персонажа і вид ззаду

Transparency (прозорість) - прозорими можуть бути частини двовимірних зображень - це дозволяє створювати зображення складної форми, які, фактично, обмежені прямокутником. Прозорими бувають і об'єкти в тривимірному ігровому світі - наприклад - це може бути прозора вода або скло, певним чином заломлюють світло.

Light Model (модель освітлення) - способи моделювання освітлення об'єктів.

ОГЛЯД XNA GAME STUDIO 2.0. - ІСТОРІЯ, РОЗВИТОК, ОСОБЛИВОСТІ ЗАСТОСУВАННЯ

На рис. 2.1. ви можете бачити логотип XNA. Зверніть увагу на те, що одна з ліній, складових букву X, побудована з двох рисок і двох точок. Код Морзе визначає -..- як X, -. - як N, .- - як A. XNA - це не акронім (XNA's Not Acronymed) - саме так в FAQ (http://msdn.microsoft.com/en-us/directx/aa937793.aspx), присвяченому XNA, представники Microsoft відповідають на питання про те, що ж означає термін XNA.

Рис. 2.1. Логотип XNA

XNA Game Studio - це середовище для розробки комп'ютерних ігор, які можуть працювати на платформах Windows і Xbox 360. Ми будемо працювати з XNA Game Studio 2.0. Ця версія середовища розробки сумісна з Visual Studio 2005 і Visual C # 2005 Express. Фактично, XNA Game Studio 2.0. - це набір бібліотек (XNA Framework) і деяких спеціальних інструментів, призначених для створення ігор. Причому, робота з програмування гри ведеться на мові C # або в середовищі Visual C # 2005 Express, або в середовищі Visual Studio 2005.

XNA підходить для різних категорій творців ігор, яких цікавить розробка ігор для Windows і Xbox 360. Справа в тому, що весь необхідний інструментарій - XNA Game Studio 2.0. і Visual C # 2005 Express, а так само - документацію до цих продуктів - можна безкоштовно завантажити з сайту Microsoft. Нижче наведені посилання, за якими можна знайти дистрибутиви продуктів:

- http://www.microsoft.com/express/2005/
- http://www.microsoft.com/xna

Після скачування дистрибутивів досить встановити Visual C # 2005 Express, після чого - XNA. В результаті, ви зможете створювати XNA-проекти в Visual C # (рис. 2.2.).

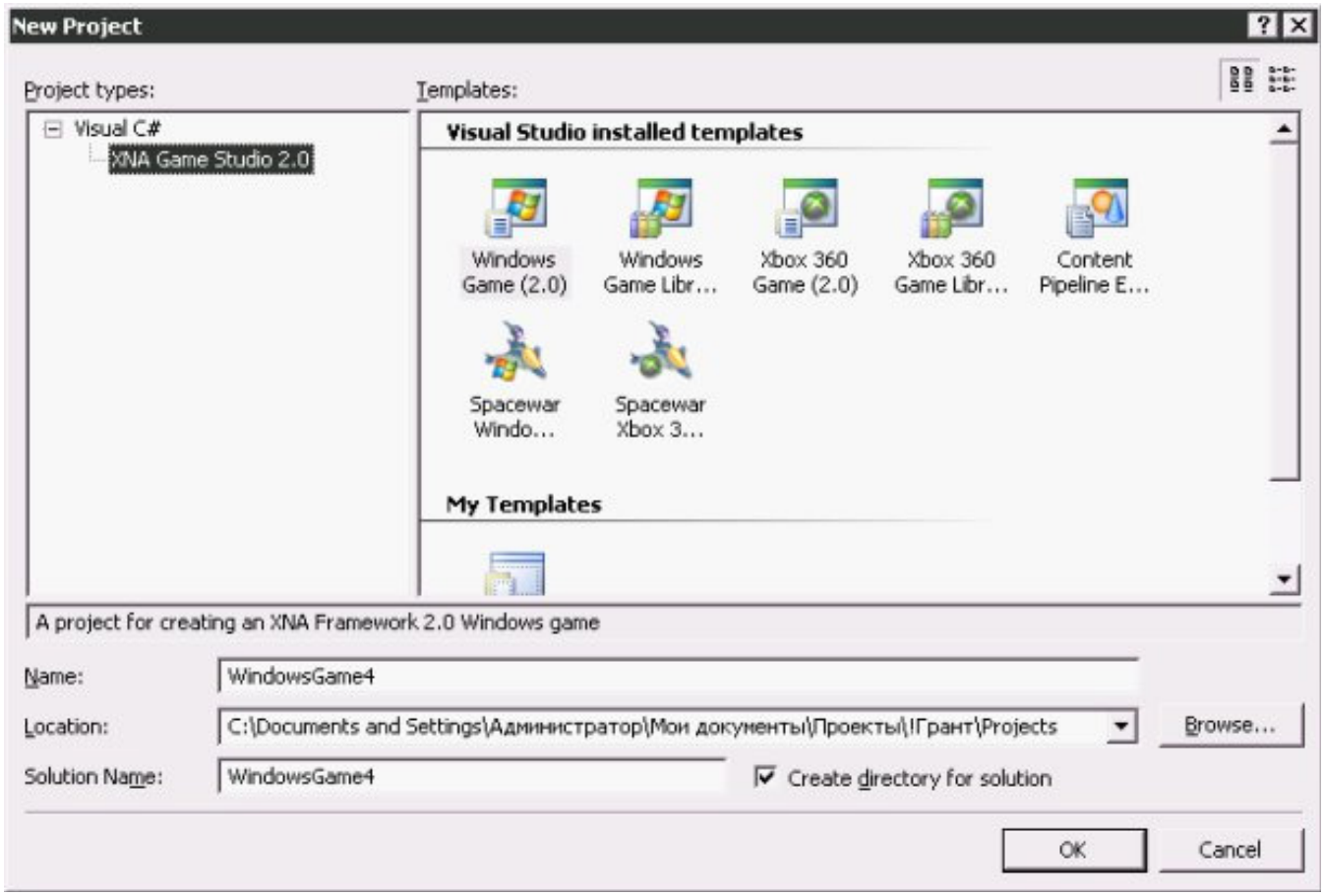

Рис. 2.2. Ігрові XNA-проекти у Visual C#

ІСТОРІЯ І РОЗВИТОК.

XNA Professional и XNA Express

Перша версія XNA вийшла в двох варіантах в 2006 році. Це були XNA Professional і XNA Express. XNA Express була призначена для аматорського використання спільно із середовищем розробки Visual C # 2005 Express Edition, а версія Professional - для професійних розробників, що працюють в Visual Studio 2005.

XNA Game Studio 2.0.

XNA 2.0. була анонсована в серпні 2007 року. Вона поєднує в собі можливість роботи в Visual C # 2005 Express Edition і в Visual Studio 2005, об'єднуючи можливості двох попередніх варіантів. Крім того, багато компонентів XNA зазнали поліпшення.

XNA Game Studio 3.0

У 2008 році була анонсована XNA Game Studio 3.0. Її головна особливість в тому, що ця версія XNA підтримує створення ігор для портативного медіаплеєра Microsoft Zune. XNA 3.0.

Подробиці про XNA Game Studio 2.0.

XNA складається з декількох ключових компонентів. Зокрема, це XNA Framework, Content Pipeline, XACT.

XNA Framework

XNA Framework - це набір бібліотек, які містять класи, необхідні при розробці гри.

Microsoft.Xna.Framework - містить найбільш часто використовувані класи - такі, як таймери і ігрові цикли.

Microsoft.Xna.Framework.Audio - дозволяє завантажувати і програвати музичні фрагменти в грі.

Microsoft.Xna.Framework.Content - містить компоненти Content Pipeline (про Content Pipeline ми поговоримо нижче).

Microsoft.Xna.Framework.Design - дозволяє займатися конверсією типів даних.

Microsoft.Xna.Framework.GamerServices - містить класи, що забезпечують різні функції для взаємодії з гравцем. Наприклад - зберігання даних грає.

Microsoft.Xna.Framework.Graphics - робота з графікою - обробка і виведення зображень.

Microsoft.Xna.Framework.Graphics.PackedVector - містить спеціальні типи даних, кількість бітів яких не кратно 8.

Microsoft.Xna.Framework.Input - підтримує роботу з пристроями введення. Це - миша, клавіатура і ігровий маніпулятор (Xbox 360 Controller)

Microsoft.Xna.Framework.Net - класи, які підтримують створення багатокористувацьких мережевих ігор.

Microsoft.Xna.Framework.Storage - класи для роботи з файлами.

CONTENT PIPELINE

Content Pipeline (конвеєр контенту) призначений для уніфікації включення в гру різних ігрових ресурсів. Він дозволяє автоматично конвертувати 2-х і 3-х мірні графічні ресурси, звуки (після попередньої обробки в XACT) в формат, який можна використовувати для Windows-ігор і для Xbox-ігор.

Content Pipeline - це дуже важлива особливість XNA, так як він дозволяє зняти з розробника завдання підготовки контенту різного формату для гри - все, що потрібно зробити для додавання ресурсу в гру - завантажити його в ігровий проект за допомогою засобів, що подаються для цього XNA Framework. Операції по перетворенню файлів різного типу в формат, який підходить для використання в грі, проводиться автоматично.

Дані в форматах, які виходять на виході різних програм по створенню контенту, обробляються імпортерами і перетворюються в формат

Цим процесом можна керувати, створювати власні контент-імпортери (content importers) і контент-процесори (content processor). Контент-імпортери перетворять ігрові ресурси в формат XNA Game Studio Content DOM (document object model). Далі, контент-процесори перетворюють дані з DOM-формату в формат, придатний для компіляції і використанні в ігрових проектах. XNA поставляється з великим набором стандартних імпортерів і процесорів, при необхідності цей набір може бути розширений.

XACT

XACT - Microsoft Cross-Platform Audio Tool - це інструмент для обробки звуку, придатного для включення в XNA-проекти. XACT - це звуковий редактор, який дозволяє створювати .XAP-проекти, що включають в себе звуки і параметри налаштування звучання. Звуки з XAP-проектів можна відтворювати в іграх, керуючи їх відтворенням. Наприклад, можна включати і вимикати відтворення звуків, управляти їх гучністю, застосовувати до них різні ефекти і т.д.

Застосування XACT направлено на уніфікацію озвучування Windows-ігор та ігор для Xbox 360.

ЗАВДАННЯ

Якщо ви займаєтеся освоєнням цього курсу самостійно - зараз саме час завантажити і встановити програми, які знадобляться вам при розробці ваших власних ігор. Як ми вже говорили, XNA Game Studio 2.0. і Visual C # 2005 Express, а так само - документація до них - це безкоштовні продукти. Однак, для роботи вам знадобляться, як мінімум, графічний редактор, редактор тривимірних моделей. Деякі з подібних програм безкоштовні, деякі ж вам доведеться придбати.

ЛЕКЦІЯ 3. ПСИХОЛОГІЯ КОМП'ЮТЕРНИХ ІГОР.

Анотація: У цій лекції ми поговоримо про жанрах комп'ютерних ігор, про те, як зробити гру цікавою. Жанри комп'ютерних ігор, з одного боку, мають досить чіткі межі, але з іншого - нерідко досить складно класифікувати ту чи іншу гру в межах якогось одного жанру. У деяких іграх жанри переплітаються, деякі ігри створюють власні жанри і викликають наслідування. Розглянемо жанри комп'ютерних ігор через аналіз так званих культових ігор.

Ключові слова: ПО, PAC, halo, duty, shareware, realm, survivability, resident, RPG, fallout, 'fantasy', quest, entertainment system, puck, empirical, e-entertainment, craft, online gaming, MMO, PVP, flight simulation, COLD, freelancer, bowl, knockout, bass, MOTOS, surfer, CAVE, hack, slash, гра, симулятор, користувач, програма

ЖАНРИ КОМП'ЮТЕРНИХ ІГОР, АНАЛІЗ ПРОВІДНИХ ПРЕДСТАВНИКІВ ЖАНРІВ

Серед ігор всіх часів виділяються так звані культові гри. Зазвичай ці ігри набувають мільйони шанувальників, творці таких ігор нерідко стають мільйонерами. Як правило, ігри такого роду дуже повільно старіють - наприклад, сьогодні багато людей із задоволенням грають в ігри десятирічної давності.

Ігри еволюціонували разом з апаратними можливостями ігрових пристроїв, технологіями програмування, новими ідеями. Не можна сказати, що успіх гри залежить лише від її графічної складової, від майстерності програмістів, або від вдалої ідеї. Але, як правило, ігри, які можна віднести до розряду культових - гри, в які грали і грають мільйони людей по всьому світу, поєднують в собі все перераховане і ще щось таке, що не піддається опису.

Зараз ми розглянемо різні ігри, які хвилювали і хвилюють геймерів усього світу, а заодно простежимо еволюцію цих ігор.

КУЛЬТОВІ ІГРИ: ЯКІ ВОНИ?

Яку гру можна вважати культовою? На це питання можна дивитися з різних точок зору. Якщо розглядати «культовість» гри в рамках окремої людини - геймера, то це - гра, в яку любить грати ця людина - незалежно від того, чи грає в неї ще хтось крім нього. Це дуже вузький підхід, але, як правило, якщо гра дуже подобається комусь, то в світі знайдеться ще кілька мільйонів чоловік, яким вона так само до душі.

Існує статистика кількості проданих копій відеоігор - по ній можна дуже точно зрозуміти, які ігри подобаються людям - причому, ця статистика не включає в себе піратські копії (піратство - дуже поширене явище, особливо - серед комп'ютерних програм) - мабуть, за найкращими ігор кількість гравців може в кілька разів перевищувати кількість проданих примірників. У той же час є ігри, які популярні лише у вузьких колах геймерів, однак ці ігри так само можна віднести до розряду культових.

Кращі ігри не старіють - взяти той же Pac Man, який вийшов більш 25 років тому - в

нього грали раніше і продовжують грати сьогодні. Деякі ігри витісняються з масової свідомості геймерів новими, проте той, хто колись «підсів» на якусь гру, буквально зжився з нею після довгих місяців (або років) - він навряд чи забуде її, навіть якщо на прилавках з'явиться програма, на десятиліття випереджає його улюблену іграшку.

Швидше за все, такий геймер все ж буде грати в нові ігри, але стару він не залишить ніколи, продовжуючи таким чином дні її життя. В якості яскравого прикладу такої гри можна назвати Duke Nukem 3D.

Як правило, найбільш успішні гри і їх продовження портируют на різні платформи (та й про емуляторах забувати не варто - з їх допомогою на звичайному ПК або КПК ви зможете пограти практично в усі кращі ігри для будь-яких ігрових пристроїв), тому ми не будемо детально зупинятися на те, для якого пристрою була написана та чи інша гра.

Зараз ми розглянемо основні види ігор і найбільш значущих представників кожного виду.

PLATFORMERS

Platformers (платформні гри або платформери) це ігри, де герой подорожує по ігровому світу, переходячи з платформи на платформу - стрибаючи над прірвами, перепливаючи річки, піднімаючись на стіни і попутно борючись зі злочинцями. Спочатку платформери - це двовимірні ігри з видом збоку. Наприклад, до таких ігор належать Sonic the Hedgehog і Mario.

У той же час, є і тривимірні платформери - наприклад - відома гра Tomb Raider, та й тривимірні Sonic Adventures і Mario 64 теж популярні.

Серед відомих платформер можна відзначити такі ігри, як Super Mario Bros., Metroid, Sonic the Hedgehog, Mega Man, Donkey Kong, Contra, Lode Runner, Spyro the Dragon, Prince of Persia.

Еволюція культових платформер йшла по шляху вдосконалення графіки і розширення ігрових сюжетів завдяки новим можливостям обладнання.

MARIO

Mario - герой однойменної гри-платформера - це талісман компанії Nintendo вже більше 25 років. Перша гра за участю легендарного Mario з'явилася в 1981 році - вона називалася Donkey Kong і Mario виступав в ній в якості героя, що бореться проти головного лиходія гри - Donkey Kong'а. Персонаж виявився цікавим - і в 1983 році з'явилася гра Mario Bros. - там Mario вперше виступав в якості головного героя. З тих пір і до нашого часу Mario регулярно з'являється в нових іграх Nintendo. Наприклад, одна з останніх версія гри з Mario - це Super Paper Mario для консолі Wii.

Всього, за статистикою, продано понад 190 мільйонів копій ігор за участю Mario. Серія цих ігор є однією з найбільш успішних серій, а сам Mario став одним із найбільш упізнаваних ігрових персонажів в світі. У 1993 році вийшов перший фільм (Super Mario Bros.), заснований на відеогрі - головним героєм фільму став, як ви вже, напевно, здогадалися - Mario.

Ще одна популярна гра в тому ж платформенном стилі - це Sonic the Hedgehog.

SONIC THE HEDGEHOG

Перша гра Sonic the Hedgehog була випущена для ігрової приставки Sega Mega Drive в 1991 році. З тих пір їжачок Sonic став одним з найвідоміших персонажів фірми Sega. Ігри за участю Sonic'а виходять і по сей день - наприклад - остання гра серії Sonic and the Secret Rings для ігрової консолі Wii, яка випущена в 2007 році.

Перша гра серії, по всій видимості, здобула популярність через унікальних ігрових характеристик героя і ігрового світу. Зокрема, вона додала в платформні ігри поняття швидкості - гра стала динамічніше. Герой міг атакувати ворогів новим способом - стикаючись з ними. До цього були поширені інші методи атаки - стрілянина у ворогів, стрибки їм на голову. Зіткнення героя з ворогом зазвичай означало втрату одного з життів. А їжачок міг згортатися і котитися, збиваючи всіх на своєму шляху. В процесі гри персонаж збирав золоті каблучки, які поряд з самим Sonic'ом стали однією з пам'ятних деталей гри.

Тепер на черзі гри, які сьогодні знаходяться на піку популярності.

ACTION

Action (в російській варіанті цей жанр називається стрілялки) - це ігровий жанр, який цікавий тим, що гравець в ході ігрового процесу повинен постійно щось робити. Найбільш популярне заняття в таких іграх - це знищення ворогів. А найбільш популярним видом екшенів стали так звані шутери, або, по-українськи - стрілянина.

SHOOTERS

Шутери стали популярним вже дуже давно. Типовий сценарій такої гри простий - ви керуєте якимось механізмом (або героєм), блукаєте по ігровому світу (або переміщується в обмежених межах) і знищуєте орди ворогів.

SPACE INVADERS

Гра Space Invaders, яка відноситься до жанру Scrolling Shooters (скроллінговие шутери) була розроблена в 1978 році компанією Taito Corporations. Існує безліч варіацій цієї гри - зокрема, одна з них вона відома під назвою Galaxian. В основному саме Galaxian (вона з'явилася в 1979) стала прообразом всіх ігор на тему Space Invaders.

Сюжет гри простий - ви керуєте лазерною гарматою і відбиваєтеся від атаки космічних прибульців, які хочуть висадитися на Землю. Чим більше прибульців ви знищите - тим швидше вони будуть рухатися і тим нижче з'являтися. До того ж шкідливі прибульці бомблять гравця, а іноді (правда, це відноситься не до класичної Space Invaders, а до Galaxian) навіть, як камікадзе, кидаються на нього щоб ціною свого життя знищити захисника планети.

Незважаючи на простоту, гра, практично відразу після виходу, буквально завоювала світ. Є думка, що це сталося почасти тому, що в той час багато хто був захоплені ідеєю науково-фантастічекой епопеї «Зоряні війни» (Star Wars), і вчасно з'явилася гра на

подібну тему дозволяла всім бажаючим відчути себе в гущі «зіркових» подій.

Величезна популярність Space Invaders привела, по-перше, до того, що розробники зрозуміли, наскільки прибутковою і успішною може бути гра (це підстьобнуло розробку нових ігор), а так само до перших виступів противників відеоігор, які говорили про згубний вплив ігор на дитячу психіку .

У Space Invaders була одна цікава перевага перед іншими іграми - гра закінчувалася не по закінченні певного часу, як це було з іншими іграми, а лише після того, як гравець втрачав все життя. З урахуванням того, що в ті часи основним ігровим пристроєм був платний ігровий автомат, це було суттєвою перевагою. Є статистика рекордів, набраних в цій грі - максимальний рекорд в 1114020 очок був поставлений за 38 годин 30 хвилин гри.

FIRST PERSON SHOOTERS

FPS – или First Person Shooters – то есть «стрелялки от первого лица» - это игры, которые уже

FPS - або First Person Shooters - тобто «стрілялки від першої особи» - це ігри, які вже багато років знаходяться в стані постійного вдосконалення і розвитку. Як правило, в такій грі популярний вид з очей героя - ви бачите ігровий світ так, як бачить його герой.

Тривимірний світ в таких іграх дуже важливий - тому перші помітні FPS з'явилися на початку 1990-х років з розвитком тривимірної ігрової графіки. Зокрема, першою помітною грою в такому стилі став - Wolfenstein 3D. Серед інших відомих 3D FPS можна відзначити Doom, Duke Nukem 3D, Quake, Half-Life, Unreal, Far Cry, Counter-Strike, Postal, Halo, Medal of Honor, Call of Duty, Battlefield.

DOOM

До 1996 року всі 3D FPS-гри називали просто: «Doom-like» - тобто ігри, зроблені за типом Doom'а. Цю гру іноді називають грою №1 усіх часів і народів. Дійсно, принципи, закладені Doom'ом, актуальні і сьогодні. До речі, і сам Doom все ще живий - остання версія гри - Doom 3 славиться не тільки захоплюючим геймплеєм, але і високою вимогливістю до ресурсів системи.

Doom спочатку випускався як комп'ютерна гра - його перша версія, створена id Software, побачила світ у 1993 році. Вона мала всі ознаки сучасних FPS - тому нескладно зрозуміти, чому гра викликала такий неймовірний інтерес у ті далекі часи. Мабуть, популярності Doom'а сприяв і той факт, що гра поширювалася на умовах Shareware (тобто безкоштовно) - за два роки її скопіювали близько 10 мільйонів чоловік. До того ж, Doom дозволяв створювати призначені для користувача розширення, підтримував мережеву багато користувачів гру.

POSTAL

Ігри, подібні Doom, викликали безліч критики - причиною тому стала жорстокість таких ігор, яка, на думку критикують, може спровокувати жорстокість гравців в реальному світі. До речі, серед FPS-ігор є чимало ігор, заборонених в деяких країнах. Як правило,

агресія в більшості ігор спрямована на негативних персонажів, але деякі ігри «вивертають навиворіт» ідею доброго героя, який бореться з монстрами - наприклад, серія ігор Postal, де вам пропонується грати роль маніяка, який полює за мирними жителями.

Зовсім інша справа - Duke Nukem 3D там герой чесно відпрацьовує свою роль, знищуючи полчища чудовиськ з усіх видів зброї.

DUKE NUKEM 3D

Гра була випущена компанією 3D Realms в 1996 році. Duke Nukem 3D була заснована на більш ранніх двомірних іграх - Duke Nukem і Duke Nukem II. 3D-версія гри могла похвалитися гідною графікою, відмінними саунд-треками (навіть сьогодні нечасто зустрінеш гру, кожен рівень якої забезпечений унікальною музичною композицією), таким, що запам'ятовується головним героєм.

Розробники Duke Nukem 3D були не позбавлені почуття гумору - це серйозно позначилося на їхній грі і її персонажі - від нього періодично чуються різні цікаві репліки, іноді в грі зустрічаються об'єкти, які якимось чином зачіпають компаній-конкурентів (наприклад, підірваний в одному з рівнів гри будівля зроблено схожим на штаб-квартиру id Software).

В процесі гри герою доводиться не тільки знищувати ворогів, але і вирішувати різні головоломки, зазвичай полягають в пошуку карток для відкривання будь-яких дверей або механізмів, що дають можливість продовжувати гру.

Що цікаво, Duke Nukem 3D досі не має гідного сучасного продовження (за винятком оновлень, портів та інших подібних речей). З 1997 року 3D Realms розробляє гру Duke Nukem Forever, випуск якої геймери всього світу чекають вже кілька років, але до цих пір компанія на всі питання про терміни появи нової гри відповідає: «коли буде готова, тоді і вийде» (в даний час гра вже вийшла).

HALF-LIFE

Гра Half-Life була створена в 1998 році компанією Valve Software (сьогодні вона еволюціонувала до Half-Life 2). Відмітна особливість гри - це безперервний ігровий процес - вона не розбита на рівні. В ході гри герой не тільки знищує ворогів - він ще й вирішує різні головоломки.

Half-Life знаменита, крім усього іншого, модами - тобто іграми, зробленими на її основі. Фактично, мод гри змінює зовнішній вигляд об'єктів, героїв, додає нову зброю - причому, все це робиться на графічному движку (тобто з використанням базових графічних можливостей) вихідної гри.

Мабуть, одним з найвідоміших і популярних Half-Life-модів стала розрахована на багато користувачів мережева командна гра Counter Strike (випущена в 1999 році). Гравці, розділені на команди, змагаються на різних картах - грати можна як на онлайнових серверах, так і в локальних мережах.

Наступний стиль, на якому ми зупинимося, схожий на FPS, однак гри в цьому стилі зазвичай похмуріше ніж традиційні стрілянина.

SURVIVAL HORROR

Ігри стилю Survival Horror (що на російський зазвичай перекладається як «жахи», іноді - як «гри на виживання») - це ігри, як правило, мають властивості FPS-ігор, проте крім звичайних для битв містять елементи жанру жахів. Гравця оточує похмура атмосфера, монстри, він потрапляє у важкі халепи, які повинні тримати його в постійній напрузі - власне кажучи - як і будь-який хороший твір в жанрі жахів.

Основи Survival Horror заклала гра Alone in the Dark, популярність ігор цього стилю принесла серія ігор Resident Evil, зокрема, вперше словосполучення Survival Horror з'явилося саме в першій версії Resident Evil. Серед інших помітних ігор цього жанру можна відзначити Silent Hill - порівняно недавно вийшов фільм, заснований на ідеях цієї гри - фанати гри відзначають, що екранізація вийшла гідною.

FIGHTING

Ігри в стилі Fighting, або, по-українськи, бійки (іноді для позначення цих ігор застосовують пряму кальку з англійської - файтинг) - це ігри, де ви вступаєте в сутичку з віртуальним противником, який або управляється комп'ютером, або - людиною. Цей жанр виник в середині 1980-х років і популярний досі.

Серед помітних представників жанру можна відзначити Mortal Kombat, Street Fighter, Super Smash Bros., King of Fighters, Soul Calibur, Dead or Alive, Virtua Fighter.

VIRTUA FIGHTER

Гра Virtua Fighter, випущена компанією Sega в 1993, році вважається родоначальницею всіх сучасних тривимірних бійцівських ігор - вона була першою такою грою.

В процесі гри вам треба боротися з противниками - для управління персонажем використовується кілька базових прийомів і рухів, яким відповідають окремі кнопки: удар рукою, ногою, блок, пересування - включає в себе стрибки і присідання. Але якби активність гравця в Virtua Fighter обмежувалася лише цими рухами - гра була б занадто нудним - величезна кількість рухів ховається за комбінаціями клавіш.

Різні персонажі гри б'ються в різних стилях - тому одна і та ж комбінація для різних героїв часто відповідає різним прийомам - в результаті виходить незабутня динамічна гра. Навіть старі версії гри все ще виглядають цілком гідно. Наприклад, Virtua Fighter 2, яка була випущена в 1995 році. Гра розвивається до сих пір - остання версія називається Virtua Fighter 5 - вона випущена для ігрових консолей PlayStation 3 і Xbox 360.

BEAT 'EM UP

Ще один стиль ігор - це суміш файтинга і платформер - в англійському варіанті вони називаються Beat 'em up. Як правило, мета такої гри - пройти гру, знищивши всіх ворогів. Герой гри зазвичай користується якою-небудь зброєю, однак система ударів і рухів в таких іграх зазвичай проста - кілька базових ударів, плюс як мінімум один «суперудар», який позвлояет «розкидати» цілу юрбу ворогів. Ці ігри були особливо популярні в 1990-х роках.

Серед помітних ігор цього типу можна відзначити Double Dragon, Battle Toads, Streets of Rage, Golden Axe, Dynasty Warriors, Viewtiful Joe, The Warriors, Samurai Warriors.

RPG

RPG - Role Playing Game - рольові ігри -ігри, де ви повинні відігравати роль якого-небудь персонажа. Перші рольові ігри виникли завдяки автоматизації класичних рольових ігор, в які грали без участі комп'ютерів - популярність таких ігор визначила популярність комп'ютерних RPG. Зазвичай події в таких іграх розвиваються хід за ходом, в сучасних RPG є епізоди (бої, наприклад), які передбачають гру в реальному часі - можна сказати, що RPG еволюціонують в бік FPS.

Серед помітних RPG можна відзначити Planescape: Torment, Baldur's Gate, The Elder Scrolls, Fallout, Final Fantasy, Ultima, Pokemon, Dragon Quest, Neverwinter Nights.

FINAL FANTASY

Історія цієї гри почалася в 1986 році. Компанія Square, справи якої після випуску декількох продуктів йшли не дуже добре, зважилася на сміливий крок - гру, яка або стане останньою, або дасть компанії шанс на виживання. В результаті Square (вірніше - Square Enix) живе і процвітає досі

Final Fantasy замислювалася як конкурент популярної гри вісімдесятих Dragon Quest, яка була випущена компанією Enix, в результаті ж, в 2003 році компанії Square і Enix об'єдналися в Square Enix Co. - і гри Dragon Quest і Final Fantasy тепер виходять під маркою Square Enix.

З простої 8-ми бітної RPG для приставки NES (Nintendo Entertainment System) гра перетворилася в цілу імперію. Граючи в сучасну Final Fantasy ви потрапляєте в світ магії і політичних інтриг, любові і битв, відмінної музики, чудової графіки і захоплюючого сюжету. Мабуть, Final Fantasy можна назвати «ожила казкою» - і цим все сказано.

PUZZLE, MAZE

Puzzle (головоломки) - це ігри, які зазвичай не вимагають серйозних графічних можливостей системи - їх головна цінність в ідеї гри. Головоломок існує безліч - серед найбільш відомих можна відзначити такі ігри, як Tetris, Minesweeper, Lemmings, Boulder Dash.

А одна з найстаріших популярних відеоігор Pac Man відноситься до розряду maze (лабіринтових) ігор.

TETRIS

Всім відома комп'ютерна головоломка тетріс була створена нашим співвітчизником Олексієм Пажитновим в 1984 році.

Мабуть, немає потреби говорити про популярність цієї гри або розповідати про її правилах і «дійових осіб» - фігурках, складених з 4-х квадратиків. У неї грав майже кожен. Пройдіться по магазинах - напевно дуже скоро вам пощастить знайти місце, де продають кишенькові «тетріси».

Ця гра існує, мабуть, для всіх ігрових платформ, постійно з'являються нові варіації на тему тетрису (наприклад, онлайновий тетріс, коли грати доводиться проти супротивника-людини, а так само маса інших ігор) - і з часом його популярність не падає.

PAC MAN

Гра була розроблена в 1979 році в Японії компанією Namco, яка і в наш час продовжує випускати нові продукти.

Японська версія гри називалася Puck Man, але коли її готували до виходу в США, назва була змінена для того, щоб гравці ненароком не замінили б в слові «Puck» букву «P» на букву «F».

Концепція гри проста: головний герой, роль якого виконує кульку з відкривається ротом, бігає по ігровому полю - лабіринту і їсть все, що попадається йому на шляху. Pac Man бігає по лабіринту не один - його переслідують кілька привидів. Герой може напасти на привид лише тоді, коли йому пощастить поживитися особливим об'єктом (енерджайзером).

Є кілька думок з приводу виникнення цього героя - по одному з них Pac Man - це щось на кшталт піци з вирізаним шматочком, по іншому - герой гри - це округлений і кілька змінений японський ієрогліф, що має відношення до їжі.

Здавалося б - все дуже просто, навіть примітивно, але гра не втрачає популярність вже дуже багато років, її персонажі періодично з'являються в інших іграх, є пісні, написані під впливом Pac Man, фільми, так чи інакше використовують ідею або персонажів гри.

STRATEGY

RTS - Real Time Strategy - стратегії реального часу - це ігри, де ви, як правило, керуєте будь-якої армією, причому події розвиваються в реальному часі.

Серед таких ігор можна відзначити Warcraft, StarCraft, Command and Conquer, Age of Empires, Total Annihilation, Rise of Nations.

Ще один вид стратегій - Turn Based Strategy (покрокові стратегії) - для них характерно покрокове розвиток подій. Серед них можна відзначити такі ігри, як Civilization, Heroes of Might and Magic, the Advance Wars, Fire Emblem, Shattered Union.

STAR CRAFT

У стратегії реального часу StarCraft є система підказок. Одна з них пропонує заводити будильник для того, щоб ненароком не забути про важливі справи в реальному світі під час гри. Це хороший рада - StarCraft, навіть при проходженні стандартних, вбудованих в гру кампаній, затягує просто неймовірно. Вже не кажучи про мережевій грі.

Гра була випущена в 1998 році компанією Blizzard Entertainment - для свого часу вона відрізнялася чудовою тривимірною графікою, яка актуальна і сьогодні. Секрет графічної складової гри полягав у тому, що насправді гра була двовимірною, однак персонажі були намальовані таким чином, щоб імітувати тривимірність. В результаті

гра відмінно виглядала і відмінно працювала навіть на дуже слабких за сучасними мірками машинах тих часів.

У грі потрібно управляти однією з трьох рас, будувати армію, зміцнювати табір і воювати з противником. Як правило, перемогу в грі здобуває той, хто знищує все, що має відношення до супротивника - «живу силу» і будови.

Матчі в StarCraft проводяться на картах обмеженого розміру, при мережевій грі одночасно можуть грати кілька гравців - карта може бути досить великий для того, щоб їм усім вистачило місця. Типовий ігровий процес проходить так: ви будуєте будівлі, здобуваєте за допомогою робочих юнітів ресурси, на які можна будувати воїнів і нові будівлі. У грі є система розвитку бойових одиниць - для того, щоб побудувати більш досконалого воїна вам доведеться будувати додаткові будівлі, а для поліпшення існуючих юнітів - проводити поліпшення. Коли армія побудована і ваш табір захищений (або навіть раніше) - приходить час атакувати противника або самому захищатися від його атак.

Гра, незважаючи на поважний вік, все ще популярна - в неї грають мільйони геймерів у всьому світі. Особливо StarCraft популярний в Південній Кореї - деякі професійні гравці з цієї країни заробили на різних турнірах по Star Craft сотні тисяч доларів.

HEROES OF MIGHT AND MAGIC

Heroes of Might and Magic - це покрокова стратегія, випущена в 1995 році компанією New World Computing. З тих пір гра постійно розвивалася.

В ході гри ви повинні розвивати вашу імперію - захоплювати нові землі, добувати сировину, збирати артефакти, боротися з ворогами. Гра проходить в покроковому режимі.

Наприклад, бій виглядає так: один навпроти одного коштує ваша армія і армія противника і ви по черзі наносите один одному удари - ви виділяєте один зі своїх юнітів і наказує йому атакувати юніт супротивника - після серії взаємних атак визначається переможець.

Гра не так динамічна, як той же StarCraft, однак вона точно так само затягує гравця - вам доводиться дбати про захист вашого замку, ви подорожуєте по ігровому світу, відкриваючи нові землі і збираючи корисні предмети, вступаєте в сутички з ворогами - все це, приправлене чудовою графікою (принаймні починаючи десь з 3-й версії гри графічна складова Heroes of Might and Magic просто чудова), рідкого любителя стратегій залишить байдужим.

MASSIVELY MULTIPLAYER ONLINE GAMES

Massively Multiplayer Online Games (MMO) - або онлайнові ігрові світи - це ігри, в яких люди в буквальному сенсі живуть день і ніч, підключившись до ігрового сервера через Інтернет. Існує кілька різновидів цих ігор. Перша скорочено називається MMORPG - тобто онлайнова рольова гра, а друга - MMOFPS - тобто онлайнова стрілялка.

Перші ігри цього стилю з'явилися в 1970-х роках - тоді це були текстові гри. Пізніше, на

початку 1990-х, вони знайшли графічний інтерфейс і з тих пір розвиваються, займаючи уми все більшої кількості гравців. Серед популярних MMORPG можна відзначити Ultima Online, EverQuest, Final Fantasy XI, Lineage, World of Warcraft, RuneScape, MapleStory EVE Online. Серед MMOFPS можна відзначити World War 2 Online і PlanetSide.

LINEAGE

Перша гра серії Lineage була випущена компанією NCsoft в 1998 році. Гра користувалася непоганою популярністю, проте даний світове визнання отримала лише Lineage II, яка побачила світ у 2003-му році. Це - одна з найбільш успішних комерційних MMORPG.

В ході гри вам належить створити персонажа, вибрати його професію і жити в ігровому світі - працювати, розважатися, битися з іншими персонажами (цей режим гри називається PvP - або Player versus Player), покращувати характеристики свого персонажа, заводити «домашніх тварин», брати участь в політичному та економічному житті віртуального світу.

SIMULATOR

Simulator (або, по-українськи - симулятор) - це гра, покликана якомога точніше (або хоча б правдоподібно) що-небудь імітувати. Ці ігри були популярні у всі часи - не можна сказати, що симулятори - найпопулярніший вид ігор, але найстаріші з них - наприклад - Microsoft Flight Simulator, перша версія якого вийшла в 1982 році, існують вже декілька десятків років. Популярність симуляторів криється в тому, що вони дозволяють «не виходячи з дому» відчути себе за штурвалом літака або вертольота, покерувати гоночної машиною або посмикати за важелі танка, стати директором транспортної компанії, політати в космосі, повоювати і так далі. Розвиток цих ігор полягає в підвищенні якості графічної складової гри, в поліпшенні імітації реальних подій.

Існують економічні симулятори (Railroad Tycoon), авіасимулятори (Microsoft Flight Simulator, Falcon), військові симулятори (наприклад - симулятор танка Abrams або Operation Flashpoint: Cold War Crisis, поліпшена версія якого іспользуюется для військових тренувань), космічні симулятори (Freelancer), симулятори поїздів (Microsoft Train Simulator), симулятори міст (SimCity) симулятори автомобільних гонок зазвичай виділяють в окремий жанр - до нього належать такі ігри, як популярна серія Need For Speed і Gran Turismo.

NEED FOR SPEED

Симулятор автомобільних гонок Need For Speed з'явився в 1994 році з надр компанії Electronic Arts. Якщо точніше класифікувати цю гру - то це швидше за аркадний симулятор - гра, де характеристики автомобілів перевищують їх реальні аналоги, в той час як класичні симулятори прагнуть до найбільш точного відтворення реальних характеристик автомобілів.

Ігровий процес нових версій Need For Speed (наприклад - Need For Speed: Underground) цікавий ще й тим, що ви можете в процесі гри покращувати свій автомобіль на гроші, виграні в гонках.

RAILROAD TYCOON

Серія Railroad Tycoon з'явилася в 1990 році зусиллями компанії MicroProse. Гра існує і до цього дня - так, нові версії гри виходили в 1993 (якщо бути точним, в 1993 вийшло оновлення гри - Railroad Tycoon Deluxe), 1998, 2003 і 2006 роках.

В ході гри ви вирішуєте дві основні задачі. Перша - це споруда доріг між об'єктами і перевезення різних товарів (наприклад, ліс служить джерелом колод для деревообробної фабрики, яка в свою чергу виробляє папір для друкарні). І друга - ви спостерігаєте за конкурентами, купуєте їх акції, вчасно продавши які можете серйозно поправити своє фінансове становище, а в сучасній версії гри Sid Meyer's Railroad ще й берете участь в аукціонах на ексклюзивні права використання нових технологій.

SPORT

Одна з перших відеоігор - Tennis for Two 1958 року - ставилася до жанру спортивних ігор.

Звичайні спортивні ігри були популярні задовго до відеоігор, тому цілком природно, що спорт популярний в комп'ютерному вигляді. Не можна сказати, що популярність цих ігор так само величезна, як у кращих FPS або RPG, але вони мають стійкі позиції на ігровому ринку. Спортивні ігри, в свою чергу, можна розбити на підгрупи, грунтуючись на видах спорту. Мабуть, кожна спортивна гра, яка знаходить відгук серед фанатів того чи іншого виду спорту, має право називатися культової.

З помітних спортивних відеоігор можна відзначити наступні: NBA 2K (баскетбол), All-Star Baseball (бейсбол), FIFA (футбол), NHL 2K (хокей), Kings of the Beach (волейбол), Virtual Pool (більярд), PBA Bowling (боулінг), Knockout Kings (бокс), World Darts (дартс), Sega Bass Fishing (рибна ловля), World Tour Golf (гольф), Moto Racer (мотогонки), F1 Challenge (симулятор гонок Formula 1), World Karate Championship (карате), Ski or Die (катання на ковзанах), Kelly Slater's Pro Surfer (серфінг), Virtua Tennis (теніс).

ADVENTURE

Клас Adventure включає в себе ігри пригодницької тематики. Перша пригодницька гра з'явилася в 1970 році - це була Colossal Cave Adventure.

Пік популярності цих ігор припав на початок 1990-х років, зокрема - це пов'язано з виходом гри Myst. Після цього популярність жанру падала - справа в тому, що ці ігри не створюють такого ж атмосфери залученості в ігровий процес (гра йде неспішно, ви змушені багато всього читати і слухати, управління, як правило, ведеться за допомогою миші), як інші - ті ж FPS. В результаті, з одного боку, ці ігри не зникли зовсім, а з іншого - їх популярність і поширеність непорівнянна з іншими видами ігор.

В ході гри гравець розгадує загадки, вирішує головоломки, спілкується з іншими персонажами. Гра чимось нагадує RPG, проте тут персонаж зазвичай не піддається розвитку - всю гру можна представити у вигляді такого собі заздалегідь визначеного плану дій, який ви проходите за допомогою вашого персонажа.

Серед помітних Adventure-ігор можна відзначити Day of the Tentacle, Myst, King's

Quest, Monkey Island, The Longest Journey, Shenmue, Grim Fandango, Trace Memory, Fahrenheit, Dreamfall

Почасти до жанру adventure відноситься і популярна сучасна гра Grand Theft Auto (до речі, заборонена в деяких країнах - її герой - злочинець, а ігрові місії, як правило, пов'язані з вбивствами і грабежами), однак вона включає в себе і величезну частку action-ігри і почасти RPG. Існують і інші ігри змішаних жанрів.

ІГРИ ЗМІШАНИХ ЖАНРІВ

Ігри змішаних жанрів поєднують в собі відразу кілька жанрів. Наприклад, це може бути FPS і RPG - як у випадку з грою Deus Ex (вона помітна тим, що це, з одного боку - захоплююча гра, а з іншого - вона отримала позитивні відгуки багатьох критиків), це може бути суміш екшену і RPG - наприклад Diablo - поза всяким сумнівом, культова гра для мільйонів геймерів.

DIABLO

Diablo вийшла з надр компанії Blizzard North в 1996 році. Це - досить похмура іграшка - ви керуєте гравцем, характеристики якого вам належить покращувати протягом всієї гри, подорожуєте по світу, де панують породження пекла, і методично знищуєте зустрінуте зло. Гра має вигляд «зліва-зверху» - ні про яке FPS і мови не йде, однак те, що дії відбуваються в реальному часі, дають право віднести цю гру до розряду екшенів, а постійне поліпшення героя - до рольових ігор.

Іноді цю гру відносять до класу - Hack and Slash - маючи на увазі той факт, що основне заняття героя в таких іграх - це знищення ворогів, причому, переважно за допомогою різних видів холодної зброї.

DEUS EX

У грі Deus Ex, яка була випущена компанією Ion Storm Inc. у 2000 році, ви керуєте персонажем, який в ході гри піддається розвитку (шляхом поліпшення і додавання різних навичок) і проходите різні місії в вигляді, стандартному для сучасних FPS.

Гра має розвинену сюжетну лінію, вам досить часто доводиться приймати самостійні рішення - не завжди ті, кого ви до цього вважали друзями, насправді такими є. Маючи досить сильну рольову складову, гра відрізняється гідною графікою (в основі лежить движок Unreal Engine), у неї є продовження. В Deus Ex переважає FPS, рольова складова так само цікава, але вона лише доповнює «бойову» частину гри, роблячи її осмисленої і цікавою.

Ми розглянули ключові ігрові жанри, тепер поговоримо про застосування ігор у навчанні.

ІГРИ ТА НАВЧАННЯ

Деякі ігри відрізняються досить складним ігровим процесом. Наприклад, гра Freespace 2 - космічний симулятор. Якщо людина без спеціальної підготовки відразу

візьметься за виконання однієї з ігрових місій - він змушений буде витратити досить багато часу для того, щоб освоїти ігрове управління і роботу з елементами інтерфейсу. За керування космічним кораблем і його підсистемами в цій грі відповідають, без перебільшення, майже всі клавіатурні клавіші. Тут і управління кораблем, і управління захисним полем корабля, настройка і застосування різних видів зброї, переговори з іншими кораблями, управління різними підсистемами корабля, наприклад - системою захоплення цілі.

Розробники гри не стали покладатися на те, що користувач самостійно навчиться керувати всією цією пишністю. Дійсно, досить складно за п'ять хвилин (зазвичай саме за такий проміжок часу людина розуміє, чи цікава йому гра) освоїти призначення кількох десятків кнопок і особливості ігрового світу. Розробники створили кілька так званих навчальних місій, в якому гравця вчать керувати кораблем і його підсистемами. Завдяки цим навчальним місіям гравець, до цього жодного разу не бачив Freespace 2, через півгодини досить впевнено керує своїм кораблем і виконує «реальні» завдання. Це - приклад явного навчання в грі. Причому, навчання ведеться навичкам, які потрібні для успішної взаємодії з ігровим світом. Інший варіант навчання в грі виглядає наступним чином.

Гравець починає гру, яка досить проста. Наприклад, в платформних іграх гравець переміщається по рівних майданчиків без ям і пасток, вороги йому зустрічаються нечасто. Але поступово складність гри наростає. Дуже важливим моментом в таких іграх є обмеженість кількості програшів (або, як кажуть, кількості життів гравця), які можна допустити в одній ігровій сесії. Таким чином гравець змушений знову і знову проходити одні й ті ж рівні, в результаті, до того моменту, як він зможе пройти всю гру, у нього формуються навички, які дозволяють йому успішно грати і вигравати.

Існують програми, які в ігровій формі дозволяють займатися, наприклад, освоєнням методу десятипальцевой друку по сліпому методу. Наприклад, відома програма СОЛО на клавітуре Володимира Шахиджаняна. Навчається за цією програмою, з одного боку, отримує хорошу психологічну підтримку, певний настрій, який допомагає займатися, а з іншого - проходить «рівні», все більше ускладнюються, які дозволяють йому, якщо він успішно закінчить навчання, навчитися швидко і якісно набирати на клавіатурі. Тут ми теж можемо бачити багаторазове повторення невдалих завдань - рису, характерну для багатьох ігор.

Існують ігри, які дозволяють формувати у які грають навички, які цінні не тільки в ігровому процесі. Наприклад, гра Роберта Кіосакі CASHFLOW має дуже сильну практичну спрямованість - вона вчить людей поводитися з грошима, роблячи це в невимушеній ігровій формі. У грі є така деталь - зробивши угоду, гравець повинен розподілити суми, пов'язані з цією угодою, в бухгалтерському балансі. Спочатку це, природно, виходить не дуже швидко і не дуже правильно - програма не дає продовжувати гру, пропонує по-новому розподілити суми. В результаті людина, який не пошкодував кілька днів на роботу з програмою, починає з першого погляду відрізняти актив від пасиву, прибутку від збитків. Схожі цілі переслідує відома гра Монополія -вона дозволяє проникнути в економічні таємниці, зрозуміти дію економічних законів.

Комп'ютерні ігри - незважаючи на те, що вони, як і будь-які ігри, орієнтовані на процес, а не на результат, є потужним засобом для навчання. Вище наведені приклади такого навчання. Тому можна говорити про те, що навчальні ігри - це одне з важливих напрямків освіти майбутнього. Якщо взяти вдалу, перевірену часом, ігрову ідею і забезпечити її смисловим навантаженням по якому-небудь предмету навчальної програми, ми можемо отримати відмінний інструмент для цікавого й ефективного навчання. На жаль, зараз гри вкрай слабо інтегровані в навчальний процес. Виною тому недостатня розвиненість індустрії розробки ігор, недостатня поширеність інструментів для швидкої розробки ігор і недостатній рівень навчання розробці ігрових продуктів. Якби викладачі мали інструменти, які дозволили б їм замість традиційних навчально-методичних комплексів створювати повноцінні комп'ютерні ігри, що реалізують ті ж функції, освіту зробило б великий крок вперед.

XNA Game Studio - це один з прикладів спроби максимально наблизити розробку ігор до звичайному користувачеві. Зараз, звичайно, не можна говорити про те, що розробка гри в XNA - це гранично проста справа, доступне кожному. Все-таки, для розробки гри в XNA потрібно володіти серйозними навичками програміста. Існують, звичайно, ігрові конструктори, що вимагають мінімальних знань програмування, що дозволяють розробляти гри в візуальному режимі навіть тим людям, які абсолютно не володіють навичками програмування. Але майбутнє, думається, саме за продуктами, що нагадують XNA.

ІГРИ ТА ФОРМУВАННЯ ПРОФЕСІЙНИХ КОМПЕТЕНЦІЙ

Освіта завжди так чи інакше намагається готувати професіоналів, які відповідають запитам роботодавців. Останнім часом роботодавці приділяють дедалі більшу увагу підбору персоналу відповідно до компетентнісного моделями. Компетенція - це виражена здатність застосовувати свої знання і вміння на практиці. Спеціально створені гри здатні служити як потужним допоміжним інструментом для формування різних знань і умінь, так і способом перевірки цих знань і умінь.

ЛЕКЦІЯ 4. ТРИВИМІРНА ГРАФІКА

Анотація: Ця лекція присвячена основам роботи з тривимірною графікою. Тут ми поговоримо про тривимірної системі координат, про складові частини тривимірних об'єктів, про матричних обчисленнях, які використовуються в тривимірній графіці. Так само ми розглянемо відповідні частини об'єктної моделі XNA.

Ключові слова: система координат, координати, об'єкт, вершина, vertices, мережа, вектор, довжина, простір, чергу, матриця, matrix, матриця проекцій, шаблон, коефіцієнти, проекція, площину, ambient light, point light, direct light, клас, net, статичні методи, операції, значення, pixel, model

- Завдання лекції
- Ознайомитися з правобічної системою координат.
- Ознайомитися з поняттям точки, вершини, вектора, полігону в тривимірному просторі
- Ознайомитися з застосуванням світової, видовий і проекційної матриць
- Ознайомитися з матричними перетвореннями в тривимірному просторі.
- Ознайомитися з концепціями освітлення об'єктів.
- Ознайомитися з найбільш важливими властивостями і методами об'єктів XNA, використовуваних при роботі з тривимірною графікою

СИСТЕМА КООРДИНАТ

При роботі з тривимірною графікою використовується кілька видів систем координат. Для відображення двовимірних об'єктів нам була потрібна відповідна система координат з двома осями - горизонтальною віссю X і вертикальною віссю Y. Нагадаємо, що екранна система координат для двовимірної графіки має початок (точку 0,0) в лівому верхньому кутку монітора, позитивна частина осі X розташовується праворуч від початку координат, позитивна частина осі Y - знизу.

Для роботи з тривимірними об'єктами нам знадобиться ще одна вісь - вона називається вісь Z. Існує кілька варіантів тривимірних систем координат, зокрема, поширені так звані правобічна і лівостороння системи. Ми будемо користуватися правобічної системою - вона застосовується в XNA Framework. Її схематичне зображення наведено на рис. 4.1.

Особливість цієї системи координат полягає в тому, що початок координат можна зіставити з лівим нижнім кутом монітора, позитивна частина осі X знаходиться праворуч від початку координат, позитивна частина осі Y - зверху, а позитивна частина осі Z - спереду. А це означає, що видима частина осі Z - це її негативна частина. Ця частина осі перебуває ніби «в глибині монітора», в той час, як позитивна частина знаходиться «попереду монітора». На рис. 4.1. пунктиром зображена негативна частина осі Z.

У двовимірної системі координат існує поняття точки - її координати задаються двома значеннями - X і Y. Точки існують і в тривимірній системі координат - вони задаються вже трьома значеннями - X, Y, Z.

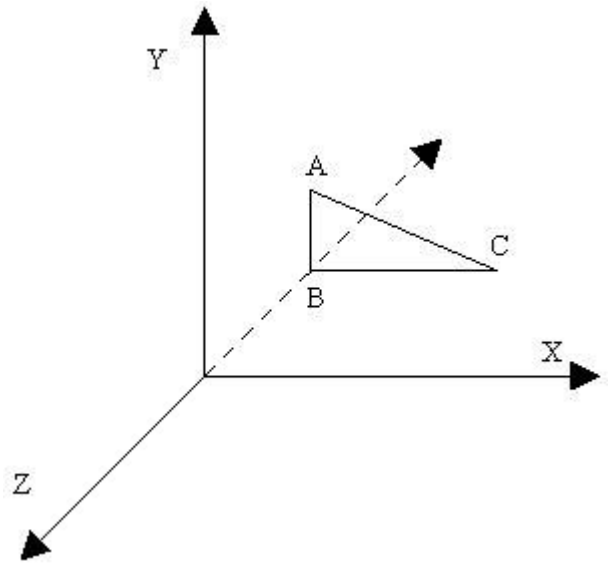

Рис. 4.1. Правостороня система координат

Точки використовують для того, щоб ставити координати вершин багатокутників (полігонів), зокрема - трикутників. Так, трикутник, зображений на рис. 18.1., Заданий трьома крапками - A, B, C.

Як правило, більш складні тривимірні об'єкти будуються саме з трикутників.

У тривимірній графіці існує таке поняття, як грань (face). Це - плоский об'єкт, який визначають кілька вершин. У нашому випадку звичайний трикутник - це саме грань. З кількох плоских граней можна зібрати об'ємний об'єкт.

Чим більше трикутників використано при побудові моделі - тим більше деталізованої вона виходить. Точки, що відповідають вершинам трикутника, який можна зобразити у тривимірному просторі, називаються вершинами. Працюючи з тривимірною графікою в XNA вам часто доводиться зустрічати англійський варіант слова вершина - vertex. Можливо, вам зустрінеться множину слова вершина: «вершини» виглядає по-англійськи як «vertices». Іноді для позначення вершин використовують кальку з англійської - вертекс.

Трикутник не випадково була обрана в якості базової геометричної фігури - по-перше - цей багатокутник завжди є опуклим, по-друге - неможливо розташувати три точки таким чином, щоб вони не належали одній площині. Тобто, трикутник - це фігура, яка завжди є опуклою і плоскою, що дозволяє з успіхом використовувати його в цілях тривимірної графіки.

Кілька граней, з яких складається тривимірний об'єкт, називаються мережею (mesh). Мережа являє собою набір трикутників.

Ще одне поняття, яке стане в нагоді вам при роботі з тривимірною графікою - це поняття вектора. Вектор (vector), так само як і точка, може бути визначений трьома параметрами, однак він описує не положення в просторі, а напрямок і швидкість руху. Вектор має початок і кінець, для його повного визначення потрібно знати координати точки початку і кінця вектора, тобто, замість трьох значень координат нам знадобиться вже шість значень. Однак, якщо за замовчуванням прийняти за початок вектора

початок координат (точку 0,0,0) - тоді для його визначення вистачить і трьох точок.

Наприклад, вектор з координатами (1,0,0) означає: «напрямок - вправо, швидкість - 1». Якщо відкласти цей вектор від початку координат, то добре видно, що він спрямований саме вправо (рис. 4.2.).

Напрямок вектора визначається положенням другої точки щодо першої (в нашому випадку - положення точки кінця вектора, якої задається вектор щодо початку координат), а швидкість - довжиною вектора - тобто - різницею між початковою і кінцевою точкою. У нашому випадку довжина вектора збігається з координатами його кінця.

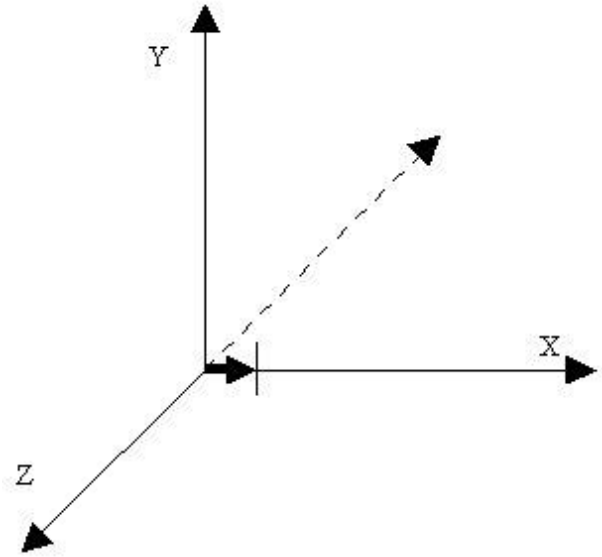

Рис. 4.2. Вектор (1,0,0)

Існує особливий вид векторів - нормалі (normals). Нормалі можуть бути побудовані для граней і для вершин об'єкта. Нормалі для граней перпендикулярні цим гранях. Вони використовуються при розрахунку кольору об'єкта.

Перетворення в тривимірному просторі

Знаючи координати вершин полігонів, з яких складається об'єкт, ми можемо розташувати його в просторі. Тепер потрібно розібратися зі зміною положення об'єктів в просторі. Існує кілька основних операцій, які можуть використовуватися для переміщення об'єктів в тривимірному просторі. Це - переміщення (translation), обертання (rotation) і масштабування (scale).

Результати роботи графічної підсистеми тривимірної гри ми бачимо на плоскому екрані монітора - змодельована комп'ютером тривимірна сцена проектується на двовимірну поверхню. При проектуванні потрібно вибрати точку, яка виконує роль камери, що дозволяє бачити тривимірний простір. У свою чергу, об'єкти в тривимірному просторі можуть переміщатися відповідно до певних правил. Для управління всім цим використовуються кілька матриць. Це - світова матриця (World Matrix), матриця виду (View Matrix) і матриця проекції (Projection Matrix).

Матрицю можна представити у вигляді таблиці, що складається з m рядків і n стовпців.

У комп'ютерній графіці застосовуються матриці 4x4. Перших три стовпці цієї матриці відповідають за модифікацію координат X, Y, Z вершин об'єкта, який бере участь в трансформації.

Світова матриця дозволяє задавати перетворення - переміщення, обертання і трансформації об'єктів.

Матриця виду дозволяє управляти камерою.

Матриця проекції служить для настройки проекції тривимірної сцени на екран.

Припустимо, є трикутник, заданий наступними вершинами (табл. 4.1.)

Таблиця 4.1. Вершини трикутника до переміщення

Вершина	**X**	**Y**	**Z**
1	20	10	5
2	15	20	10
3	25	30	10

При переміщенні цього трикутника на 10 позицій по осі X ми повинні додати по 10 до кожної з координат X його вершин. В результаті вийде матриця такого виду (табл. 4.2.).

Таблица 4.2. Вершини трикутника після переміщення

Вершина	**X**	**Y**	**Z**
1	30	10	5
2	25	20	10
3	35	30	10

Того ж ефекту можна досягти, помноживши координати кожної з вершин на світову матрицю. Для цього координати вершини представляють у вигляді матриці, що складається з одного рядка і чотирьох стовпців. У перших трьох стовпці містять координати X, Y, Z, в четвертому - 1. Світова матриця представлена у вигляді таблиці 4x4. Ось як виглядає операція множення матриць (формула 4.1.):

$$\begin{bmatrix}20 & 10 & 5 & 1\end{bmatrix} * \begin{bmatrix}1 & 0 & 0 & 0\\ 0 & 1 & 0 & 0\\ 0 & 0 & 1 & 0\\ 10 & 0 & 0 & 1\end{bmatrix} = \begin{bmatrix}30 & 10 & 5 & 1\end{bmatrix} \quad (4.1)$$

При перетворенні кожна з вершин множиться на світову матрицю.

Кожне з перетворень в просторі вимагає особливої настройки світової матриці. У формулі 4.2. наведено шаблон світової матриці, яка дозволяє переміщати об'єкти в просторі.

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & 1 & 0 & 0 \\ 0 & 0 & 1 & 0 \\ & & & 1 \end{bmatrix} \quad (4.2$$

Тут X, Y и Z - це прирощення координат X, Y и Z.

Мирова матриця для обертання об'єктів навколо осі X виглядає так (формула 4.3.).

$$\begin{bmatrix} 1 & 0 & 0 & 0 \\ 0 & \cos(\alpha) & \sin(\alpha) & 0 \\ 0 & -\sin(\alpha) & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.3)$$

Тут - кут поворота в радіанах

Мирова матриця для обертання об'є ктів по осі Y виглядає так (формула 4.4.)

$$\begin{bmatrix} \cos(\alpha) & 0 & -\sin(\alpha) & 0 \\ 0 & 1 & 0 & 0 \\ \sin(\alpha) & 0 & \cos(\alpha) & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.4)$$

Матриця для обертання об'єктів навколо осі Z наведена в формулі 4.5.

$$\begin{bmatrix} \cos(\alpha) & \sin(\alpha) & 0 & 0 \\ -\sin(\alpha) & \cos(\alpha) & 0 & 0 \\ 0 & 0 & 1 & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.5)$$

Формула 4.6. представляє матрицю, яка служить для трансформації об'єктів.

$$\begin{bmatrix} \phi X & 0 & 0 & 0 \\ 0 & \phi X & 0 & 0 \\ 0 & 0 & \phi Z & 0 \\ 0 & 0 & 0 & 1 \end{bmatrix} \quad (4.6)$$

Δx, Δy, Δz - це коефіцієнти масштабування, які застосовуються до вершин. Вони дозволяють «стискати» або «розтягувати» об'єкти.

Матриця виду впливає на стан камери - точки, з якої здійснюється перегляд тривимірної сцени.

Матриця проекції дозволяє управляти проектуванням сцени на екран. У XNA існує два види проекцій.

Перший - це перспективна проекція (Perspective projection). У цій проекції об'єкти

виглядають так само, як ми звикли їх бачити в реальному світі. Об'єкти, які розташовані далі, здаються менше об'єктів, розташованих ближче.

Другий вид проекції - це ортогональна проекція. Тут об'єкти проектуються на площину екрану без урахування перспективи.

ОСВІТЛЕННЯ

Освітлення об'єктів в іграх виконує ту ж роль, яка відведена йому в реальному світі. Існує безліч типів освітлення. Навколишній розсіяне світло (ambient light) - це світло, яке освітлює всі об'єкти сцени з однаковою інтенсивністю. Джерело розсіяного світла не має місця розташування.

Точкове джерело світла (point light) - це джерело, яке випромінює світло в усіх напрямках. Його можна порівняти зі світлом, що виходить від лампочки, не прикритої абажуром.

Направлене джерело світла (directional light). Це джерело, на відміну від точкового, не має місця розташування, проте має орієнтацію

Зональний джерело світла (spot light) або прожектор має місце розташування, орієнтацію, а його світловий потік обмежений у формі конуса.

Джерела світла можуть мати різну інтенсивність, різний колір, при висвітленні сцен можна використовувати кілька різних джерел. Все це робить освітлення найважливішим елементом тривимірної графіки.

ШЕЙДЕРЫ

Шейдери, або шейдерниє програми - це програми, які дозволяють застосовувати до моделей різні ефекти. Вони пишуться на спеціальній мові програмування, як правило, не вручну, а з використанням відповідного програмного забезпечення. Шейдери діляться на верхові і піксельні. Вершинні шейдери дозволяють застосовувати різні ефекти до вершин моделей, піксельні шейдери обробляють колір кожного з пікселів моделі перед виведенням її на екран.

ТЕКСТУРИ

Текстури - це растрові (двовимірні) зображення, які накладаються на тривимірні моделі. Наприклад, тривимірна модель автомобіля може являти собою автомобіль, який як би «вирізаний» з твердого матеріалу, а після накладення на цю модель відповідної текстури автомобіль набуває кольору, створюється ілюзія наявності у нього дрібних деталей оформлення і т.д. Мінімальна одиниця текстури називається тексель. Чим більше пікселів доводиться на один тексель, чим більше дозвіл текстури - тим більш якісною буде виглядати модель після накладення на неї текстури.

ОБ'ЄКТИ XNA ДЛЯ РОБОТИ З 3D-ГРАФІКОЮ

В об'єктній моделі XNA існує кілька об'єктів, які інтенсивно використовуються для роботи з тривимірною графікою.

Клас Matrix дозволяє створювати матриці, які використовуються для перетворень

об'єктів в просторі, для управління камерою і проекцією тривимірної сцени на екран. Цей клас, який, як це зазвичай буває в NET Framework, є так само типом даних і містить статичні методи, використовувані для створення матриць. Як правило, доводиться, наприклад, «вручну» модифікувати світову матрицю для того, щоб організувати поворот або переміщення моделі - всі необхідні операції можна виконати за допомогою статичних методів класу Matrix, так само об'єкти класу Matrix мають безліч корисних властивостей. Розглянемо деякі з них.

Властивості Mxy повертають значення, які розташовуються в рядку з індексом x і в стовпці з індексом y, які змінюються від 1 до 4. Наприклад, властивість M11 повертає значення, що знаходиться на перетині першого рядка та першого стовпчика матриці, M23 - другого рядка і третього стовпця , M44 - четвертого рядка і четвертого стовпця.

Метод CreateLookAt дозволяє створити видову матрицю (камеру).

Метод CreateOrthographic дозволяє створити ортогональную проекційну матрицю.

Метод CreatePerspective дозволяє створити перспективну проекційну матрицю.

Методи CreateRotationX, CreateRotationY, CreateRotationZ дозволяють створювати матриці повороту навколо відповідної осі.

Метод CreateScale призначений для створення матриці масштабування (зміни розмірів) об'єкта.

Метод CreateTranslation дозволяє створити матрицю переміщення об'єкта.

Метод CreateWorld створює світову матрицю.

Клас Vector3 використовується для зберігання координат точок в просторі, для завдання векторів і для деяких інших завдань, наприклад - для завдання параметрів матриць.

Клас BasicEffect містить безліч елементів, важливих при створенні тривимірної сцени. Саме об'єкт цього класу містить опису матриць, опису джерел світла в грі і багато інших важливих властивості і методи.

Властивість AmbientLightColor дозволяє встановити навколишнє освітлення розсіяним світлом

Властивості DirectionalLight0, DirectionalLight1, DirectionalLight2 дозволяє встановити джерела спрямованого кольору

Властивості FogColor, FogEnabled, FogEnd, FogStart дозволяють управляти ефектом туману

Властивість LightingEnabled дозволяє вмикати освітлення

Властивість PreferPerPixelLighting дозволяють управляти ефектом попиксельного освітлення - цей тип освітлення доступний для графічних адаптерів, що підтримують Pixel Shader Model 2.0 (Модель піксельних шейдеров 2.0.)

Властивість Projection дозволяє управляти проекційної матрицею.

Властивість Texture призначене для управління текстурою

Властивість TextureEnabled дозволяє включати використання текстури

Властивість View дозволяє управляти матрицею виду (камерою)

Властивість World призначений для управління світовою матрицею

Методи Begin і End дозволяють вказати початок і кінець блоку обробки ефекту.

Метод EnableDefaultLighting дозволяє увімкнути освітлення за замовчуванням.

MATHHELPER

Містить корисні методи для проведення різних математичних операцій. Наприклад, статичний метод ToRadians позвлояет переводити значення з градусів в радіани.

При роботі з тривимірною графікою використовуються і інші об'єкти XNA, з ними ви познайомитеся в лекціях.

Лабораторні роботи

ЛАБОРАТОРНА РОБОТА №1. ВВЕДЕННЯ ДО XNA GAME STUDIO 2.0.

Анотація: У цій лабораторній роботі ми розглянемо середу розробки, в якій нам належить працювати, а так само вивчимо стандартний ігровий проект, на основі якого створюються комп'ютерні ігри.

Ключові слова: середовище розробки, Visual Studio, пункт, поле, досвід, Solution Explorer, error, list, простір, діяльність, ПО, файл, Windows, С, виконання, команда, папка, content, контент, конструктор класу, об'єкт, Graphics , конструктор, клас, циклу, параметр, знання, реальний час, простір імен, audio, утиліта, input, миша, маніпулятор, storage, операції, net, live, pipeline, змінна, 3D, розділи

Завдання роботи:

- Створити стандартний ігровий проект
- Вивчити його пристрій
- Вивчити вміст і особливості файлу Program.cs
- Ознайомитися зі спрощеною схемою роботи гри
- Вивчити призначення методів стандартного ігрового проекту і особливості підключаються до нього просторів імен (файл Game1.cs)

СТВОРЮЄМО ІГРОВИЙ ПРОЕКТ

Перед виконанням цієї лабораторної роботи переконайтеся, що на ПК встановлена середовище розробки Visual Studio і XNA 2.0.

Для запуску середовища розробки, виконайте команду Пуск Все програми Microsoft Visual C # 2005 Express Edition. Подальші приклади будуть показані саме з використанням Express-версії. Виконайте команду File New Project - відкриється вікно New Project, в якому нам пропонують вибрати один із стандартних типів проектів (рис. 1.1.)

У лівій частині вікна виберемо пункт XNA Game Studio 2.0., В правій - Windows Game (2.0.). Ім'я проекту, яке сгенеровано автоматично (поле Name) замінимо на ім'я P1_1.

Після натискання на кнопку OK шаблонний проект буде створений. Ось як виглядає вікно середовища розробки після його створення (рис. 1.2.).

Якщо ви виконуєте цю лабораторну роботу, то ви, швидше за все, маєте досвід роботи з Visual Studio. Можна помітити, що зовнішній вигляд середовища розробки виглядає цілком традиційно. Праву частину вікна займають панелі Solution Explorer і Properties, в нижній частині вікна можна бачити вікно Error List, основний простір зайнято вікном редактора коду.

Нагадаємо, що Solution Explorer використовується для перегляду і модифікації файлів, які входять в проект, для додавання нових файлів в проект, вікно Properties застосовується для модифікації властивостей виділеного об'єкта, вікно Error List містить повідомлення про помилки, які генеруються при виконанні програми, а вікно редактора коду застосовується при редагуванні текстів програм. Основна діяльність по створенню гри ведеться саме з використанням вищезазначених інструментів.

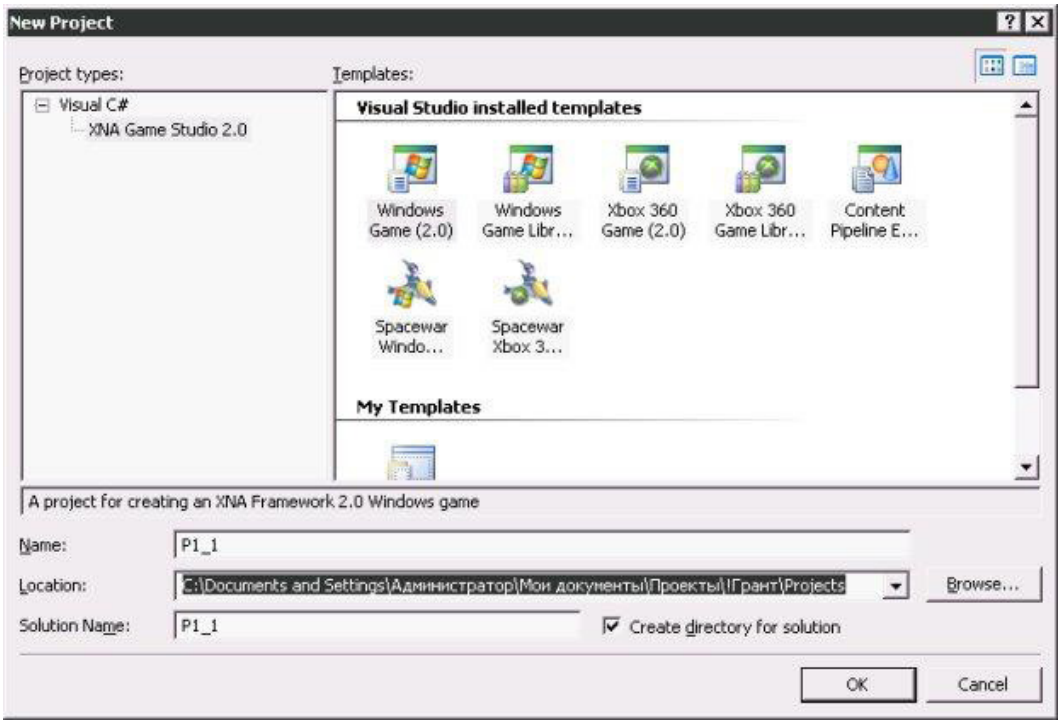

Рис. 1.1. Вибір типу проекту, що створюється

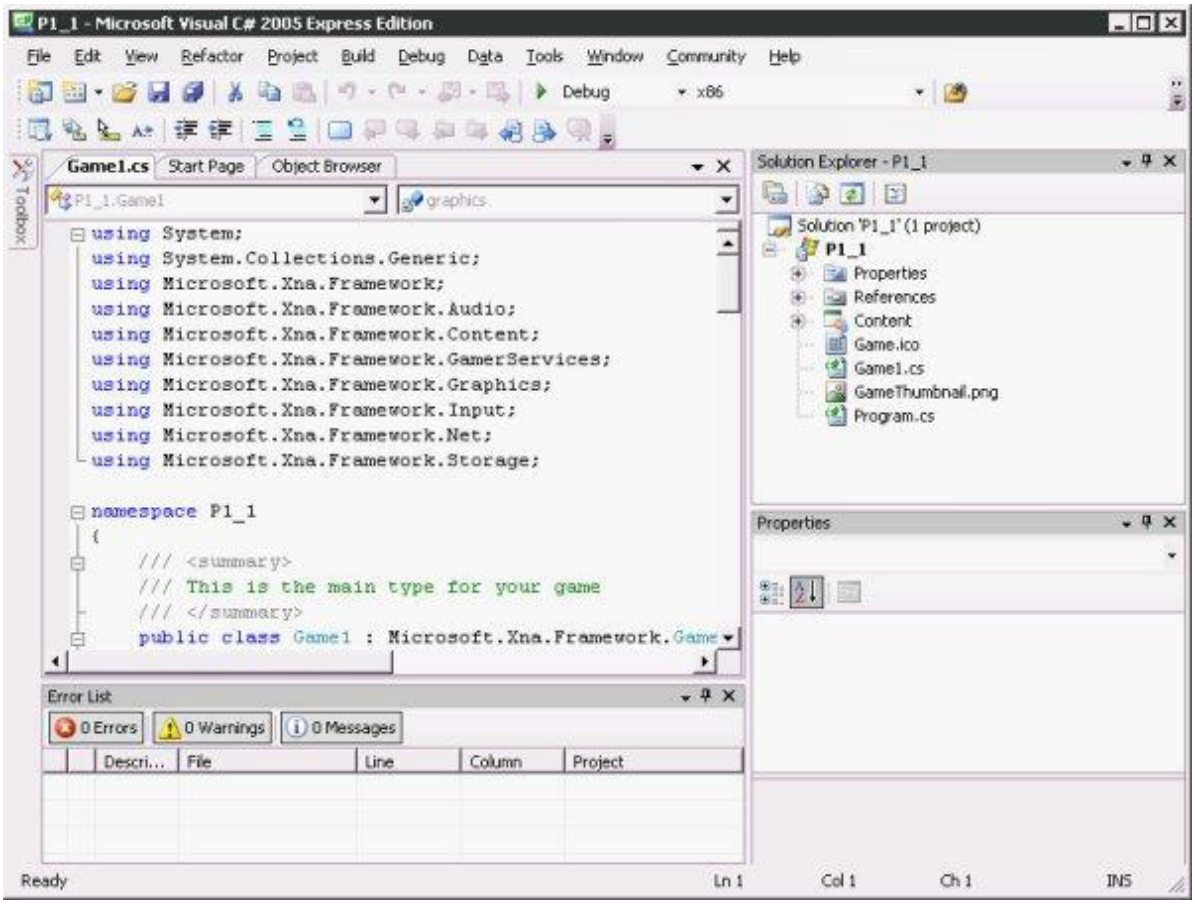

Рис. 1.2. Середовище розробки після створення шаблонного проекту

Для тестового запуску гри служать команди Debug Start Debugging і Debug Start Without Debugging.

Якщо зараз ми спробуємо запустити гру, ми побачимо лише порожнє вікно (рис. 1.3.).

Рис. 1.3. Вікно шаблонного проекту

Це вікно готово для ваших ігрових експериментів.

Розглянемо найважливіші елементи вмісту панелі Solution Explorer (рис 1.4).

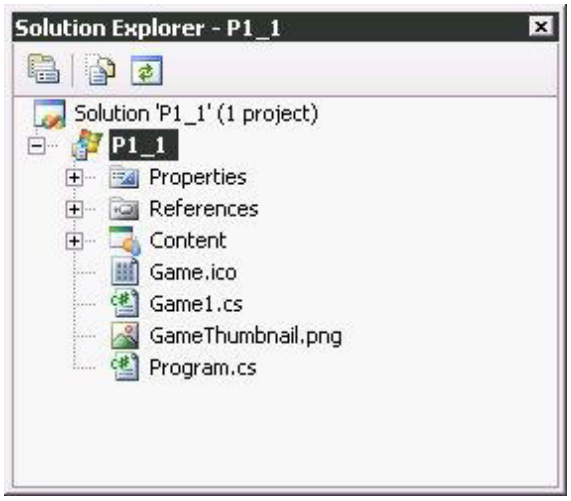

Рис. 1.4. Панель Solution Explorer

Файл Program.cs - це звичайний файл програми на C #. Точно такі ж файли зазвичай генеруються при створенні консольних Windows-додатків на C #. Даний файл грає важливу роль - в ньому міститься точка входу в програму, з нього починається виконання гри, і саме в ньому міститься команда, яка запускає гру.

Файл Game1.cs - це файл, в якому зберігається програмний код гри. Як правило, вищеописаний Program.cs в модифікації не потребує, а ось Game1.cs - це файл, з яким доводиться працювати в процесі створення гри. Причому, ігри зазвичай включають в себе безліч файлів з програмним кодом.

Папка Content містить ігровий контент - тобто - файли різних форматів, які використовуються в грі.

Розглянемо код стандартного проекту. Нам доведеться модифікувати цей код в процесі створення власних ігор.

РОЗБІР КОДУ СТАНДАРТНОГО ІГРОВОГО ПРОЕКТУ

В лістингу 1.1 наведено вміст файлу Program.cs.

```
using System;
namespace P1_1
{
   static class Program
   {
      /// <summary>
      /// The main entry point for the application.
      /// </summary>
      static void Main(string[] args)
      {
         using (Game1 game = new Game1())
         {
            game.Run();
         }
      }
   }
}
```

Лістинг 1.1. Код файлу Program.cs

Як бачите, виглядає він цілком стандартно. У тілі статичного методу Main містяться команди створення нового об'єкта класу Game1. Об'єкту присвоюється ім'я game, після чого виконується метод Run об'єкта Game, який і запускає гру.

Спрощено структуру ігрового проекту можна представити так (Табл. 1.1.).

Таблиця 1.1. Структура ігрового проекту

Назва	Опис
Ініціализація	На етапі ініціалізації проекту проводиться ініціалізація графічної і звукових підсистем, системи введення даних.
Загрузка ресурсів	Тут завантажуються ігрові ресурси, такі, як текстури і тривимірні моделі, звуки, шрифти.
Початок ігрового циклу	Основні дії, які виконуються при роботі гри, організовані у вигляді ігрового циклу. Фактично, ігровий цикл - це обов'язковий елемент будь-якої гри.
Зчитування даних користувацького введення	Користувач управляє грою за допомогою різних пристроїв введення даних. У нашому випадку це - клавіатура і миша. Тут так само може оброблятися інформація, яка надходить від інших пристроїв введення. Наприклад - від джойстика.
Ігровые обчислення	Ігрові обчислення включають в себе всі обчислення, які необхідні для організації ігрового процесу. Зокрема, тут обраховуються позиції ігрових об'єктів, проводяться обчислення, пов'язані з роботою штучного інтелекту, обчислення, пов'язані з перевіркою зіткнень ігрових об'єктів, ігрова фізика, підрахунок очок, набраних гравцем і так далі. Якщо мова йде про оптимізацію швидкодії гри, то така оптимізація зазвичай стосується саме ігрових обчислень, так як на них витрачається велика частина часу виконання ігрового циклу.
Перевірка критерію припинення гри	Ігровий цикл триває до тих пір, поки гра не буде припинена. Наприклад, критерієм зупинки може бути витікання часу, виділеного гравцеві на виконання ігрової задачі, набір певної кількості очок, якась подія, що сталася в процесі гри, примусова зупинка гри, якщо користувач вирішив припинити грати і так далі.

Назва	Опис
Вивід зображень, програвання звуків	Після завершення етапу ігрових обчислень в програмі є дані для візуалізації. Зокрема, є дані про нові позиції об'єктів, про повідомлення, які потрібно вивести на ігрове поле, про звуки, які потрібно відтворити. На даному етапі проводиться переміщення об'єктів по ігровому полю, виведення повідомлень, програвання звуку.
Кінец ігрового циклу	Ігровий цикл закінчується після виконання умови зупинки гри.
Звільнення ресурсів	На останньому етапі роботи гри проводиться звільнення системних ресурсів, зайнятих грою і вихід з гри.

У стандартному ігровому проекті всі ці етапи представлені за допомогою спеціальних методів. Ці методи успадковані створеним ігровим проектом від класу Game, який служить основою для проекту. Отже, це такі методи (табл. 1.2.).

Таблиця 1.2. Методи стандартного ігрового проекту

Назва	Опис
Game1()	Загальна ініціалізація. Game1 () - це конструктор класу Game1, який виконується при створенні об'єкта Game в процедурі Main файлу Program.cs
Initialize()	Ініціалізация гри
LoadContent()	Завантаження ігрових ресурсів
Run()	Запуск ігрового циклу. Як було показано вище, метод Run () викликається в процедурі Main файлу Program.cs
Update()	Ігровий цикл в стандартному проекті складається з двох методів. Перший з них - це метод Update (), в якому виконується прийом призначеного для користувача введення, всі необхідні обчислення, перевірка критеріїв зупинки гри.
Draw()	Другий метод ігрового циклу - це метод Draw (), який містить код для візуалізації ігрової графіки.
UnloadContent()	Після закінчення ігрового циклу цей метод звільняє системні ресурси.

Тепер, коли ми ознайомилися з типовою структурою гри і призначенням стандартних методів, розглянемо код класу Game1 (Лістинг 1.2.). У стандартному ігровому проекті можна знайти англомовні коментарі. Тут, для більшої наочності коду, вони опущені. Коментарі, ключові для розуміння роботи окремих методів, наведені українською мовою.

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace P1_1
{
   public class Game1 : Microsoft.Xna.Framework.Game
   {
      GraphicsDeviceManager graphics;
      SpriteBatch spriteBatch;
      public Game1()
      {
         graphics = new GraphicsDeviceManager(this);
         Content.RootDirectory = «Content»;
      }

      protected override void Initialize()
      {
         // Сюди слід додати код ініціалізації
         base.Initialize();
      }

      protected override void LoadContent()
      {
          spriteBatch = new SpriteBatch(GraphicsDevice);
         // Сюда надо добавить код загрузки игровых ресурсов
      }
```

```
        protected override void UnloadContent()
        {
            // Код вивантаження ігрових ресурсів
        }

        protected override void Update(GameTime gameTime)
        {
            // Ця послідовність команд обробляє призначений для користувача введення і
дозволяє завершувати гру, вона розрахована на використання джойстика
            if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
                this.Exit();
            // Сюди слід додати код ігрових обчислень
            base.Update(gameTime);
        }

        protected override void Draw(GameTime gameTime)
        {
            graphics.GraphicsDevice.Clear(Color.CornflowerBlue);
            // Сюди слід додати код візуалізації ігрових об'єктів
            base.Draw(gameTime);
        }
    }
}
```

Лістинг 1.2. Код файла Game1.cs

Розглянемо кожен з наведених вище методів. У лістингу 1.3. наведено код методу Game1 ().

```
public Game1()
        {
            graphics = new GraphicsDeviceManager(this);
            Content.RootDirectory = «Content»;
        }
```

Лістинг 1.3. Код методу Game1()

Метод Game1 () - це конструктор класу Game1. Команда graphics = new GraphicsDeviceManager (this); створює нове ігрове вікно, а об'єкт graphics використовується для проведення різних графічних операцій. Наприклад, за допомогою цього об'єкта можна налаштовувати ширину і висоту вікна, переводити гру

в повноекранний режим.

Крім описаних команд, автоматично включаються в стандартний ігровий проект, конструктор може містити й інші команди. Наприклад, команди додаткової настройки об'єкта graphics.

В лістингу 1.4 наведений код методу Initialize().

```
protected override void Initialize()
    {
        // Сюди слід додати код ініціалізації
        base.Initialize();
    }
```

Лістинг 1.4. Код методу initialize()

Команда base.Initialize(); вызывает метод Initialize() базового класса. Сюда же добавляют команды для инициализации звуковой подсистемы игры, здесь же выполняются игровые настройки. Надо отметить, что в этом методе инициализируется все, кроме графических ресурсов игры. Для них существует метод LoadContent(), листинг 1.5.

```
protected override void LoadContent()
    {
        spriteBatch = new SpriteBatch(GraphicsDevice);
        // Сюди треба додати код завантаження ігрових ресурсів
    }
```

Лістинг 1.5. Код методу LoadContent()

SpriteBatch - це клас, який позвляет виводити зображення на пристрій, що задається параметром GraphicsDevice. В цей же метод додають команди для завантаження графічних ресурсів, шрифтів. Метод UnloadContent () (лістинг. 1.6.) Призначений для звільнення системних ресурсів.

```
protected override void UnloadContent()
    {
        // Код вивантаження ігрових ресурсів
    }
```

Лістинг 5.6. Код методу UnloadContent()

Тут розміщують команди для вивантаження ігрових ресурсів. Зокрема, в даному методі можна використовувати команду base.UnloadContent (); - це метод базового класу, який виносить ресурси. Так само в цей метод додають команди для вивантаження шрифтів і інших ресурсів.

Ігровий цикл складається з двох методів, що виконуються циклічно це методи Update () і Draw () - лістинг 1.7.

```
protected override void Update(GameTime gameTime)
```

```
{
   // Сюди слід додати код ігрових обчислень
   base.Update(gameTime);
}

protected override void Draw(GameTime gameTime)
{
 graphics.GraphicsDevice.Clear(Color.CornflowerBlue);
   // Сюди слід додати код візуалізації ігрових об'єктів
   base.Draw(gameTime);
}
```

Лістинг 1.7. Код методів Update() и Draw()

Спочатку виконується метод Update (), потім - метод Draw ().

Команда base.Update (gameTime); методу Update () відповідальна за організацію ігрового циклу. Вона знову і знову викликає метод Update () базового класу. Саме ця команда є «двигуном» гри. Якщо її прибрати - ігровий цикл не буде працювати. Параметр gameTime - являє собою об'єкт класу GameTime, який містить інформацію про ігровий часу. Знання ігрового часу потрібно для організації взаємодії всередині гри. По відношенню до ігрового часу можна виділити два типи ігор. Перші - це гри реального часу, такі, як симулятори. Другі - це покрокові гри, де гравцеві дається деякий час - фіксоване або немає - для обмірковування наступного ходу. Знання ігрового часу потрібно для багатьох ігрових обчислень. Клас GameTime має кілька важливих властивостей.

Властивість ElapsedGameTime дозволяє дізнатися час, що минув після останнього виклику ігрового циклу. TotalGameTime - ігровий час, що минув з моменту старту гри.

ElapsedRealTime - реальний час, що минув після останнього виклику ігрового циклу. TotalRealTime - реальний час, що минув з моменту старту гри.

Команда graphics.GraphicsDevice.Clear (Color.CornflowerBlue); методу Draw () очищає ігровий екран, заливаючи його кольором CornflowerBlue. А команда base. Draw (gameTime); викликає метод Draw () базового класу, який виводить на екран зображення.

Поговоримо про просторах імен (компонентах), які підключені до стандартного проекту за замовчуванням. У лістингу 1.8. наведено їх виклики.

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
```

```
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;
```

Лістинг 1.8. Компоненти, які під'єднуються

Простір імен Audio містить інструменти для роботи зі звуком. Для роботи зі звуком в XNA існує утиліта XACT (Microsoft Cross-Platform Audio Creation Tool). Підготовка музики і звуків здійснюється за допомогою цієї утиліти, створюється спеціальний XACT-проект, який завантажується в гру. Застосування XACT дозволяє використовувати один і той же звукове оформлення як в Windows-іграх, так і в Xbox360-іграх.

Простір імен Graphics є графічну частину XNA і містить в собі засоби для роботи з графікою.

Простір імен Input підтримує роботу із засобами введення даних, зокрема, це клавіатура, миша і ігровий маніпулятор - джойстик. Як правило, при створенні Windows-ігор використовують миша і клавіатуру, при створенні Xbox-ігор - джойстик.

Простір імен Storage містить засоби для роботи з файловою системою. Саме інструменти з Storage дозволяють, наприклад, зберігати ігрові дані в файли, читати їх і виконувати інші файлові операції.

Простір імен Net підтримує створення мережевих ігор за допомогою XNA. Реалізація мережевого взаємодії в XNA-іграх здійснюється за допомогою Windows LIVE.

Простір імен Content містить засоби, що забезпечують роботу Content Pipeline. Content Pipeline - це дуже важливий механізм XNA Framework.

Простір імен GamerService забезпечує функції роботи з сервісом LIVE - для ігор під Xbox і Windows. Ці функції необхідні для створення мережевих ігор. Зокрема, в GamerService знаходяться функції, що дозволяють гравцеві реєструватися для участі в грі, працювати з інформацією про гравця, зберігати локальні призначені для користувача настройки для мережевих ігор.

У розділі оголошення властивостей класу Game1 є дві змінних (лістинг 1.9.).

```
GraphicsDeviceManager graphics;
    SpriteBatch spriteBatch;
```

Лістинг 1.9. Змінні в розділі властивостей класу

Мінлива graphics має тип GraphicsDeviceManager - вона призначена для управління графічним пристроєм, який використовується для виведення інформації.

Мінлива spriteBatch має тип SpriteBatch - вона потрібна для виведення спрайтів на екран.

ЛАБОРАТОРНА РОБОТА №2. 2D-ГРАФІКА В XNA GAME STUDIO 2.0.

Анотація: У цій лабораторній роботі ми розглянемо систему координат, яка використовується при створенні двовимірних ігор, поговоримо про виведення двовимірних зображень в XNA, розглянемо питання накладення зображень, налаштування розмірів ігрового вікна і роботи у віконному і повноекранному режимах.

Ключові слова: система координат, об'єкт, координати, Windows, paint, прямокутник, PNG, Solution Explorer, content, меню, пункт, файл, параметр, змінна, тип даних, текстура, чергу виведення, клас, список, шаблон, Add, конструктор класу, простір, component, компонент, успадкування, draw, enter, висновок, площа, info, миша, знання, спрайт, висота, Width, Height, команда, посилання, перевантажений метод, гра, виклик методу, background.

Задачі роботи:

1) Ознайомитися з двовимірної системою координат
2) Створити двовимірне зображення в графічному редакторі
3) Навчитися завантажувати зображення в ігровий проект
4) Навчитися виводити зображення в ігрове вікно
5) Розробити стандартний клас, який використовується для конструювання ігрових об'єктів і зберігання інформації про текстуру і позиції зображення
6) Розробити клас, успадкований від DrawableGameComponent, навчитися виводити зображення, асоційоване з ним

СИСТЕМА КООРДИНАТ

У розробці двовимірних ігор використовується двовимірна система координат. На рис. 2.1. ви можете бачити звичайну систему координат і об'єкт, зображений в цій системі.

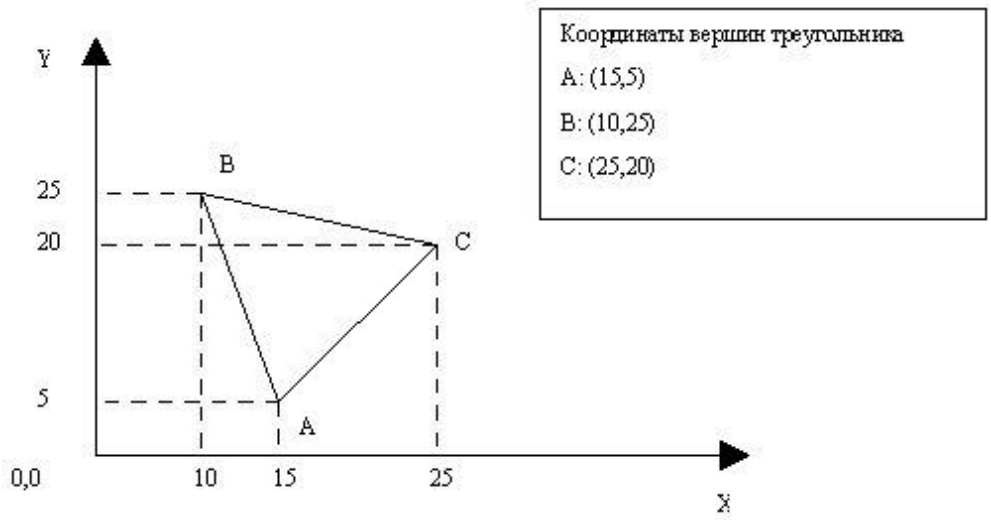

Рис. 2.1. Традиційна система координат

У даній системі координат початок координат розташовано в лівому нижньому кутку системи - саме тут знаходиться точка 0,0. Координати по осі X зростають зліва направо,

по осі Y - від низу до верху.

При розробці комп'ютерних ігор використовується екранна система координат. Головна відмінність цієї системи від традиційної полягає в тому, що початок координат розташовано в лівому верхньому кутку екрану. Координати по осі X зростають зліва направо, а ось вісь Y виявляється перевернутої - її координати зростають зверху вниз. На рис. 2.2. ви можете бачити цю систему координат, в якій намальований той же трикутник.

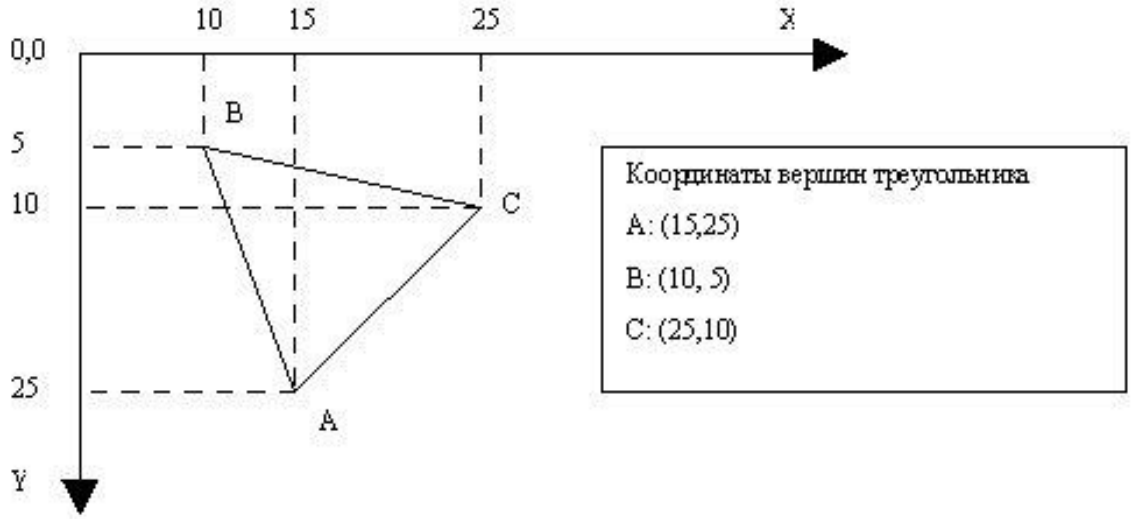

Рис. 2.2. Екранна система координат

Межі екранної системи координат - це або кордону ігровий області вікна - при віконному режимі роботи, або - дозвіл екрана при екранному режимі. Наприклад, якщо в віконному режимі ми маємо розмір вікна, рівним 800x600 пікселів, то саме ці числа є межами системи координат.

Перш ніж займатися роботою з зображеннями в XNA, ці зображення, для початку, потрібно створити. Створимо просте зображення в стандартному графічному редакторі Windows - Paint. Для запуску редактора скористаємося командою Пуск Все програми Стандартні Paint. Намалюємо в редакторі прямокутник (рис. 2.3.), Збережемо його під ім'ям P2_1.png. Як випливає з розширення, ми зберегли малюнок у форматі PNG.

Створимо новий стандартний ігровий проект з ім'ям P2_1. В панелі Solution Explorer клацнемо правою кнопкою миші по папці Content і в контекстному меню виберемо пункт Add Existing Item. З'явиться стандартне вікно для роботи з файлами. Знайдемо з його допомогою графічний файл і додамо в проект. В результаті панель Solution Explorer повинна виглядати так, як зображено на рис. 2.4.

Зверніть увагу на те, що при додаванні зображення в проект, воно фізично копіюється в папку Content, яка знаходиться в папці, що містить файли проекту. Файл, для якого створено проект має те ж саме ім'я, яке він мав при збереженні. Виділивши цей файл, ми можемо переглянути його властивості - у вікні Properties. Це вікно можна бачити на тому ж рис. 2.4. В даному випадку зверніть увагу на параметр Asset Name - тут ми можемо бачити ім'я зображення в проекті. Як і можна було очікувати, воно носить ім'я P2_1.

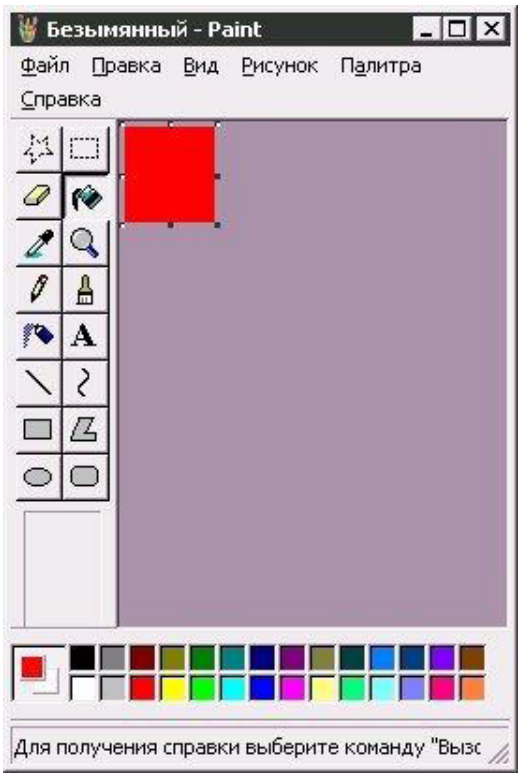

Рис. 2.3. Створюємо просте зображення в редакторі Paint

Тепер виконаємо дії, необхідні для виведення нашого зображення на ігровий екран. Для цього в розділ оголошення властивостей класу додамо дві змінні (лістинг 2.1.).

```
private Texture2D MySprite;
    private Vector2 position = new Vector2(150,200);
```

Лістинг 2.1. Змінні для роботи з зображенням

Перша змінна - MySprite - має тип Texture2D. Цей тип даних призначений для зберігання двовимірних зображень, або текстур. Саме в цю змінну ми будемо завантажувати зображення, яке раніше додали в проект.

Друга змінна - position - потрібна для зберігання позиції, в яку наша текстура буде виведена на екран. Ми відразу ж инициализируем цю змінну значеннями 150,200 - це координати виведення зображення. Координати прив'язані до лівого верхнього кута зображення. Тобто - саме його лівий верхній кут буде знаходитися в позиції 150,200 при виведенні. Якби ми ініціалізували цю змінну значеннями 0,0, зображення було б виведено на початку координат - тобто - його верхній кут знаходився б в позиції 0,0, а саме воно розташовувалося б у верхньому лівому кутку ігрового поля.

Тепер завантажимо текстуру в змінну MySprite. Для цього модифікуємо метод LoadContent наступним чином (лістинг 2.2.).

```
protected override void LoadContent()
    {
        spriteBatch = new SpriteBatch(GraphicsDevice);
        MySprite = Content.Load<Texture2D>(«P2_1»);
            }
```

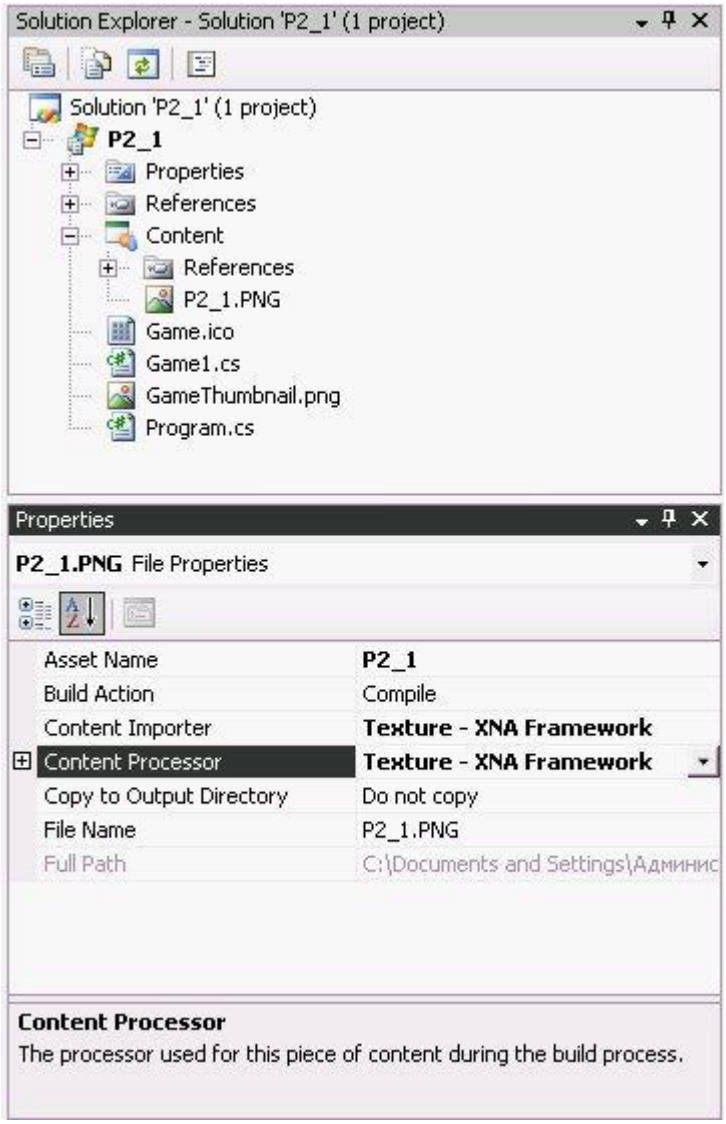

Рис. 2.4. Зображення, додане в проект

Лістинг 2.2. Завантаження зображення до змінної MySprite

Тут, для наочності, видалені стандартні коментарі. Отже, для завантаження ми користуємося методом Load об'єкта Content. Причому, зверніть увагу на те, що метод Load викликається із зазначенням типу завантажується ігрового ресурсу. У нашому випадку це - двовимірне зображення, тому тип ресурсу вказується як Texture2D. Нескладно помітити, що цей тип відповідає типу змінної MySprite.

Тепер настала черга виведення зображення на ігровий екран. Для цього нам знадобиться модифікувати метод Draw. У лістингу 2.3. наведено його код.

```
protected override void Draw(GameTime gameTime)
    {
    graphics.GraphicsDevice.Clear(Color.CornflowerBlue);
        spriteBatch.Begin();
    spriteBatch.Draw(MySprite, position, Color.White);
```

```
spriteBatch.End();
base.Draw(gameTime);
}
```

Лістинг 2.3. Вивід зображення

Метод Begin () об'єкта spriteBatch готує графічний пристрій до висновку зображення. Метод Draw того ж об'єкта приймає в якості параметрів змінну типу Texture2D (MySprite), змінну типу Vector2 (position) і колір, відповідно до якого буде змінений відтінок зображення. В даному випадку це білий колір - це означає, що кольори зображення залишаться неізменнимі.Метод End () завершує процедуру виведення - зображення виводяться на екран.

Метод Draw базового класу в даному випадку не використовується для виведення зображень. Ми розглянемо його роль у виведенні зображень нижче.

Запустивши програму, ми отримаємо таке зображення (рис. 2.5.):

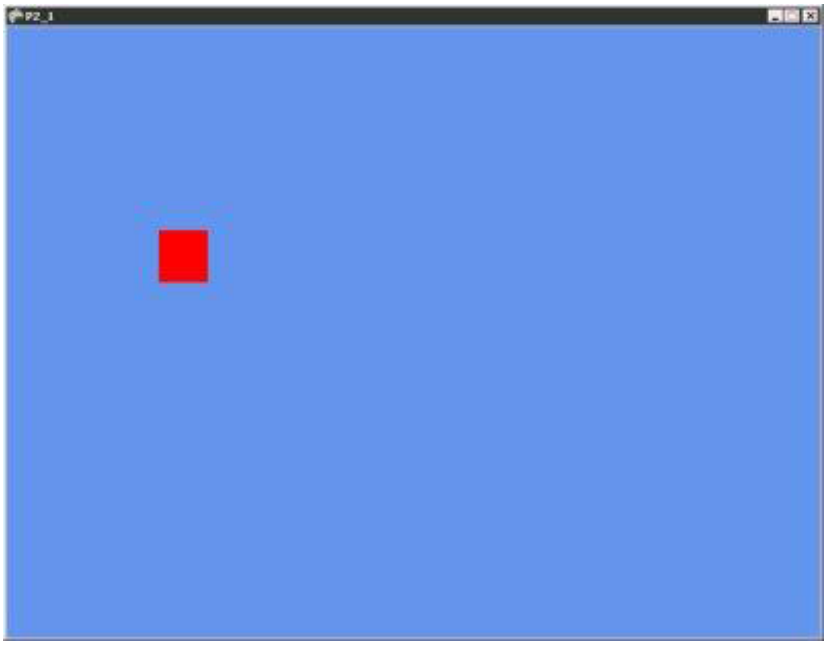

Рис. 2.5. Зображення, яке виведене на игровий екран

Ми розглянули найпростіший спосіб виведення зображення. Однак використання такого способу виведення зображень при розробці реальних ігрових проектів не дуже зручно. Справа в тому, що ігровий об'єкт, який стоїть за кожним зображенням, може мати безліч властивостей. Наприклад - швидкість і напрямок руху. Так само кількість об'єктів в грі може змінюватися. До того ж, не дуже зручно зберігати безліч властивостей різних ігрових об'єктів, використовуючи область оголошення властивостей основного ігрового проекту. Зручніше було б виділити дані про ігрові об'єктах в окремий клас і користуватися об'єктами, згенерували на основі цього класу для виведення зображень і організації ігрової логіки. Власне кажучи, на практиці так і надходять, створюючи окремі ігрові компоненти (класи) для подання ігрових об'єктів. Як правило, в реальних проектах створюється ціла система класів ігрових об'єктів, пов'язаних відносинами спадкування.

Спосіб, описаний вище, може використовуватися для виведення деяких об'єктів, наприклад - так можна вивести фоновий малюнок.

Розглянемо підхід, який дозволяє спростити підхід до висновку зображень методом, описаним вище.

РОЗРОБКА КЛАСУ ДЛЯ ЗБЕРІГАННЯ ГРАФІЧНОЇ ІНФОРМАЦІЇ

Створимо новий стандартний проект з ім'ям P2_2. Додамо в проект зображення, яке ми створили вище (2.1).

Тепер подумаємо, яким повинен бути клас для подання ігрового об'єкта. Зараз нам потрібен клас, об'єкт якого зможе зберігати текстуру спрайту і інформацію про його позиції на екрані. При необхідності цей список можна розширити - причому - як властивостями - так і методами. Але над подібним проектом ми будемо працювати на одному з наступних занять. Обмежимося поки двома вищезгаданими властивостями.

Щелкнем по значку проекту P2_2 в панелі Solution Explorer і виберемо в меню пункт Add Class. З'явиться вікно додавання нового елемента, в якому буде виділено шаблон порожнього класу (рис. 2.6.).

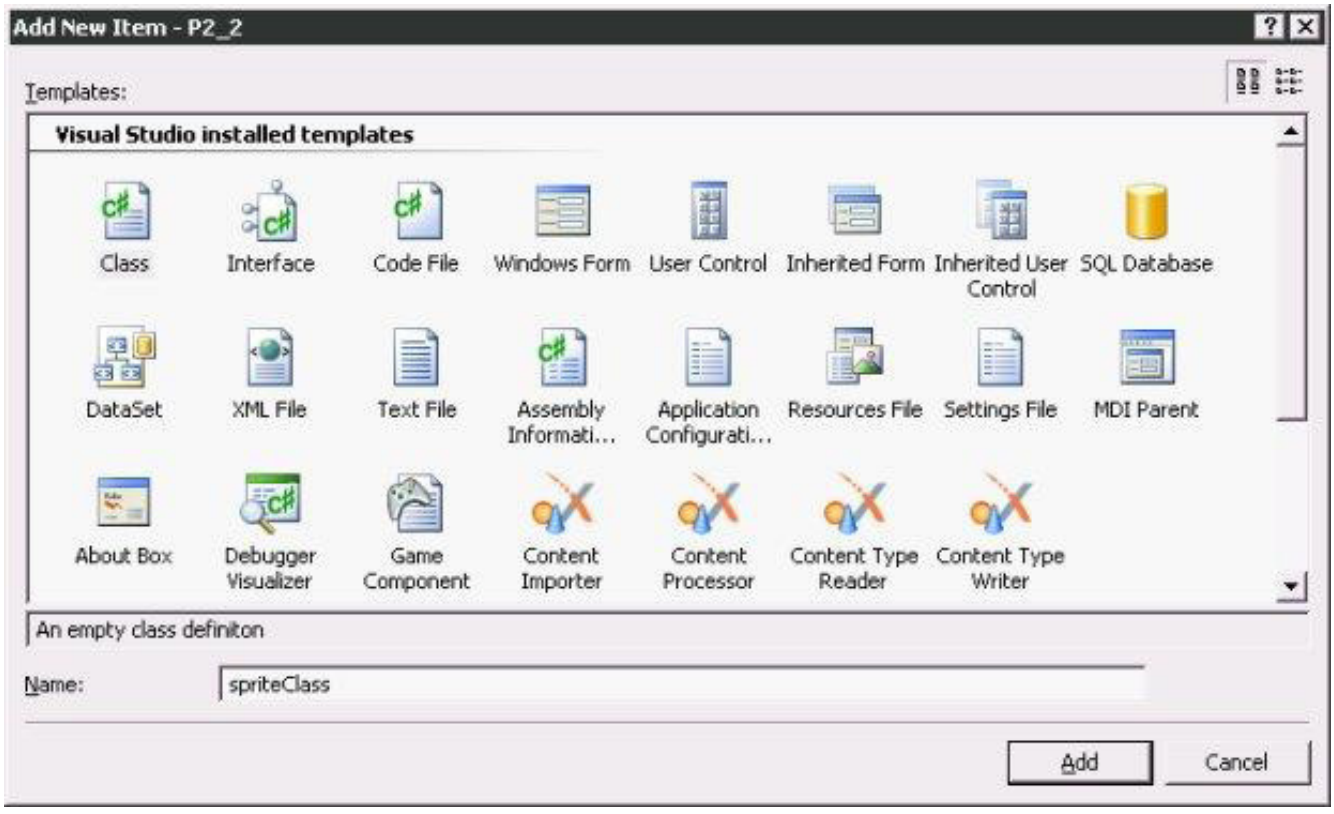

Рис. 2.6. Додаємо до проекту новий клас

Задамо ім'я нового класу - нехай це буде spriteClass, і натиснемо кнопку Add. У проект буде додано новий порожній клас. Ось як виглядає наш проект після всіх перерахованих дій (рис. 2.7.).

Модифікуємо код класу spriteClass наступним чином.

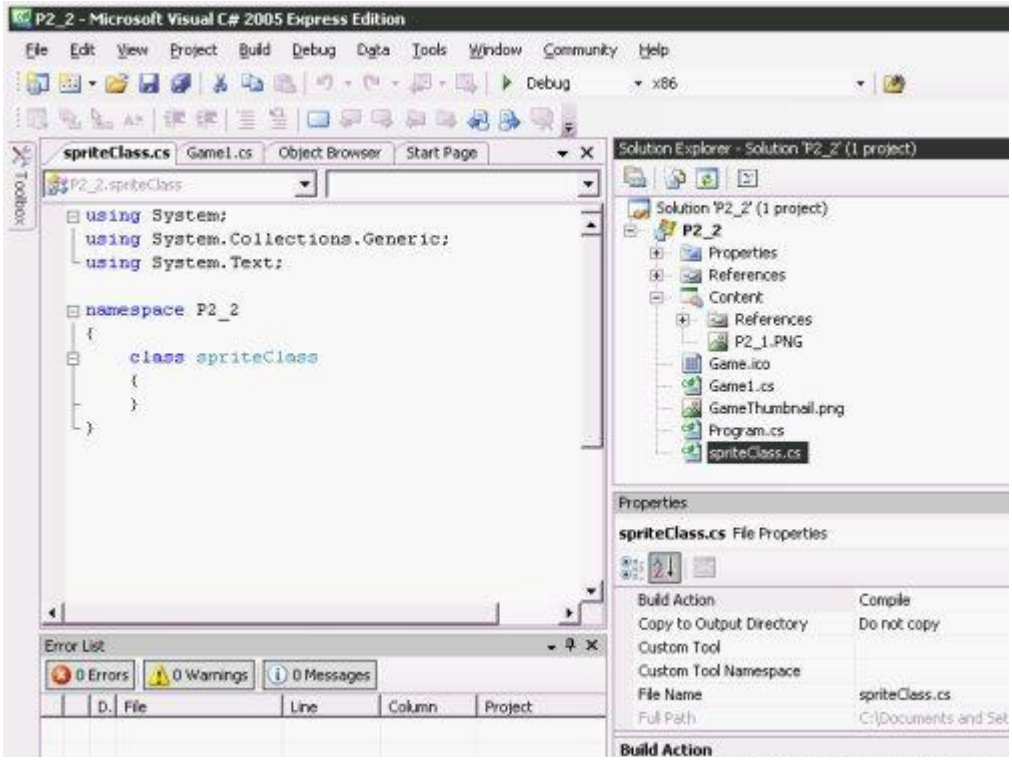

Рис. 2.7. Зовнішній вигляд вікна проекту після додання нового класу

По-перше, в розділ підключення просторів імен додамо директиви using, що містять ті простору імен, які містять оголошення типів даних, необхідних для зберігання текстури і її позиції.

По-друге - створимо необхідні властивості класу.

По-третє - побудуємо конструктор класу, який буде ініціалізувати ці властивості.

Ось як виглядає код класу після всіх модифікацій (лістинг 2.4.):

```
using System;
using System.Collections.Generic;
using System.Text;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework;
namespace P2_2
{
   class spriteClass
   {
      public Texture2D spTexture;
      public Vector2 spPosition;
      public spriteClass(Texture2D newSpTexture, Vector2 newSpPosition)
      {
         spTexture = newSpTexture;
```

```
            spPosition = newSpPosition;
        }
    }
}
```

Лістинг 2.4. Код класу spriteClass

Тут ми підключили до нашого класу простір імен Microsoft.Xna.Framework.Graphics - воно потрібно нам, тому що містить оголошення типу даних Texture2D. Простір імен Microsoft.Xna.Framework містить оголошення типу даних Vector2.

В розділ оголошення властивостей класу ми додали дві змінні. Мінлива spTexture типу Texture2D використовується для зберігання текстури, змінна spPosition типу Vector2 - для зберігання даних про стан об'єкта на екрані. У конструкторі класу ми инициализируем ці змінні значеннями відповідних типів, переданими при створенні об'єкта типу spriteClass в основному тексті програми.

ПРИМІТКА. Властивості класу оголошені з модифікаторами доступності public. Це означає, що ми зможемо модифікувати ці властивості ззовні. В даному випадку такий підхід виправданий. Однак треба зазначити, що при конструюванні класів рекомендується захищати властивості від прямої модифікації або читання. Для організації доступу до таких властивостей можна створювати додаткові властивості-методи, користуючись директивами get / set.

Тепер внесемо зміни в основний текст гри, який розташований в файлі Game1.cs. У лістингу 2.5. наведені ці зміни з коментарями, що вказують на їх місцезнаходження.

```
/// Об'єкт mySpriteObj для зберігання даних про зображення для виведення на екран
spriteClass mySpriteObj;
/// Додаємо в метод LoadContent команду створення нового об'єкта типу spriteClass.
Передаємо в якості текстури зображення текстуру, завантажену з допомогою
команди Content.Load,
 як позиції - новий об'єкт типу Vector2
mySpriteObj = new spriteClass(Content.Load<Texture2D>(«P2_1»), new Vector2(100f,
150f));
/// У метод Draw додаємо відповідні виклики методів об'єкта spriteBatch.
Зверніть увагу на те, що при виклику методу Draw ми користуємося властивостями
об'єкта mySpriteObj.
spriteBatch.Begin();
spriteBatch.Draw(mySpriteObj.spTexture, mySpriteObj.spPosition, Color.White);
spriteBatch.End();
/// У метод UnloadContent додаємо команди для звільнення системних ресурсів.
mySpriteObj.spTexture.Dispose();
spriteBatch.Dispose();
```

Лістинг 2.5. Зміни,які внесені в код файлу Game1.cs

Як бачите, якщо порівняти попередній підхід зі зберіганням усіх даних про виводиться зображенні в класі Game1 і підхід з зберіганням цих же даних в спеціально розробленому класі, можна помітити, що виділення окремого класу для спрайту дозволяє зручно організовувати всі необхідні властивості і методи для роботи з об'єктом .

Однак, і цей підхід не є єдино можливим. Ще більш цікавий спосіб виведення зображень і створення ігрових об'єктів - це створення ігрових об'єктів на базі ігрових компонентів.

РОЗРОБКА ІГРОВОГО КОМПОНЕНТУ

Створимо новий ігровий проект, назвемо його P2_3. Клацнемо правою кнопкою миші по значку проекту в панелі Solution Explorer, виберемо пункт Add New Item. У вікні виберемо мініатюру Game Component, задамо ім'я для компонента - spriteComp і натиснемо на кнопку Add. У проект буде додано новий ігровий компонент. У лістингу 2.6. ви можете бачити повний код цього компонента.

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Storage;
using Microsoft.Xna.Framework.Content;
namespace P2_3
{
   /// <summary>
   /// This is a game component that implements IUpdateable.
   /// </summary>
   public class spriteComp : Microsoft.Xna.Framework.GameComponent
   {
      public spriteComp(Game game)
         : base(game)
      {
         // TODO: Construct any child components here
      }
      /// <summary>
      /// Allows the game component to perform any initialization it needs to before starting
```

```
        /// to run.  This is where it can query for any required services and load content.
        /// </summary>
        public override void Initialize()
        {
            // TODO: Add your initialization code here
            base.Initialize();
        }
        /// <summary>
        /// Allows the game component to update itself.
        /// </summary>
        /// <param name=»gameTime»>Provides a snapshot of timing values.</param>
        public override void Update(GameTime gameTime)
        {
            // TODO: Add your update code here
            base.Update(gameTime);
        }
    }
}
```

Лістинг 2.6. Код ігрового компоненту, який успадкоємлений від GameComponent

Можна відзначити, що ігровий компонент являє собою клас, який успадковує клас Microsoft.Xna.Framework.GameComponent. Спадкування класу GameComponent підходить для створення ігрових компонентів, які не мають графічних уявлень. Ми ж хочемо створити компонент, який планується візуалізувати. Тому потрібно замінити GameComponent на DrawableGameComponent.

Зверніть увагу на те, що створений компонент має кілька методів. Так, метод spriteComp - це конструктор класу, метод Initialize призначений для ініціалізації компонента, метод Update - це ігровий цикл компонента.

Тут не вистачає ще одного методу - методу Draw. Додамо цей метод в клас (після введення тексту public override void нам автоматично запропонують список, з якого досить вибрати Draw і натиснути Enter - набір вхідних змінних і структура методу буде створена автоматично. У лістингу 2.7. Представлений змінений код компонента, готовий для подальшої роботи.

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
```

```
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Storage;
using Microsoft.Xna.Framework.Content;
namespace P2_3
{
   /// <summary>
   /// This is a game component that implements IUpdateable.
   /// </summary>
   public class spriteComp : Microsoft.Xna.Framework.DrawableGameComponent
   {
      public spriteComp(Game game)
         : base(game)
      {
         // TODO: Construct any child components here
      }
      /// <summary>
      /// Allows the game component to perform any initialization it needs to before starting
      /// to run.  This is where it can query for any required services and load content.
      /// </summary>
      public override void Initialize()
      {
         // TODO: Add your initialization code here
         base.Initialize();
      }
      /// <summary>
      /// Allows the game component to update itself.
      /// </summary>
      /// <param name=»gameTime»>Provides a snapshot of timing values.</param>
      public override void Update(GameTime gameTime)
      {
         // TODO: Add your update code here
         base.Update(gameTime);
      }
```

```
        public override void Draw(GameTime gameTime)
        {
            base.Draw(gameTime);
        }
    }
}
```

Лістинг 2.7. Код ігрового компоненту, який успадкоємленеий віт DrawableGameComponent

Вище ми розглядали висновок прямокутного зображення, яке займало всю площу графічного файлу. Нам не потрібно ніяких додаткових налаштувань для виведення такого зображення. На практиці не дуже зручно користуватися безліччю окремих файлів, які зберігають зображення. До того ж, далеко не всі зображення - це прямокутники. Тут нам знадобиться, по-перше - більш досконалий, ніж Paint, інструмент для малювання, а по-друге - дещо інша методика виведення зображень. Раніше для виведення зображення нам була потрібна текстура і координата зображення в ігровому вікні. Тепер же знадобляться координати положення потрібного нам зображення в графічному файлі.

Створимо новий графічний файл. Скористаємося для цього графічним редактором Adobe Photoshop CS. На рис. 2.8. ви можете бачити зображення в вікні редактора. Зверніть увагу на те, що один графічний файл містить декілька зображень, а так само на те, що фон зроблений прозорим - така техніка дозволяє виводити в ігрове вікно малюнки складної форми.

Тут ми створили три зображення - ці зображення на наступних заняттях знадобляться нам для створення власного варіанту класичної гри Pong.

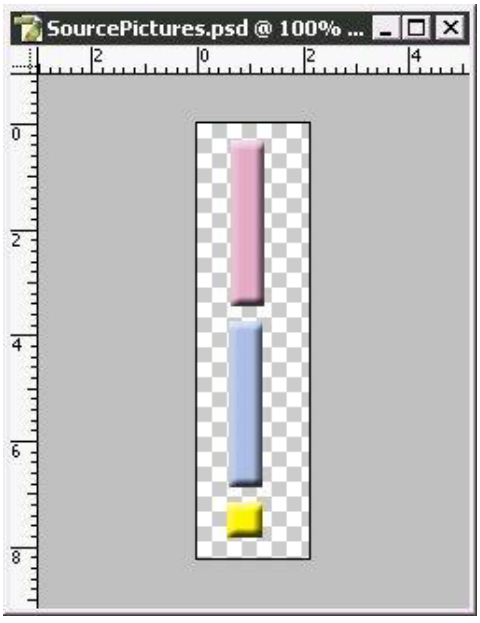

Рис. 2.8. Створення зображень в Adobe Photoshop

Збережемо зображення у вигляді PNG-файлу. Такі файли підтримують прозорість

фону. Тепер нам потрібно вирішити ще одну задачу - визначити координати окремих спрайтів в нашому великому зображенні. Це можна зробити наступним чином. Відкриємо збережене зображення в будь-якому графічному редакторі, можна - в тому ж Photoshop CS - ми розглянемо питання визначення координат саме для цієї програми. Збільшимо масштаб зображення, перемкнемося на панель Info, і, навівши курсор на піксель лівого верхнього кута зображення, запишемо відобразилися координати покажчика (рис. 2.9.). У нашому випадку це (18, 9).

Точно так же дізнаємося координату правого нижнього кута зображення. У нашому випадку це - (35, 97). Для подальшої роботи нам знадобиться знання координати лівої верхньої точки прямокутника, що обмежує спрайт, а так само - його ширина і висота. У нашому випадку ширина становить 17 пікселів (35- 18), висота - 88 пікселів (97 - 9). У підсумку отримуємо наступні параметри прямокутника, в який укладено наш спрайт:

X: 18

Y: 9

Width: 17

Height: 88

Нижче ці дані знадобляться нам.

Тепер завантажимо зображення в проект і продовжимо роботу. Як ви пам'ятаєте, вище ми створили шаблон ігрового компонента, який підходить для виведення на екран. Тепер доопрацюємо цей шаблон. У лістингу 2.8. ви можете бачити доопрацьований код ігрового компонента

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Storage;
using Microsoft.Xna.Framework.Content;

namespace P2_3
{
    /// <summary>
    /// This is a game component that implements IUpdateable.
    /// </summary>
```

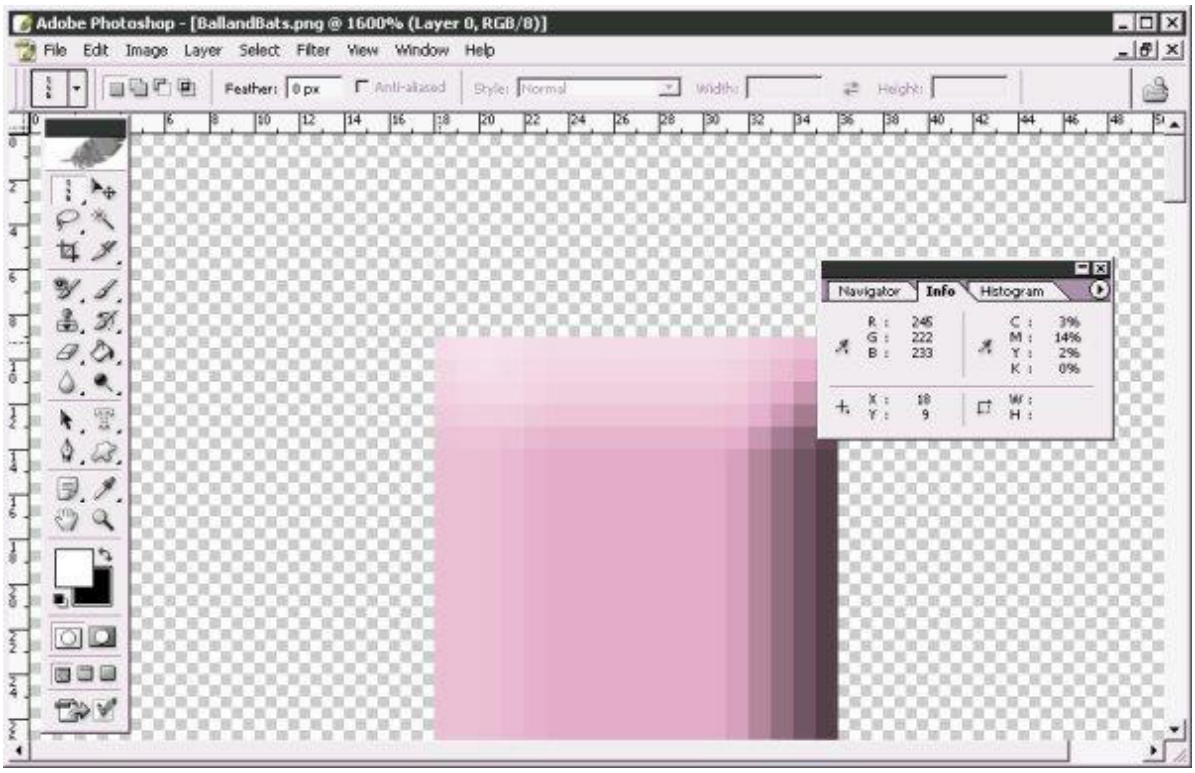

Рис. 2.9. Визначаємо координати спрайта в графічному файлі.

```
public class spriteComp : Microsoft.Xna.Framework.DrawableGameComponent{
private Texture2D sprTexture;
private Rectangle sprRectangle;
private Vector2 sprPosition;
public spriteComp(Game game, ref Texture2D newTexture,
   Rectangle newRectangle, Vector2 newPosition)
   : base(game)
{
   sprTexture = newTexture;
   sprRectangle = newRectangle;
   sprPosition = newPosition;
   // TODO: Construct any child components here
}
/// <summary>
/// Allows the game component to perform any initialization it needs to before starting
/// to run.  This is where it can query for any required services and load content.
/// </summary>
public override void Initialize()
```

```
        {
            // TODO: Add your initialization code here
            base.Initialize();
        }
        /// <summary>
        /// Allows the game component to update itself.
        /// </summary>
        /// <param name=»gameTime»>Provides a snapshot of timing values</param>
        public override void Update(GameTime gameTime)
        {
            // TODO: Add your update code here
            base.Update(gameTime);
        }
        public override void Draw(GameTime gameTime)
        {
            SpriteBatch sprBatch =
                (SpriteBatch)Game.Services.GetService(typeof(SpriteBatch));
            sprBatch.Draw(sprTexture, sprPosition, sprRectangle, Color.White);
            base.Draw(gameTime);
        }
    }
}
```

Лістинг 2.8. Допрацьований код ігрового компоненту.

Розглянемо зміни, внесені в компонент.

Ми додали три змінних - sprTexture, sprRectangle, sprPosition - для зберігання текстури, прямокутника, що показує координати спрайту в текстурі, і позиції виведення спрайту на екран.

Далі, ми модифікували конструктор класу, додавши до списку переданих параметрів наступні змінні:

ref Texture2D newTexture - передача текстури. Параметр передається по посиланню, тобто об'єкт одержує не копію текстури, а лише посилання на неї - це економить системні ресурси.

Rectangle newRectangle - прямокутник, що задає координати спрайту в зображенні.

Vector2 newPosition - позиція спрайту на ігровому екрані.

У тілі конструктора ми ініціалізуємо відповідні властивості класу переданими параметрами.

Далі, ми допрацьовуємо код методу Draw. Тут нам належить найбільш складна частина модифікації. Справа в тому, в грі, для якої ми створюємо компонент, вже є всі засоби для виведення зображень. Тому найрозумніше скористатися вже існуючим об'єктом типу SpriteBatch для того, щоб з його допомогою вивести зображення на екран. Для того, щоб зробити один раз створений SpriteBatch доступним для всіх компонентів гри, існує спеціальна методика. Є таке поняття, як ігрові сервіси, які доступні всім компонентам. Спочатку ми додамо створений в класі Game1 об'єкт типу SpriteBatch в список ігрових сервісів (ми розглянемо це нижче), після чого зможемо витягти цей об'єкт в ігровому компоненті. Ми так і робимо - наступна команда створює новий об'єкт sprBatch, який містить посилання на вихідний об'єкт типу SpriteBatch, який раніше, в класі Game1, доданий в список ігрових ресурсів.

SpriteBatch sprBatch = (SpriteBatch)Game.Services.GetService(typeof(SpriteBatch))

Після того, як у нас з'явилося посилання на вихідний об'єкт типу SpriteBatch, ми можемо вивести зображення, використовуючи перевантажений метод Draw такого об'єкта. Раніше ми користувалися цим методом, передаючи йому текстуру, позицію текстури в ігровому вікні і відтінок текстури. Тепер же до списку переданих йому параметрів додається прямокутник, що задає позицію спрайту в файлі текстури. Цей виклик виглядає так:

sprBatch.Draw(sprTexture, sprPosition, sprRectangle, Color.White);

Вище ми виводили спрайт на екран, спочатку викликавши метод Begin () об'єкта типу SpriteBatch, потім - викликали метод Draw (), потім - метод End (), завершальний висновок. Тут ми не викликаємо цих методів. Справа в тому, що дані методи будуть викликані в методі Draw (), який викликає основна гра (об'єкт типу Game1). У код класу Game1 внесені деякі зміни, які ми зараз розглянемо. У лістингу 2.9. ви можете бачити цей код.

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace P2_3
{
   public class Game1 : Microsoft.Xna.Framework.Game
```

```
{
    GraphicsDeviceManager graphics;
    SpriteBatch spriteBatch;
    spriteComp gameObject;
    Texture2D texture;
    public Game1()
    {
        graphics = new GraphicsDeviceManager(this);
        Content.RootDirectory = «Content»;
    }
    protected override void Initialize()
    {
        // TODO: Add your initialization logic here
        base.Initialize();
    }
    protected override void LoadContent()
    {
        // Create a new SpriteBatch, which can be used to draw textures.
        spriteBatch = new SpriteBatch(GraphicsDevice);
        Services.AddService(typeof(SpriteBatch), spriteBatch);
        texture = Content.Load<Texture2D>(«BallandBats»);
        CreateNewObject();
    }
    protected void CreateNewObject()
    {
        gameObject  = new spriteComp (this, ref texture,
            new Rectangle (18,9,17,88), new Vector2 (100,150));
        Components.Add(gameObject );
    }
    protected override void UnloadContent()
    {
        // TODO: Unload any non ContentManager content here
    }
    protected override void Update(GameTime gameTime)
    {
```

```
        // Allows the game to exit
        if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
            this.Exit();
        // TODO: Add your update logic here
        base.Update(gameTime);
    }
    protected override void Draw(GameTime gameTime)
    {
        graphics.GraphicsDevice.Clear(Color.CornflowerBlue);
        // TODO: Add your drawing code here
        spriteBatch.Begin();
        base.Draw(gameTime);
        spriteBatch.End();
    }
  }
}
```

Лістинг 2.9. Код класу Game1

Для початку ми додали нові змінні, одна з яких (типу spriteComp) служить для зберігання об'єкта класу spriteComp, а друга (типу Texture2D) служить для зберігання текстури. Так як текстура містить кілька спрайтів, змінна Texture2D може знадобитися при створенні декількох об'єктів, тому має сенс створити цю змінну, один раз завантажити в неї зображення і користуватися їй.

У методі LoadContent () ми, як завжди, створюємо об'єкт типу SpriteBatch, і, за допомогою команди Services.AddService (typeof (SpriteBatch), spriteBatch); додаємо об'єкт SpriteBatch до списку ігрових сервісів. Саме після цієї команди spriteBatch можна викликати в ігрових компонентах і виводити їх зображення.

Далі в цьому методі ми завантажуємо текстуру з зображеннями в змінну texture і викликаємо метод CreateNewObject ();. Цей метод ми створили самостійно - ми використовуємо його для створення об'єкта gameObject.

gameObject = new spriteComp (this, ref texture, new Rectangle (18,9,17,88), new Vector2 (100,150));

При створенні цього об'єкта ми передаємо конструктору класу spriteComp об'єкт гри (this), посилання на текстуру, об'єкт Rectangle - зверніть увагу на те, що ми створюємо новий об'єкт Rectangle, який ініціалізований значеннями координати потрібного нам спрайту в файлі текстури, обчисленими раніше. Так само ми передаємо конструктору новий об'єкт типу Vector2, який містить координати спрайту для виведення на ігровий екран.

У методі CreateNewObject () ми розмістили ще одну команду - Components.Add

(gameObject);. З її допомогою ми додаємо в список компонентів гри щойно створений об'єкт. Це дуже важлива команда - завдяки їй при виконанні команди base.Draw (gameTime); буде оброблений метод Draw нашого ігрового компонента, і зображення, які він виведе, будуть виведені на екран.

У методі Draw () класу Game1 ми використовуємо такий код:

```
spriteBatch.Begin() ;
base.Draw(gameTime) ;
spriteBatch.End() ;
```

Команда spriteBatch.Begin (); готує графічний пристрій для виведення зображень, після цього здійснюється висновок зображень - за викликом base.Draw (gameTime); обробляються відповідні методи ігрових компонентів, успадкованих від DrawableGameComponent. Нагадаю, що в методі Draw () нашого компонента spriteComp є лише виклик методу Draw () об'єкта класу SpriteBatch, без викликів Begin () і End (). В результаті, після того, як компонент виведе зображення, висновок завершується командою End () об'єкта spriteBatch в методі Draw () класу Game1 і зображення з'являється на екрані.

Такий механізм роботи дозволяє створювати безліч компонентів, додавати їх в список компонентів гри і не піклуватися про виведення кожного з них в методі Draw () класу Game1. Після того, як компонент доданий в список компонентів гри, висновок його графічного зображення здійснюється автоматично. Можна сказати, що метод base. Draw (gameTime); «Переглядає» всі зареєстровані компоненти і виводить їх на екран.

Саме такий підхід найзручніше використовувати при розробці реальних ігор.

Ось як виглядає ігровий екран після виведення в нього зображення (рic.2.10.)

Рис. 2.10. Ігровий екран після виводу компонента.

Тут ми розглянули різні підходи до висновку зображень. Однак, можна помітити,

що виведення зображення, яке асоціюється з ігровим об'єктом - це лише одне із завдань, яке доводиться вирішувати при розробці гри. Інші завдання - наприклад - переміщення об'єкта, перевірка, не зіткнувся чи об'єкт з іншими об'єктами - так само можна вирішувати з використанням різних підходів. Можна помітити, що найбільш складний з точки зору реалізації підхід - розробка окремого ігрового компоненту, є одночасно і найбільш зручним для реалізації різних можливостей ігрового об'єкта. Пізніше, допрацьовує функціональність ігрових об'єктів, ми будемо користуватися саме цим підходом.

Розглянемо висновок декількох зображень в ігрове вікно. Як ми вже говорили, з використанням зареєстрованих ігрових компонентів, успадкованих від DrawableGameComponent, висновок автоматизований. Однак це не означає, що використовувати більш прості способи вирішення не потрібно.

Наприклад, розглянемо висновок фонового зображення в уже створеному проекті P2_3. Вище ми створили зображення, які можна використовувати для створення власної версії гри Pong. Точно так же створимо зображення, яке буде служити фоном для гри. Стандартний ігровий екран має роздільну здатність 800x600 пікселів. Створимо зображення такого розміру (назвемо його background.png), завантажимо його в проект. Для цього додамо в клас нову змінну

Texture2D backTexture;

Після цього, в методі LoadContent (), завантажимо зображення в цю змінну командою

backTexture = Content.Load <Texture2D> («Background»);

Тепер виведемо зображення за допомогою методу Draw () об'єкта spriteBatch.

spriteBatch.Draw (backTexture, new Rectangle (0, 0, 800, 600), Color.White);

Помістимо цю команду між викликами Begin () і End () перед командою base.Draw (gameTime);

В результаті вікно гри набуде такого вигляду (рис. 2.11.).

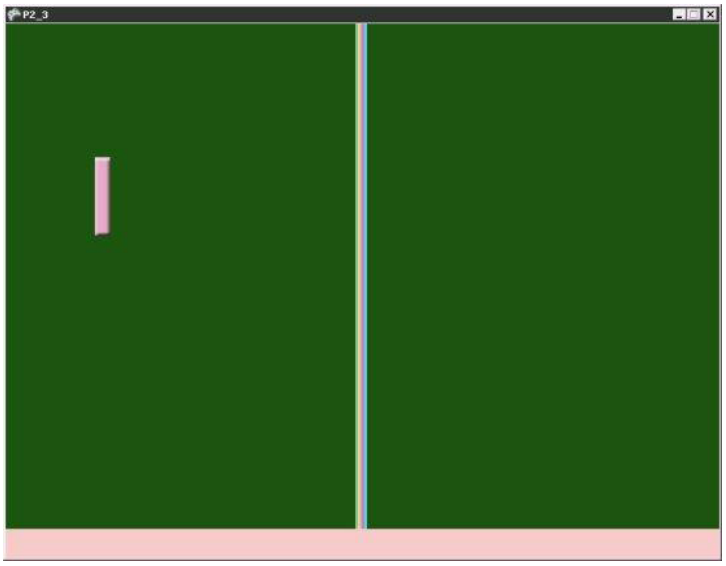

Рис. 2.11. Вікно виводу гри після додавання фону.

Як бачите, спрайт ігрового об'єкта виведено поверх фонового зображення.

ПРИМІТКА. Підхід, який вимагає дотримання черговості виведення зображень для регулювання їх накладення, не дуже зручний. Однак, XNA Framework передбачає спеціальні інструменти для настройки взаємного розташування об'єктів. Зокрема, при виведенні спрайту можна вказувати «глибину» (depth), на якій він розташований.

ЛАБОРАТОРНА РОБОТА 3. ПРИСТРОЇ ВВЕДЕННЯ, ПЕРЕМІЩЕННЯ ОБ'ЄКТІВ.

Анотація: У цій лабораторній роботі ми ознайомимося з основними способами роботи з пристроями введення. Зокрема, поговоримо про переміщення об'єктів, обговоримо роботу з різними пристроями введення і питання автоматичного переміщення об'єктів.

Ключові слова: об'єкт, ПО, одиниця, Left, координати, right, поле, значення, mouse, спрайт, покажчик, this, відображення, рівність, змінна, різниця, циклу, компонент, конструктор, перетин, алгоритм, клас, механізми, висновок , virtual, дочірній клас, виклик методу, конструктор класу, базовий клас, список, маніпулятор.

Задачі роботи:

- Навчитися отримувати дані про стан миші і клавіатури
- Навчитися переміщати ігрові об'єкти відповідно до параметрів, отриманими від пристрою введення
- Навчитися переміщати ігрові об'єкти в автоматичному режимі, використовуючи засоби ігрового циклу
- Навчитися організовувати управління декількома об'єктами для застосування в іграх, в які можуть грати два гравці, які користуються одним комп'ютером
- Навчитися впізнавати і змінювати розміри ігрового вікна
- Навчитися переводити програму в повноекранний режим, налаштовувати дозвіл екрана при роботі в такому режимі
- Навчитися запобігати перетин об'єктом кордонів екрану при переміщенні об'єкта

ОБРОБКА СТАНУ КЛАВІАТУРИ

Створимо новий стандартний проект, назвемо його P3_1. Додамо в проект зображення, змінні для зберігання текстури і позиції зображення в ігровому вікні, завантажимо його в методі LoadContent. Ось, як виглядає текст програми після модифікації (лістинг 3.1.).

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;
namespace P3_1
{
```

```
/// <summary>
/// This is the main type for your game
/// </summary>
public class Game1 : Microsoft.Xna.Framework.Game
{
    GraphicsDeviceManager graphics;
    SpriteBatch spriteBatch;
    private Texture2D MySprite;
    private Vector2 position = new Vector2(150, 200);
    public Game1()
    {
        graphics = new GraphicsDeviceManager(this);
        Content.RootDirectory = «Content»;
    }
    protected override void Initialize()
    {
        // TODO: Add your initialization logic here
        base.Initialize();
    }
    protected override void LoadContent()
    {
        spriteBatch = new SpriteBatch(GraphicsDevice);
        MySprite = Content.Load<Texture2D>(«ball»);
    }
    protected override void UnloadContent()
    {
        // TODO: Unload any non ContentManager content here
    }

    protected override void Update(GameTime gameTime)
    {
        // Allows the game to exit
        if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
            this.Exit();
```

```
            // TODO: Add your update logic here
            base.Update(gameTime);
        }

        protected override void Draw(GameTime gameTime)
        {
            graphics.GraphicsDevice.Clear(Color.CornflowerBlue);
            spriteBatch.Begin();
            spriteBatch.Draw(MySprite, position, Color.White);
            spriteBatch.End();
            // TODO: Add your drawing code here
            base.Draw(gameTime);
        }
    }
}
```

Лістинг 3.1. Код класу Game1

Тепер додамо в метод Update код, який буде змінювати позицію виведеного спрайту відповідно до зміни стану клавіатури. Зокрема, буде реагувати на клавіші-стрілки, переміщаючи об'єкт у відповідному напрямку. На рис. 3.1. ви можете бачити зовнішній вигляд ігрового вікна

Ігрове вікно

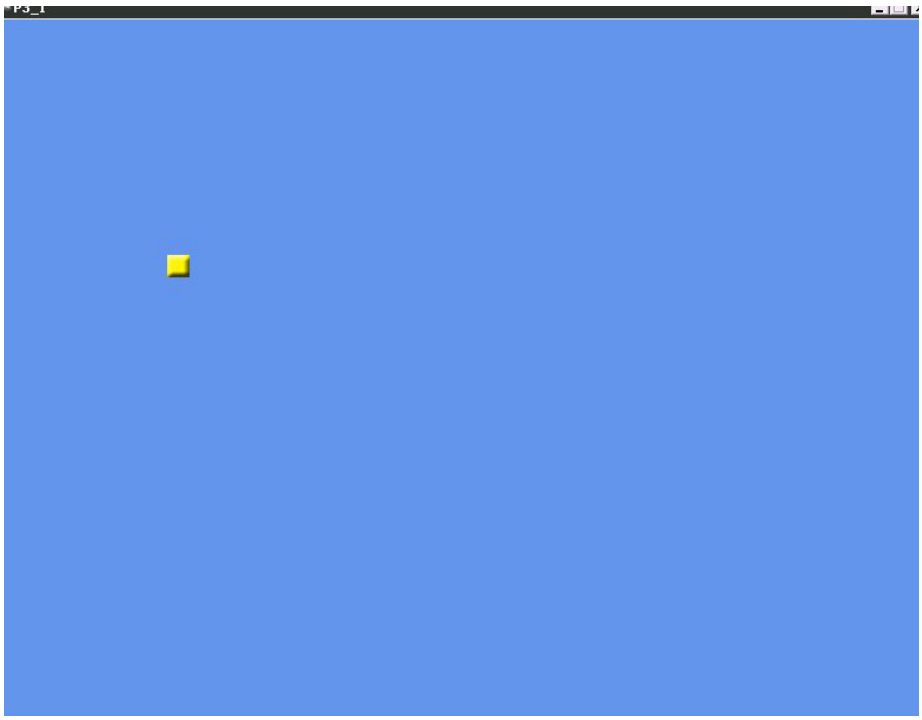

Рис. 3.1. Ігрове вікно

Для того, щоб налаштувати переміщення спрайту по клавіатурним командам, нам знадобиться такий код (лістинг 3.2.).

```
KeyboardState kbState = Keyboard.GetState();
        if (kbState.IsKeyDown(Keys.Up))
           position.Y -= 1;
        if (kbState.IsKeyDown(Keys.Down))
           position.Y += 1;
        if (kbState.IsKeyDown(Keys.Left))
           position.X -= 1;
        if (kbState.IsKeyDown(Keys.Right))
           position.X += 1;
```

Лістинг 3.2. Код, який реалізує читання стану клавіатури і переміщення спрайту.

Тут ми спочатку створили змінну kbState типу KeyboardState, в яку записали поточний стан клавіатури, викликавши метод GetState () об'єкта KeyBoard. Цей метод повертає стан клавіатури.

Після цього ми перевірили, за допомогою методу IsKeyDown, чи була натиснута клавіша-стрілка «Вгору». Найменування клавіш зберігаються в перерахуванні Keys. У нашому випадку клавіша-стрілка «вгору» символізується ім'ям Keys.Up. Якщо клавіша натиснута - змінимо координату Y позиції об'єкта, вирахувавши з неї 1. Позиція зміниться і при наступному виведенні спрайту він переміститься на 1 піксель вгору. Точно так само обробляються інші клавіші-стрілки. Keys.Down - це клавіша «вниз», якщо вона натиснута, до координати Y додається одиниця і об'єкт переміщається вниз. Keys.Left - це клавіша «вліво» - об'єкт переміщається вліво - від його координати X віднімається одиниця. Keys.Right - це клавіша «вправо» - об'єкт переміщається вправо, до його координаті X додається 1.

Якщо натиснути і утримувати одну з клавіш - об'єкт буде безперервно переміщатися в зазначеному напрямку. Якщо, не відпускаючи натиснуту клавішу, натиснути іншу, об'єкт змінить напрямок або зупиниться. Наприклад, натиснувши одночасно клавіші «вгору» і «вліво», ми змусимо об'єкт рухатися по діагоналі вліво і вгору. А якщо в той час, як натиснута кнопка «вгору» ми натиснемо і клавішу «униз» - об'єкт зупиниться - натискання «вгору» віднімає одиницю з його позиції, натискання «вниз» - додає одиницю, в результаті об'єкт виявляється нерухомим.

За допомогою методу IsKeyUp об'єкта kbState ми проводимо перевірку, зворотний методу IsKeyDown - тобто цей метод повертає True, якщо не натискати жодної клавіші, і False - якщо натиснута.

Для переміщення об'єкта ми модифікуємо його координати на ігровому полі. Можна помітити, що модифікація ведеться з кроком в 1 Це значення можна назвати швидкістю переміщення об'єкта. Збільшивши крок зміни координати після натискання клавіші, можна збільшити швидкість переміщення.

Можна помітити, що описаний підхід до переміщення об'єкта по клавіатурним командам працює, однак якщо складність ігрової програми збільшиться - він виявляється незручним. Тому нижче ми розглянемо приклад використання керованого з клавіатури ігрового компонента, успадкованого від DrawableGameComponent. Такий підхід робить зручним не тільки клавіатурне управління, а й автоматичне керування об'єктом, робить можливим зручне застосування обмежень на управління об'єктом, обробку зіткнень об'єктів (це питання буде розглянуто на наступному занятті) та інші завдання. Однак, варто зазначити, що принципи управління об'єктами з клавіатури навіть при управлінні ігровим компонентом, зберігаються.

Тепер розглянемо управління ігровим об'єктом за допомогою миші.

ОБРОБКА СТАНУ МИШІ

Створимо новий стандартний ігровий проект (P3_2), модифікуємо його таким чином, щоб він відповідав коду, наведеному в лістингу 3.1. - тобто - підготуємо все для розгляду концепцій роботи з мишею. Тепер додамо в код методу Update () такий код (лістинг 3.3.).

```
MouseState mState = Mouse.GetState();
    position.X = mState.X;
    position.Y = mState.Y;
```

Лістинг 3.3. Код, який реалізує читання стану миші і переміщення об'єкта.

Для початку ми створили змінну mState типу MouseState - вона зберігає стан миші, отримане за допомогою методу GetState () об'єкта Mouse. Тепер ми прирівнюємо координату X позиції спрайту поточної координаті X миші, те ж саме робимо з координатою Y.

Запустивши на виконання даний приклад, можна помітити, що спрайт переміщається відповідно до переміщеннями миші, при виході покажчика миші за межі ігрового вікна, спрайт так само «йде» з екрану. Нижче ми розглянемо приклад, який показує, як уникнути такого «відходу».

Стан миші, крім ее координат, містить і іншу інформацію. Зокрема - дані про натискання клавіш, про стан колесіка прокрутки. Створимо приклад, який демонструє переміщення спрайту до позиції ігрового вікна, в якій ми клацнули мишею.

Створимо новий стандартний ігровий проект (P3_3), код якого аналогічний лістингу 3.1.

Додамо в розділ оголошення змінних класу дві змінних (лістинг 3.4.).

```
bool is_MouseLB_Pressed = false;
    Vector2 new_position = new Vector2();
```

Лістинг 3.4. Змінні для визначення факту натискання на ліву кнопку миші і позиції, в якій було скоєно натискання.

Першу змінну - is_MouseLB_Pressed будемо встановлювати в True при натисканні на ліву кнопку миші, в змінну new_position будемо записувати координати, в яких

відбулося натискання.

Додамо в метод Initialise () класу Game1 код, який відобразить покажчик миші в ігровому вікні. За замовчуванням при переміщенні миші в межі ігрового вікна покажчик зникає. Код, наведений у лістингу 3.5. дозволяє відобразити покажчик..

```
protected override void Initialize()
    {
      base.Initialize();
      this.IsMouseVisible = true;
    }
```

Листинг 3.5. Код для відображення покажчика миші в ігровому вікні

Тут об'єкт this являє собою поточний ігровий об'єкт. Його властивість IsMouseVisible при установці в True включає відображення покажчика в ігровому вікні.

Тепер доповнимо метод Update () так, як показано в коді, наведеному в лістингу 3.6.

```
protected override void Update(GameTime gameTime)
    {
      // Allows the game to exit
      if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        this.Exit();
      MouseState mState = Mouse.GetState();
      if (mState.LeftButton ==ButtonState .Pressed)
      {
        is_MouseLB_Pressed = true;
        new_position.X = mState.X;
        new_position .Y = mState .Y;
      }
      if (is_MouseLB_Pressed)
      {
        if (new_position .X -position .X >0)
        {
          position .X +=1;
        }
        else
        {
          position .X-=1;
        }
```

```
        if (new_position .Y-position .Y>0)
        {
           position .Y+=1;
        }
        else
        {
           position .Y -=1;
        }
        if (position == new_position)
        {
           is_MouseLB_Pressed = false;
        }
     }
  }
```

Лістинг 3.6. Код методу Update для автоматичного переміщення об'єкта до місця клацання лівої кнопки миші.

Спочатку ми отримуємо поточний стан миші і зберігаємо його в змінної mState. Далі перевіряємо, чи була натиснута ліва кнопка миші. Для цього використовуємо властивість mState.LeftButton і перевіряємо його рівність ButtonState.Pressed. Якщо рівність виконується - встановлюємо змінну is_MouseLB_Pressed в True і записуємо в змінну new_Position значення координат миші в момент натискання лівої кнопки.

Далі, якщо змінна is_MouseLB_Pressed встановлена в True, порівнюємо поточну позицію об'єкта і позицію, в якій ми клацнули лівою кнопкою. Якщо різниця координати X нової та поточної позицій більше нуля, це означає, що об'єкт розташований лівіше нової позиції і для переміщення до неї його координата X повинна бути збільшена. Якщо різниця менше нуля - об'єкт знаходиться правіше, його координата X буде зменшена. Точно так же порівнюємо координати Y і модифікуємо їх. В кінці ми перевіряємо, чи досяг об'єкт заданої позиції. Якщо так - встановлюємо is_MouseLB_Pressed в False - це значить, що на наступному кроці циклу Update () ми не будемо змінювати координати - об'єкт досяг заданої позиції.

При запуску програми об'єкт нерухомий. Якщо ми клацнемо мишею в ігровому вікні - він почне переміщатися в позицію клацання. Якщо об'єкт уже рухається у напрямку до позиції, в якій ми клацнули мишею, а ми в цей час клацаємо мишею в новій позиції - він змінює напрямок руху і рухається до нової позиції.

Тут реалізований простий приклад автоматичного переміщення об'єкта.

Як згадувалося вище, підхід з розміщенням коду переміщення ігрового об'єкта в основному коді програми незручний при розробці реальних проектів. Тому нижче ми розглянемо приклад розробки самостійного ігрового об'єкта, що містить власний код,

який відповідає за його переміщення. Тут же ми реалізуємо перевірку на допустимість наступного кроку переміщення.

Розробка ігрового компонента з функціями переміщення і з обмеженнями

Можна помітити, що в попередніх прикладах ігрові об'єкти легко перетинали кордону ігрового поля. У реальних проектах зазвичай потрібно, щоб рухливий об'єкт не перетинав цих кордонів. Це означає, що, по-перше - нам потрібно дізнатися координати меж екрану, а по-друге - потрібно створити такий код, який відповідає за переміщення об'єкта, який перш ніж перемістити об'єкт в нову позицію, перевіряв би допустимість такого переміщення.

Створимо новий ігровий проект (P3_4), аналогічний проекту P2_3, розробленим в лабораторній роботі №2. Єдина відмінність - ми не будемо виводити фонове зображення. Цей проект містить ігровий компонент, який в нашому прикладі і буде містити весь необхідний код. Змінений код компонента ви можете бачити в лістингу 3.7.

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Storage;
using Microsoft.Xna.Framework.Content;
namespace P3_4
{
   /// <summary>
   /// This is a game component that implements IUpdateable.
   /// </summary>
   public class spriteComp : Microsoft.Xna.Framework.DrawableGameComponent
   {
      private Texture2D sprTexture;
      private Rectangle sprRectangle;
      private Vector2 sprPosition;
      private Rectangle scrBounds;
      public spriteComp(Game game, ref Texture2D newTexture,
         Rectangle newRectangle, Vector2 newPosition)
```

```
    : base(game)
{
    sprTexture = newTexture;
    sprRectangle = newRectangle;
    sprPosition = newPosition;
    scrBounds = new Rectangle(0, 0,
        game.Window.ClientBounds.Width,
        game.Window.ClientBounds.Height);
    // TODO: Construct any child components here
}
public override void Initialize()
{
    // TODO: Add your initialization code here
    base.Initialize();
}
public override void Update(GameTime gameTime)
{
    KeyboardState kbState = Keyboard.GetState();
    if (kbState.IsKeyDown(Keys.Up))
    {
        sprPosition.Y -= 5;
    }
    if (kbState.IsKeyDown(Keys.Down))
    {
        sprPosition.Y += 5;
    }
    if (kbState.IsKeyDown(Keys.Left))
    {
        sprPosition.X -= 5;
    }
    if (kbState.IsKeyDown(Keys.Right))
    {
        sprPosition.X += 5;
    }
```

```
        if (sprPosition.X < scrBounds.Left)
        {
            sprPosition.X = scrBounds.Left;
        }
        if (sprPosition.X > scrBounds.Width - sprRectangle.Width)
        {
            sprPosition.X = scrBounds.Width - sprRectangle.Width ;
        }
        if (sprPosition.Y < scrBounds.Top)
        {
            sprPosition.Y = scrBounds.Top;
        }
        if (sprPosition.Y > scrBounds.Height - sprRectangle.Height )
        {
            sprPosition.Y = scrBounds.Height - sprRectangle.Height;
        }
        base.Update(gameTime);
    }
    public override void Draw(GameTime gameTime)
    {
        SpriteBatch sprBatch =
            (SpriteBatch)Game.Services.GetService(typeof(SpriteBatch));
        sprBatch.Draw(sprTexture, sprPosition, sprRectangle, Color.White);
        base.Draw(gameTime);
    }
  }
}
```

Лістинг 3.7. Код компонента, розрахованого на переміщення з обмеженнями.

У розділі оголошення змінних ми додали нову змінну - scrBounds типу Rectangle. У цієї змінної ми будемо зберігати параметри ігрового вікна. Ми инициализируем цю змінну в конструкторі класу. Як лівій верхній координати прямокутника, що обмежує ігрове вікно, ми записали (0,0), в якості ширини - ширину ігрового поля, отриману за допомогою команди game.Window.ClientBounds.Width. Тут об'єкт game був переданий в конструктор при створенні ігрового об'єкта, об'єкт Window - це вікно гри, ClientBounds містить інформацію про параметри вікна, зокрема, про ширину (Width) і висоті (Height).

У методі Update () ми отримуємо стан клавіатури і, якщо натиснуті відповідні клавіші-стрілки, модифікуємо позицію спрайту на екрані. Однак тут, після модифікації позиції, ми проводимо серію перевірок. Наприклад, перевіряємо, чи не менше нова координата X ігрового об'єкта координати X прямокутника, відповідного ігрового вікна. Якщо ця координата менше, це означає, що об'єкт, при переміщенні в відповідну позицію, виявиться лівіше лівої межі екрану. Тому, якщо перевіряється умова виконується - ми прирівнюємо координату лівої межі екрану координаті об'єкту. При спробі перетину лівої межі екрану об'єкт «впирається» в неї.

Трохи складніше виглядає перевірка на перетин правої межі екрану. Як ви знаєте, координати X і Y прямокутника, що обмежує наш об'єкт, відповідають його лівій верхній точці. Об'єкт має ширину і висоту, тому для перевірки перетину правої межі екрану, ми порівнюємо нову позицію об'єкта з різницею ширини екрану з шириною об'єкта. Якщо виявляється, що нова позиція більше, ніж обчислена різниця - об'єкт встановлюється в позицію, координата X об'єкта встановлюється рівною різниці ширини екрану і ширини об'єкта - він «упирається» в праву межу екрану своєю правою частиною.

Таким же чином перевіряється координата Y - на перетин об'єкта верхньої і нижньої меж екрану.

Як бачите, вся логіка по переміщенню об'єкта і по перевірці на допустимість переміщення, міститься в ігровому компоненті і не перевантажує основний програмний код. Такий підхід має свої плюси і мінуси. Плюс полягає в зручності використання подібної конструкції. Але якщо ми створимо кілька ігрових об'єктів на основі ігрового компонента, то виявиться, що всі вони переміщаються одночасно після натискання на клавіші-стрілки.

Однаковий механізм переміщення був би зручний, якби ми створили алгоритм автоматичного переміщення об'єкта, який переміщував б кожен з них відповідно до якимось законом, наприклад - випадковим чином. Якщо ж нам потрібно створити кілька об'єктів, що володіють самостійним управлінням, використовуючи один і той же клас, цей механізм буде потребувати переробці. Наприклад, в грі Pong передбачається управління двома бітами - два користувача, сидячи за однією клавіатурою, керують кожен своєю битою з використанням власних клавіш. Розглянемо механізми організації управління декількома об'єктами.

Керування кількома об'єктами: система класів.

Мабуть, перша думка, яка прийде в голову при розробці роздільного управління двома об'єктами - створити другий ігровий компонент, що відрізняється від першого лише реакцією на натискання клавіш. Наприклад, перший буде управлятися за допомогою клавіш-стрілок, другий - за допомогою традиційного для ігор набору клавіш ASWD. Грамотніше всього в такій ситуації буде створити ігровий компонент, який не містить клавіатурних команд, після чого створити ще два компонента, кожен з яких успадкує вихідний компонент і перевизначити його метод, відповідальний за обробку подій клавіатури.

Створимо новий проект (P3_5), аналогічний проекту P2_3 (лише відключимо висновок

фону). Ігровий spriteComp буде служити основою для двох інших компонентів. Змінимо його код так, як показано в лістингу 3.8.

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Storage;
using Microsoft.Xna.Framework.Content;
namespace P3_5
{
   /// <summary>
   /// This is a game component that implements IUpdateable.
   /// </summary>
   public class spriteComp : Microsoft.Xna.Framework.DrawableGameComponent
   {
      protected  Texture2D sprTexture;
      protected  Rectangle sprRectangle;
      protected  Vector2 sprPosition;
      protected  Rectangle scrBounds;
      public spriteComp(Game game, ref Texture2D newTexture,
         Rectangle newRectangle, Vector2 newPosition)
         : base(game)
      {
         sprTexture = newTexture;
         sprRectangle = newRectangle;
         sprPosition = newPosition;
         scrBounds = new Rectangle(0, 0,
            game.Window.ClientBounds.Width,
            game.Window.ClientBounds.Height);
         // TODO: Construct any child components here
      }
```

```
public override void Initialize()
{
    // TODO: Add your initialization code here
    base.Initialize();
}
public virtual void Move()
{
    Check();
}
void Check()
{
    if (sprPosition.X < scrBounds.Left)
    {
        sprPosition.X = scrBounds.Left;
    }
    if (sprPosition.X > scrBounds.Width - sprRectangle.Width)
    {
        sprPosition.X = scrBounds.Width - sprRectangle.Width;
    }
    if (sprPosition.Y < scrBounds.Top)
    {
        sprPosition.Y = scrBounds.Top;
    }
    if (sprPosition.Y > scrBounds.Height - sprRectangle.Height)
    {
        sprPosition.Y = scrBounds.Height - sprRectangle.Height;
    }
}
public override void Update(GameTime gameTime)
{
    // TODO: Add your update code here
    Move();
    base.Update(gameTime);
}
```

```
        public override void Draw(GameTime gameTime)
        {
            SpriteBatch sprBatch =
                (SpriteBatch)Game.Services.GetService(typeof(SpriteBatch));
            sprBatch.Draw(sprTexture, sprPosition, sprRectangle, Color.White);
            base.Draw(gameTime);
        }
    }
}
```

Лістинг 3.8. Базовий компонент spriteComp.

Зверніть увагу на те, що цей компонент практично в точності схожий на компонент, код якого приведений в лістингу 3.7. Ключове відмінність - видалення з методу Update () коду, пов'язаного з переміщенням. Цього коду тут зовсім немає. Зате ми додали в Update () виклик нового віртуального методу Move (). З назви методу можна судити про те, що його планується використовувати для управління переміщенням об'єкта. Метод не випадково оголошений з ключовим словом virtual - в дочірніх класах ми перевизначити цей метод для вирішення завдань кожного з них. У батьківському класі ми додали в нього виклик методу Check (). Цей метод однаковий для обох об'єктів - він не допускає перетинання ними кордонів екрану.

Додамо в проект новий ігровий компонент, назвемо його gameObj1.cs і відповідним чином модифікуємо. У лістингу 3.9. ви можете бачити його код, нижче читайте пояснення.

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Storage;
using Microsoft.Xna.Framework.Content;
namespace P3_5
{
    public class gameObj1 : spriteComp
    {
        public gameObj1(Game game, ref Texture2D newTexture,
```

```
    Rectangle newRectangle, Vector2 newPosition)
    : base(game, ref newTexture , newRectangle , newPosition )
{
    // TODO: Construct any child components here
}
public override void Move()
{
    KeyboardState kbState = Keyboard.GetState();
    if (kbState.IsKeyDown(Keys.Up))
    {
        sprPosition.Y -= 5;
    }
    if (kbState.IsKeyDown(Keys.Down))
    {
        sprPosition.Y += 5;
    }
    if (kbState.IsKeyDown(Keys.Left))
    {
        sprPosition.X -= 5;
    }
    if (kbState.IsKeyDown(Keys.Right))
    {
        sprPosition.X += 5;
    }
    base.Move();
}
public override void Initialize()
{
    // TODO: Add your initialization code here
    base.Initialize();
}
public override void Update(GameTime gameTime)
{
    // TODO: Add your update code here
```

```
            base.Update(gameTime);
        }
        public override void Draw(GameTime gameTime)
        {
            base.Draw(gameTime);
        }
    }
}
```

Листинг 3.9. Клас gameObj1 – спадкоємець spriteComp

Так як доданий в гру компонент був спочатку успадкований від GameComponent - він не мав перевизначеного методу Draw (). Додамо цей метод в компонент.

Тепер модифікуємо конструктор класу. Все, що потрібно зробити тут - отримати ті ж параметри, що і базовий клас і передати їх його конструктору.

Далі - перевизначити метод Move () - а саме - додамо в нього команди обробки введення з клавіатури. Це вже знайомі вам команди, які змінюють положення об'єкта на 5 пікселів в залежності від натиснутоюклавіші-стрілки.

Додамо в проект ще один ігровий компонент, назвемо його gameObj2.cs, проведемо з ним ті ж маніпуляції, які провели з компонентом gameObj1.cs, а ось метод Move () перевизначити по-іншому. Нагадаю, що нам потрібно, щоб цим компонентом можна було управляти не за допомогою клавіш-стрілок, а клавішами ASWD. У лістингу 3.10. ви можете бачити код цього класу.

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Storage;
using Microsoft.Xna.Framework.Content;
namespace P3_5
{
    public class gameObj2 : spriteComp
    {
        public gameObj2(Game game, ref Texture2D newTexture,
            Rectangle newRectangle, Vector2 newPosition)
```

```
    : base(game, ref newTexture, newRectangle, newPosition)
{
    // TODO: Construct any child components here
}
public override void Move()
{
    KeyboardState kbState = Keyboard.GetState();
    if (kbState.IsKeyDown(Keys.W))
    {
        sprPosition.Y -= 5;
    }
    if (kbState.IsKeyDown(Keys.S ))
    {
        sprPosition.Y += 5;
    }
    if (kbState.IsKeyDown(Keys.A ))
    {
        sprPosition.X -= 5;
    }
    if (kbState.IsKeyDown(Keys.D))
    {
        sprPosition.X += 5;
    }
    base.Move();
}
public override void Initialize()
{
    // TODO: Add your initialization code here
    base.Initialize();
}
public override void Update(GameTime gameTime)
{
    // TODO: Add your update code here
    base.Update(gameTime);
```

```
        }
        public override void Draw(GameTime gameTime)
        {
            base.Draw(gameTime);
        }
    }
}
```

Лістинг 3.10. Клас gameObj2 – спадкоємець spriteComp

Как видите, единственное отличие этого кода от кода компонента gameObj1 – это проверка на нажатие других клавиш.

Теперь перейдем к основному файлу игры – коду Game1. Для наглядности он приведен полностью (листинг 3.11.).

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;
namespace P3_5
{
    /// <summary>
    /// This is the main type for your game
    /// </summary>
    public class Game1 : Microsoft.Xna.Framework.Game
    {
        GraphicsDeviceManager graphics;
        SpriteBatch spriteBatch;
        gameObj1 gameObject1;
        gameObj2 gameObject2;
        Texture2D texture;
```

```
public Game1()
{
    graphics = new GraphicsDeviceManager(this);
    Content.RootDirectory = «Content»;
}
protected override void Initialize()
{
    // TODO: Add your initialization logic here
    base.Initialize();
}
protected override void LoadContent()
{
    // Create a new SpriteBatch, which can be used to draw textures.
    spriteBatch = new SpriteBatch(GraphicsDevice);
    Services.AddService(typeof(SpriteBatch), spriteBatch);
    texture = Content.Load<Texture2D>(«BallandBats»);
    CreateNewObject();
    // TODO: use this.Content to load your game content here
}
protected void CreateNewObject()
{
    gameObject1 = new gameObj1 (this, ref texture,
        new Rectangle(18, 9, 17, 88),
        new Vector2(100, 150));
    Components.Add(gameObject1);
    gameObject2 = new gameObj2(this, ref texture,
        new Rectangle(17, 106, 17,88),
        new Vector2(200, 150));
    Components.Add(gameObject2);
}
protected override void UnloadContent()
{
    // TODO: Unload any non ContentManager content here
}
```

```
        protected override void Update(GameTime gameTime)
        {
            // TODO: Add your update logic here
            base.Update(gameTime);
        }
        protected override void Draw(GameTime gameTime)
        {
            graphics.GraphicsDevice.Clear(Color.CornflowerBlue);
            // TODO: Add your drawing code here
            spriteBatch.Begin();
            base.Draw(gameTime);
            spriteBatch.End();
        }
    }
}
```

Лістинг 3.11. Код Game1

Для початку ми оголосили пару об'єктних змінних - однутипу gameObj1, другу - gameObj2. Далі, після завантаження ігрового контенту, ми виконали метод CreateNewObject (), який містить код для створення двох об'єктів. Після створення ми реєструємо кожний з них в списку ігрових компонентів.

Як бачите, в коді основної програми ми лише завантажуємо ресурси і створюємо ігрові об'єкти, після чого вся робота по управлінню ними ведеться за допомогою коду, що знаходиться в цих же об'єктах. На рис. 3.2. ви можете бачити ігрове вікно з двома виведеними в ньому об'єктами, які управляються роздільно.

ЦЕНТРАЛІЗОВАНЕ КЕРУВАННЯЕ ДЕКІЛЬКОМА ОБ'ЄКТАМИ.

Створимо новий проект (P3_6), аналогічний проекту P2_3 (лише відключимо висновок фону). Ігровий компонент spriteComp буде використаний для побудови двох об'єктів. Змінимо його код так, як показано в лістингу 3.12.

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
```

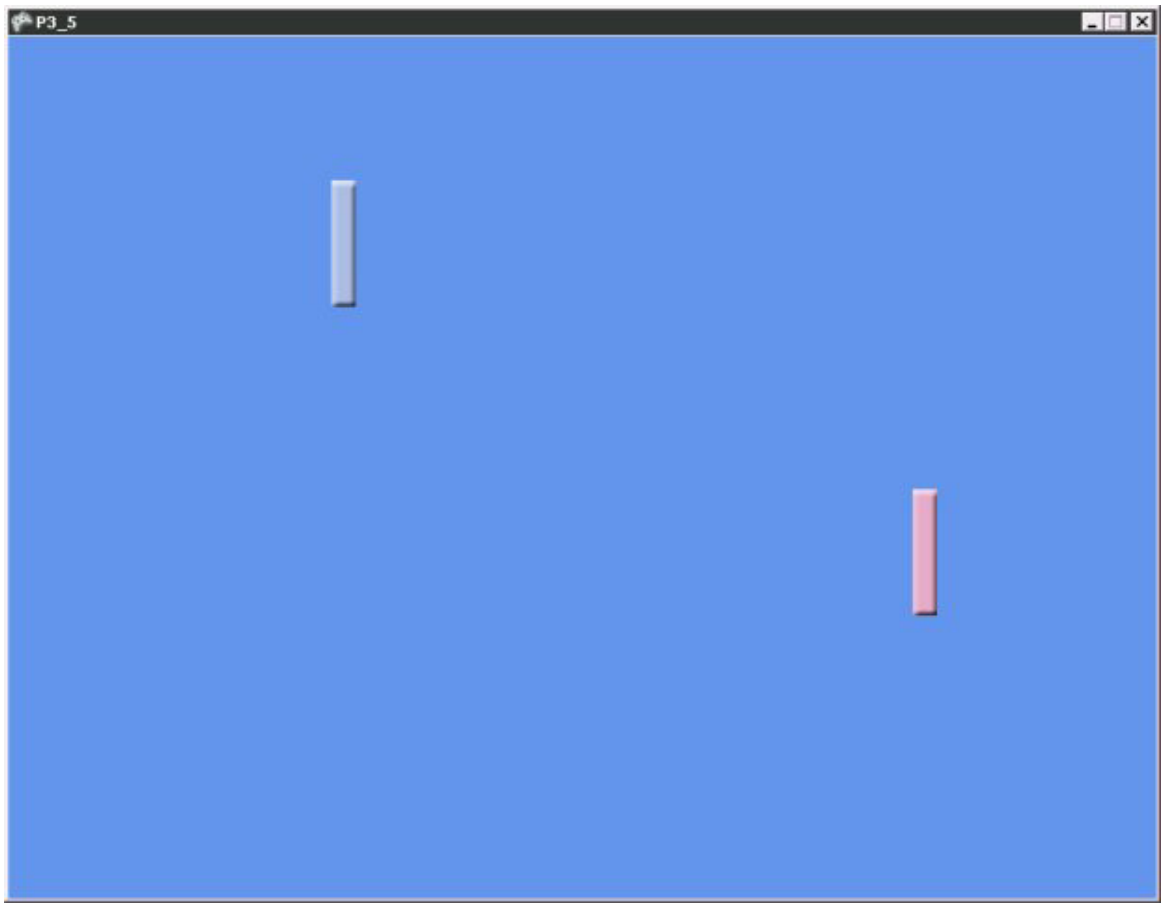

Рис. 3.2. Ігрове вікно з двома ігровими компонентами.

```
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Storage;
using Microsoft.Xna.Framework.Content;
namespace P3_6
{
   public class spriteComp : Microsoft.Xna.Framework.DrawableGameComponent
   {
      protected Texture2D sprTexture;
      public Rectangle sprRectangle;
      public Vector2 sprPosition;

      public spriteComp(Game game, ref Texture2D newTexture,
         Rectangle newRectangle, Vector2 newPosition)
         : base(game)
      {
         sprTexture = newTexture;
         sprRectangle = newRectangle;
```

```
            sprPosition = newPosition;

            // TODO: Construct any child components here
        }
        public override void Initialize()
        {
            // TODO: Add your initialization code here
            base.Initialize();
        }

        public override void Update(GameTime gameTime)
        {
            // TODO: Add your update code here
            base.Update(gameTime);
        }
        public override void Draw(GameTime gameTime)
        {
            SpriteBatch sprBatch =
                (SpriteBatch)Game.Services.GetService(typeof(SpriteBatch));
            sprBatch.Draw(sprTexture, sprPosition, sprRectangle, Color.White);
            base.Draw(gameTime);
        }
    }
}
```

Лістинг 3.12. Код класу spriteComp

Зверніть увагу на те, що клас містить лише кілька властивостей, власний конструктор і команди для виведення зображення на екран в методі Draw (). Властивості sprRectangle і sprPosition оголошені з модифікатором доступності Public - вони стануть в нагоді нам для управління поведінкою об'єкта з об'єкта класу Game1.

Ось як виглядає код класу Game1. Вся логіка по переміщенню об'єктів і по перевірці допустимості переміщення реалізована в ньому.

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
```

```
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;
namespace P3_6
{
   public class Game1 : Microsoft.Xna.Framework.Game
   {
      GraphicsDeviceManager graphics;
      SpriteBatch spriteBatch;
      spriteComp  gameObject1, gameObject2;
      Texture2D texture;
      Rectangle scrBounds;
      public Game1()
      {
         graphics = new GraphicsDeviceManager(this);
         Content.RootDirectory = «Content»;
      }
      protected override void Initialize()
      {
         // TODO: Add your initialization logic here
         base.Initialize();
      }
      protected override void LoadContent()
      {
         // Create a new SpriteBatch, which can be used to draw textures.
         spriteBatch = new SpriteBatch(GraphicsDevice);
         Services.AddService(typeof(SpriteBatch), spriteBatch);
         texture = Content.Load<Texture2D>(«BallandBats»);
         scrBounds = new Rectangle(0, 0,
            this.Window.ClientBounds.Width,
            this.Window.ClientBounds.Height);
         CreateNewObject();
```

```
    // TODO: use this.Content to load your game content here
}
protected void CreateNewObject()
{
    gameObject1 = new spriteComp (this, ref texture,
        new Rectangle(18, 9, 17, 88),
        new Vector2(100, 150));
    Components.Add(gameObject1);
    gameObject2 = new spriteComp (this, ref texture,
        new Rectangle(17, 106, 17, 88),
        new Vector2(200, 150));
    Components.Add(gameObject2);
}
void Test (spriteComp spr, Rectangle scr)
{
    if (spr.sprPosition.X < scr.Left)
    {
        spr.sprPosition.X = scr.Left;
    }
    if (spr.sprPosition.X > scr.Width - spr.sprRectangle.Width)
    {
        spr.sprPosition.X = scr.Width - spr.sprRectangle.Width;
    }
    if (spr.sprPosition.Y < scr.Top)
    {
        spr.sprPosition.Y = scr.Top;
    }
    if (spr.sprPosition.Y > scr.Height - spr.sprRectangle.Height)
    {
        spr.sprPosition.Y = scr.Height - spr.sprRectangle.Height;
    }
}
void MoveUp(spriteComp spr)
{
    spr.sprPosition.Y -= 5;
```

```
}
void MoveDown(spriteComp spr)
{
   spr.sprPosition.Y += 5;
}
void MoveLeft(spriteComp spr)
{
   spr.sprPosition.X -= 5;
}
void MoveRight(spriteComp spr)
{
   spr.sprPosition.X += 5;
}

protected override void UnloadContent()
{
   // TODO: Unload any non ContentManager content here
}
protected override void Update(GameTime gameTime)
{
   // TODO: Add your update logic here
   KeyboardState kbState = Keyboard.GetState();
   if (kbState.IsKeyDown(Keys.Up))
   {
      MoveUp(gameObject1);
   }
   if (kbState.IsKeyDown(Keys.Down))
   {
      MoveDown(gameObject1);
   }
   if (kbState.IsKeyDown(Keys.Left))
   {
      MoveLeft(gameObject1);
   }
```

```
            if (kbState.IsKeyDown(Keys.Right))
            {
                MoveRight(gameObject1);
            }
            Test(gameObject1, scrBounds);
            if (kbState.IsKeyDown(Keys.W))
            {
                MoveUp(gameObject2);
            }
            if (kbState.IsKeyDown(Keys.S))
            {
                MoveDown(gameObject2);
            }
            if (kbState.IsKeyDown(Keys.A))
            {
                MoveLeft(gameObject2);
            }
            if (kbState.IsKeyDown(Keys.D))
            {
                MoveRight(gameObject2);
            }
            Test(gameObject2, scrBounds);
            base.Update(gameTime);
        }
        protected override void Draw(GameTime gameTime)
        {
            graphics.GraphicsDevice.Clear(Color.CornflowerBlue);
            // TODO: Add your drawing code here
            spriteBatch.Begin();
            base.Draw(gameTime);
            spriteBatch.End();
        }
    }
}
```

Лістинг 3.13. Код классу Game1

Для переміщення об'єктів класу spriteComp ми модифікуємо їх властивості sprPosition з коду об'єкта Game1. Для зручності ми створили чотири процедури - Up, Down, Left, Right - кожна з них приймає на вході об'єкт типу spriteComp і модифікує відповідну координату, що зберігається у властивості sprPosition цього об'єкта. Ці процедури викликаються після аналізу стану клавіатури в методі Update (). Після цього викликається метод Test (), який приймає на вході об'єкт типу spriteComp і об'єкт типу Rectangle, який зберігає параметри екрану. У методі проводяться перевірки допустимості нових значень, при порушенні об'єктом кордонів, координати модифікуються.

Тепер розглянемо ще один спосіб організації управління і роботи з об'єктами.

Автоматичне переміщення об'єкта: кілька автономних об'єктів без створення об'єктних змінних.

Вище ми створювали схеми управління ігровими об'єктами, розрахованими на дії гравця. Але цілком можливо, що нам захочеться створити об'єкт, який буде знаходитися під управлінням комп'ютера, тобто - якогось алгоритму. Часто для таких об'єктів не виділяють окремих об'єктних змінних - їх можна динамічно створювати в потрібних кількостях і знищувати.

Розробимо наступну програму. На ігровий екран має виводитися випадкове кількість ігрових об'єктів (в діапазоні від 50 до 200), в випадкових позиціях в межах кордонів екрану. Кожен з них переміщається в випадковому напрямку на випадкове число кроків (в діапазоні від 50 до 200). Об'єкти не можуть перетинати межі екрану. При кожному новому кроці колір об'єкта повинен випадковим чином змінюватися

Створимо новий ігровий проект (P3_7), аналогічний P2_3. Модифікуємо його код. У лістингу 3.14. ви можете бачити код компонента spriteComp.

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Storage;
using Microsoft.Xna.Framework.Content;

namespace P3_7
{
   /// <summary>
   /// This is a game component that implements IUpdateable.
```

```
/// </summary>
public class spriteComp : Microsoft.Xna.Framework.DrawableGameComponent
{
    protected Texture2D sprTexture;
    protected Rectangle sprRectangle;
    protected Vector2 sprPosition;
    protected Rectangle scrBounds;
    //Для генерування випадкових чисел
    protected Random randNum;
    //Кількість кроків, яке пройде спрайт
    protected int stepNumber;
    //Координата, до якої він буде рухатися
    protected Vector2 newPosition;
    //Колір спрайта
    protected Color sprColor;
    //Швидкість переміщення
    protected Vector2 speed;
    public spriteComp(Game game, ref Texture2D newTexture,
        Rectangle newRectangle, int Speed)
        : base(game)
    {
        sprTexture = newTexture;
        sprRectangle = newRectangle;
        //Ініціалізуємо лічильник
        randNum = new Random(Speed);
        scrBounds = new Rectangle(0, 0,
            game.Window.ClientBounds.Width,
            game.Window.ClientBounds.Height);
        //Вставновлюємо стартову позицію спрайта
            sprPosition.X = (float)randNum.NextDouble() * (scrBounds.Width - sprRectangle.
Width);
          sprPosition.Y = (float)randNum.NextDouble() * (scrBounds.Height  - sprRectangle.
Height);
        //Кількість кроків равна нулю
        stepNumber = 0;
```

```
        //Задамо випадковий колір для придання зображенню
        //відповідного відтінку
            sprColor = new Color((byte)randNum.Next(0, 255), (byte)randNum.Next(0, 255),
(byte)randNum.Next(0, 255));
        //Змінна для схову швидкості поки порожня
        speed = new Vector2 ();
        // TODO: Construct any child components here
    }
    public override void Initialize()
    {
        // TODO: Add your initialization code here
        base.Initialize();
    }
    //Переміщення спрайта
    public virtual void Move()
    {
        //Якщо виконани не всі кроки
        if (stepNumber > 0)
        {
            //зменшимо змінну, що зберігає число кроків
            stepNumber--;
            //Модифікуємо координати в відповідності з положенням
            //об'єкта відносно бажаної координати
            //і в відповідності зі швидкістю
            if (sprPosition.X < newPosition.X) sprPosition.X += speed.X;
            if (sprPosition.X > newPosition.X) sprPosition.X -= speed.X;
            if (sprPosition.Y < newPosition.Y) sprPosition.Y += speed.Y;
            if (sprPosition.Y > newPosition.Y) sprPosition.Y -= speed.Y;
        }
        //Якщо попередній крок завершений, змінна зберігає 0
        if (stepNumber ==0)
        {
            //Встановимо випадкову кількість кроків
            stepNumber = randNum.Next(50, 200);
```

```
            //Сгенеруємо випадкову цільову позицію
        newPosition.X = (float)randNum.NextDouble() * (scrBounds.Width - sprRectangle.
Width);
        newPosition.Y = (float)randNum.NextDouble() * (scrBounds.Height - sprRectangle.
Height);
            //Встановимо нову швидкість
            speed.X = randNum.Next(1, 5);
            speed.Y = randNum.Next(1, 5);
            // Задамо новий колір спрайта
         sprColor = new Color((byte)randNum.Next(0, 255), (byte)randNum.Next(0, 255),
(byte)randNum.Next(0, 255));


        }
        //Виклик перевірки на допустимість переміщення
        Check();
    }
    //Перевірка допустимісті переміщення
    void Check()
    {
        if (sprPosition.X < scrBounds.Left)
        {
            sprPosition.X = scrBounds.Left;
        }
        if (sprPosition.X > scrBounds.Width - sprRectangle.Width)
        {
            sprPosition.X = scrBounds.Width - sprRectangle.Width;
        }
        if (sprPosition.Y < scrBounds.Top)
        {
            sprPosition.Y = scrBounds.Top;
        }
        if (sprPosition.Y > scrBounds.Height - sprRectangle.Height)
        {
            sprPosition.Y = scrBounds.Height - sprRectangle.Height;
        }
```

```
        }
        public override void Update(GameTime gameTime)
        {
            //Виклик методу для переміщення спрайту
            Move();
            base.Update(gameTime);
        }
        public override void Draw(GameTime gameTime)
        {
            SpriteBatch sprBatch =
                (SpriteBatch)Game.Services.GetService(typeof(SpriteBatch));
            sprBatch.Draw(sprTexture, sprPosition, sprRectangle, sprColor);
            base.Draw(gameTime);
        }
    }
}
```

Листинг 3.14. Код компоненту spriteComp

Текст супроводжують коментарями, які досить ясно висвітлюють його особливості.

У лістингу 3.15. наведено код класу Game1.

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;

namespace P3_7
{
    /// <summary>
    /// This is the main type for your game
```

```
/// </summary>
public class Game1 : Microsoft.Xna.Framework.Game
{
    GraphicsDeviceManager graphics;
    SpriteBatch spriteBatch;
    Texture2D texture;
    Random randNum;
    public Game1()
    {
        graphics = new GraphicsDeviceManager(this);
        Content.RootDirectory = «Content»;
    }
    protected override void Initialize()
    {
        // TODO: Add your initialization logic here
        base.Initialize();
    }
    protected override void LoadContent()
    {
        // Create a new SpriteBatch, which can be used to draw textures.
        spriteBatch = new SpriteBatch(GraphicsDevice);
        Services.AddService(typeof(SpriteBatch), spriteBatch);
        texture = Content.Load<Texture2D>(«BallandBats»);
        randNum = new Random();
        CreateNewObject();
        // TODO: use this.Content to load your game content here
    }
    protected void CreateNewObject()
    {
        //Цикл от 1 до случайного числ в диапазоне 50,200
        for (int i = 0; i < randNum .Next (50,200); i++)
        {
            //Додаємо до переліку компонентів новий компонент класу spriteComp
            Components.Add(new spriteComp(this, ref texture,
```

```
            new Rectangle(16, 203, 17, 17), i));
        }
    }
    protected override void UnloadContent()
    {
    }
    protected override void Update(GameTime gameTime)
    {
        if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
            this.Exit();
        base.Update(gameTime);
    }
    protected override void Draw(GameTime gameTime)
    {
        graphics.GraphicsDevice.Clear(Color.CornflowerBlue);
        // TODO: Add your drawing code here
        spriteBatch.Begin();
        base.Draw(gameTime);
        spriteBatch.End();
    }
  }
}
```

Лістинг 3.15. Код класу Game1

В цілому код цього класу вже добре знайомий вам по попередніх прикладів. Тому зверніть особливу увагу на процедуру CreateNewObject (). Вона містить циклічний виклик методу Add об'єкта Components, який додає до списку ігрових об'єктів новий об'єкт типу spriteComp. Номер ітерації передається в об'єкт для ініціалізації генератора випадкових чисел. Якщо примусово НЕ ініціалізувати генератори випадкових чисел створених об'єктів різними значеннями - це призведе до неправильної роботи програми - всі об'єкти будуть виведені в одній і тій же позиції (якщо об'єктів багато - то в декількох позиціях).

Такий підхід дозволяє створювати потрібну кількість об'єктів, не піклуючись про створення об'єктних змінних. На одному з наступних занять ми розглянемо методи роботи з такими об'єктами. На рис. 3.3. ви можете бачити ігрове вікно проекту.

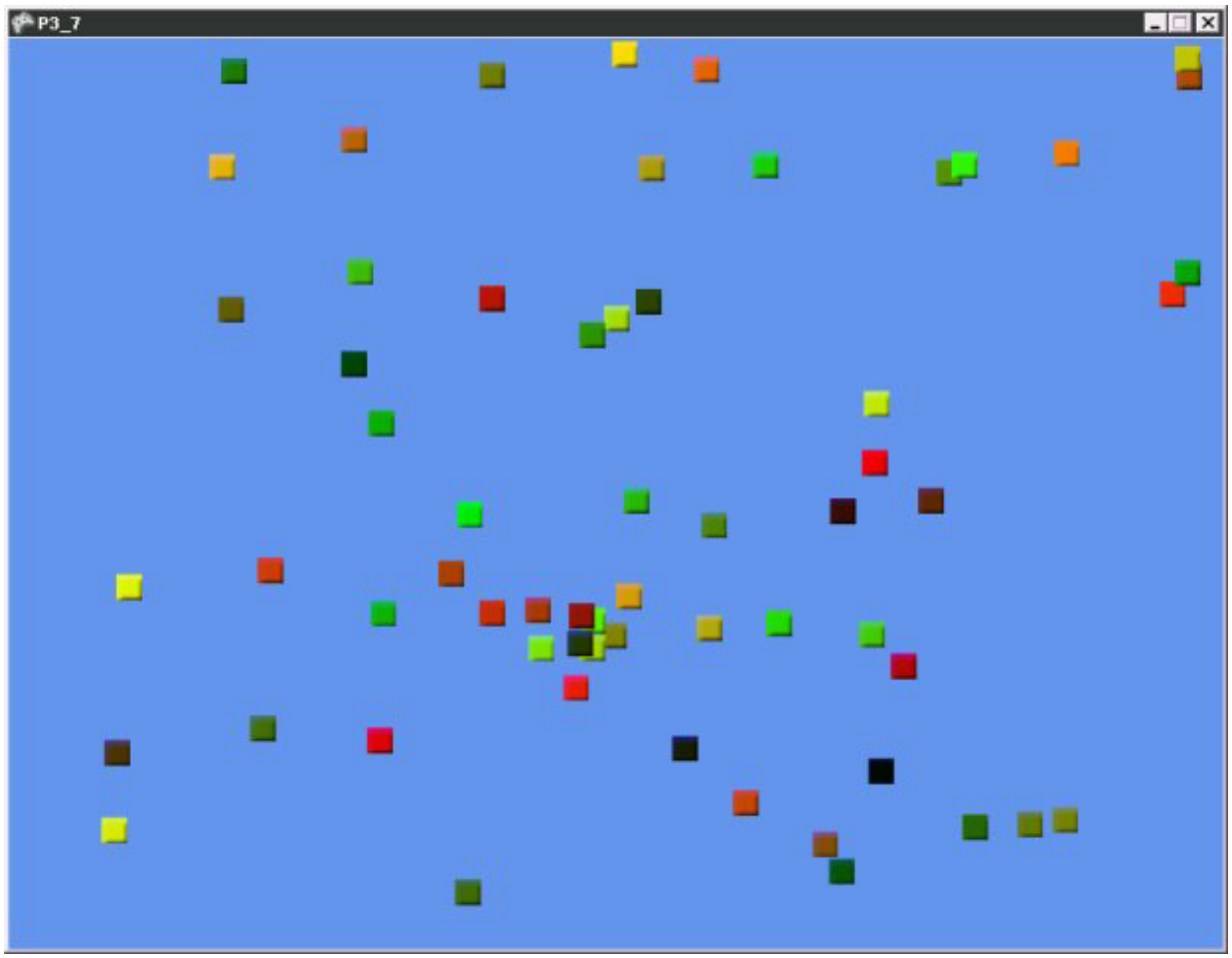

Рис. 3.3. Ігрове вікно, яке виводить автоматично створені об'єкти

РОБОТА З ІГРОВИМ МАНІПУЛЯТОРОМ.

Для роботи з ігровим маніпулятором служить об'єкт GamePad. Зберегти стан об'єкта можна в змінної типу GamePadState. Ця змінна зберігає інформацію про елементи управління, які розташовані на пристрої. Треба відзначити, що ігровий маніпулятор підтримує вібрацію - розробляючи гру, розрахована на управління маніпулятором, включаючи в потрібний момент вібрацію можна зробити її цікавішою.

ЛАБОРАТОРНА РОБОТА 4.ВЗАІМОДІЯ ОБ'ЄКТІВ

Анотація: Обробка взаємодії об'єктів - це дуже важлива частина створення гри. У цій лабораторній роботі ми розглянемо обробку зіткнень об'єктів.

Ключові слова: об'єкт, координати, алгоритм, централізоване управління, функція, true, спрайт, пошук, змінна, значення, прямокутник, висота, покажчик, відстань, радіус, Окружність, відображення

Задачі роботи:

- Вивчити теоретичні основи обробки зіткнень об'єктів в двовимірному просторі
- Вивчити алгоритми обробки зіткнень об'єктів в двовимірному просторі
- Розглянути алгоритм перевірки приналежності точки фігурі в двовимірному просторі
- Створити просту гру - ігровий об'єкт автоматично переміщається по екрану, відскакуючи від його кордонів. Гравець повинен за допомогою миші потрапити по об'єкту. Кожне попадання має привести до висновку інформації про кількість влучень в заголовок ігрового вікна.
- Створити просту гру «Стрілянина по мішені» з реалізацією алгоритму перевірки приналежності точки окружності

ОБРОБКА ЗІТКНЕНЬ.

Вище ми вже мали справу з простим прикладом обробки зіткнень об'єктів. Були створені правила, відповідно до яких ігровий об'єкт не міг перетнути межі екрану. Часто для обробки зіткнень двовимірних об'єктів обробляють зіткнення прямокутників, що описують ці об'єкти. Для цього потрібно знати координати прямокутника, в нашому випадку координати задаються координатою лівої верхньої вершини, шириною і висотою фігури.

На рис. 4.1. ви можете бачити приклад об'єктів, що не стикаються один з одним.

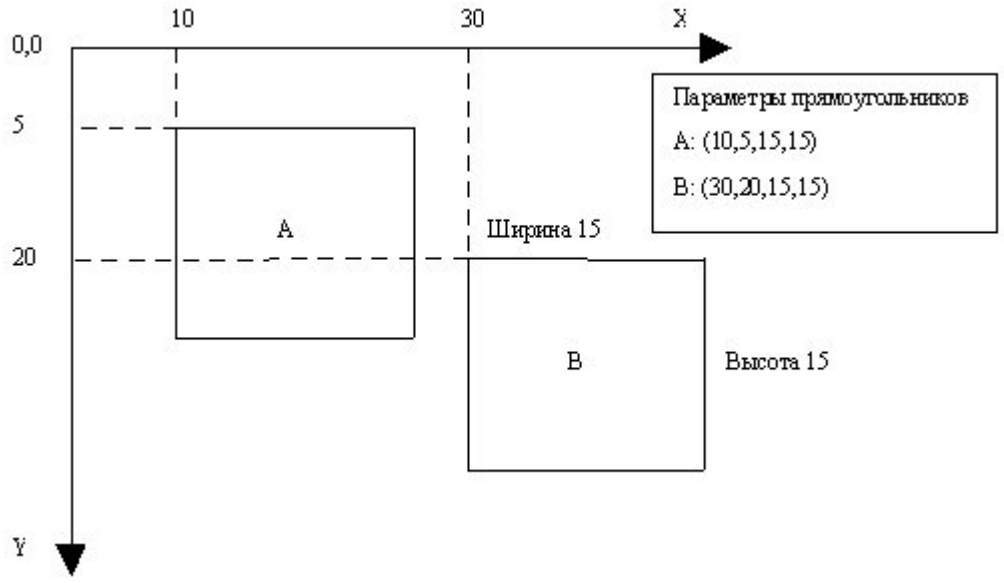

Рис. 4.1. Непересічні прямокутники.

На рис. 4.2. ви можете бачити пересічні прямокутники.

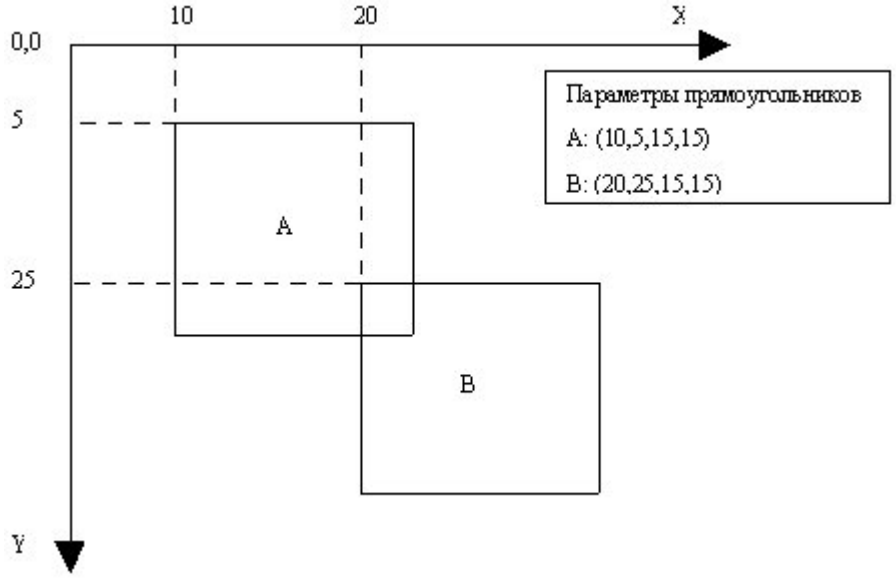

Рис. 4.2. Пересічні прямокутники.

Алгоритмічно умова перетину прямокутників можна записати у вигляді такої умови (лістинг 4.1.)

Якщо (A.X+A.Ширина > B.X И

A.X < B.X+B.Ширина И

A.Y+A.Висота>B.Висота И

A.Y<B.Y+B.Висота)

Тоді

Є зіткнення

Інакше

Немає зіткнення

Лістинг 4.1. Первірка перетину прямокутників

Створимо приклад, який ілюструє цей алгоритм. Візьмемо за основу проект P3_6. Назвемо новий проект P4_1. Нагадаємо, що в проекті P3_6 ми розглядали централізоване управління об'єктами з основного ігрового об'єкта. Продовжимо цей приклад, допрацювавши основний ігровий об'єкт таким чином, щоб при переміщенні об'єктів проводилася перевірка на їх зіткнення.

У лістингу 4.2. наведено код об'єкта Game1, де ми і реалізували таку перевірку. Так як об'єкти переміщаються шляхом модифікації їх позиції на ігровому екрані, реалізуємо перевірку на зіткнення при спробі переміщення об'єкта. Якщо об'єкти стикаються - ми таким чином модифікуємо їх координати, щоб вони не перекривалися.

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
```

```
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;
namespace P4_1
{
   /// <summary>
   /// This is the main type for your game
   /// </summary>
   public class Game1 : Microsoft.Xna.Framework.Game
   {
      GraphicsDeviceManager graphics;
      SpriteBatch spriteBatch;
      spriteComp gameObject1, gameObject2;
      Texture2D texture;
      Rectangle scrBounds;
      float sprSpeed;
      public Game1()
      {
         graphics = new GraphicsDeviceManager(this);
         Content.RootDirectory = «Content»;
      }
      protected override void Initialize()
      {
         // TODO: Add your initialization logic here
         base.Initialize();
      }
      protected override void LoadContent()
      {
         // Create a new SpriteBatch, which can be used to draw textures.
         spriteBatch = new SpriteBatch(GraphicsDevice);
```

```
    Services.AddService(typeof(SpriteBatch), spriteBatch);
    texture = Content.Load<Texture2D>(«BallandBats»);
    scrBounds = new Rectangle(0, 0,
        this.Window.ClientBounds.Width,
        this.Window.ClientBounds.Height);
    sprSpeed = 5;
    CreateNewObject();
    // TODO: use this.Content to load your game content here
}
protected void CreateNewObject()
{
    gameObject1 = new spriteComp(this, ref texture,
        new Rectangle(18, 9, 17, 88),
        new Vector2(100, 150));
    Components.Add(gameObject1);
    gameObject2 = new spriteComp(this, ref texture,
        new Rectangle(17, 106, 17, 88),
        new Vector2(200, 150));
    Components.Add(gameObject2);
}
void Test(spriteComp spr, Rectangle scr)
{
    if (spr.sprPosition.X < scr.Left)
    {
        spr.sprPosition.X = scr.Left;
    }
    if (spr.sprPosition.X > scr.Width - spr.sprRectangle.Width)
    {
        spr.sprPosition.X = scr.Width - spr.sprRectangle.Width;
    }
    if (spr.sprPosition.Y < scr.Top)
    {
        spr.sprPosition.Y = scr.Top;
    }
```

```
    if (spr.sprPosition.Y > scr.Height - spr.sprRectangle.Height)
    {
        spr.sprPosition.Y = scr.Height - spr.sprRectangle.Height;
    }
}
void MoveUp(spriteComp spr, float speed)
{
    spr.sprPosition.Y -= speed ;
}
void MoveDown(spriteComp spr,float speed)
{
    spr.sprPosition.Y += speed;
}
void MoveLeft(spriteComp spr, float speed)
{
    spr.sprPosition.X -= speed ;
}
void MoveRight(spriteComp spr, float speed)
{
    spr.sprPosition.X += speed ;
}
bool  IsCollide(spriteComp sp1, spriteComp sp2)
{
    if (sp1.sprPosition.X < sp2.sprPosition.X + sp2.sprRectangle.Width &&
        sp1.sprPosition.X + sp1.sprRectangle.Width > sp2.sprPosition.X &&
        sp1.sprPosition.Y < sp2.sprPosition.Y + sp2.sprRectangle.Height &&
        sp1.sprPosition.Y + sp1.sprRectangle.Height > sp2.sprPosition.Y)
    {
        return true;
    }
    else return false;
}
protected override void UnloadContent()
{
```

```
        // TODO: Unload any non ContentManager content here
    }
    protected override void Update(GameTime gameTime)
    {
        // TODO: Add your update logic here
        KeyboardState kbState = Keyboard.GetState();
        if (kbState.IsKeyDown(Keys.Up))
        {
            MoveUp(gameObject1, sprSpeed );
            while (IsCollide(gameObject1, gameObject2))
            {
                MoveDown(gameObject1, (sprSpeed / 10));
            }
        }
        if (kbState.IsKeyDown(Keys.Down))
        {
            MoveDown(gameObject1, sprSpeed );
            while (IsCollide(gameObject1, gameObject2))
            {
                MoveUp(gameObject1, (sprSpeed / 10));
            }
        }
        if (kbState.IsKeyDown(Keys.Left))
        {
            MoveLeft(gameObject1, sprSpeed );
            while (IsCollide(gameObject1, gameObject2))
            {
                MoveRight(gameObject1, (sprSpeed / 10));
            }
        }
        if (kbState.IsKeyDown(Keys.Right))
        {
            MoveRight(gameObject1,sprSpeed );
            while (IsCollide(gameObject1, gameObject2))
```

```
    {
        MoveLeft(gameObject1, (sprSpeed / 10));
    }
}
Test(gameObject1, scrBounds);
if (kbState.IsKeyDown(Keys.W))
{
    MoveUp(gameObject2,sprSpeed );
    while (IsCollide(gameObject1, gameObject2))
    {
        MoveDown(gameObject2, (sprSpeed / 10));
    }
}
if (kbState.IsKeyDown(Keys.S))
{
    MoveDown(gameObject2,sprSpeed );
    while (IsCollide(gameObject1, gameObject2))
    {
        MoveUp(gameObject2, (sprSpeed / 10));
    }
}
if (kbState.IsKeyDown(Keys.A))
{
    MoveLeft(gameObject2,sprSpeed );
    while (IsCollide(gameObject1, gameObject2))
    {
        MoveRight(gameObject2, (sprSpeed / 10));
    }
}
if (kbState.IsKeyDown(Keys.D))
{
    MoveRight(gameObject2,sprSpeed );
    while (IsCollide(gameObject1, gameObject2))
    {
```

```
                MoveLeft(gameObject2, (sprSpeed / 10));
            }
        }
        Test(gameObject2, scrBounds);

        base.Update(gameTime);
    }
    /// <summary>
    /// This is called when the game should draw itself.
    /// </summary>
    /// <param name=»gameTime»>Provides a snapshot of timing values.</param>
    protected override void Draw(GameTime gameTime)
    {
        graphics.GraphicsDevice.Clear(Color.CornflowerBlue);
        // TODO: Add your drawing code here
        spriteBatch.Begin();
        base.Draw(gameTime);
        spriteBatch.End();
    }
  }
}
```

Лістинг 4.2. Код об'єкту Game1, який реалізує перевірку зіткнень об'єктів

В розділ оголошення змінних класу ми додали нову змінну sprSpeed типу float, яку будемо використовувати при модифікації позиції спрайту в процедурах зміни координат в залежності від натиснутої клавіші на клавіатурі.

Зверніть увагу на логічну функцію IsCollide, її код приведений в лістингу.

```
bool  IsCollide(spriteComp sp1, spriteComp sp2)
 {
    if (sp1.sprPosition.X < sp2.sprPosition.X + sp2.sprRectangle.Width &&
        sp1.sprPosition.X + sp1.sprRectangle.Width > sp2.sprPosition.X &&
        sp1.sprPosition.Y < sp2.sprPosition.Y + sp2.sprRectangle.Height &&
        sp1.sprPosition.Y + sp1.sprRectangle.Height > sp2.sprPosition.Y)
    {
        return true;
```

```
    }
    else return false;
}
```

Лістинг 4.3. Логічна функція IsCollide.

Ця функція приймає в якості параметрів два об'єкти класу spriteComp - тобто два наших об'єкта, переміщенням яких ми управляємо з коду ігрового класу. Якщо виконується умова зіткнення об'єктів - функція повертає true, інакше - false.

Ми викликаємо цю функцію при переміщенні об'єктів, коригуючи їх позиції при зіткненні. Розглянемо приклад переміщення одного з об'єктів в напрямку «Вгору» - лістинг 4.4.

```
if (kbState.IsKeyDown(Keys.Up))
    {
      MoveUp(gameObject1, sprSpeed );
      while (IsCollide(gameObject1, gameObject2))
      {
   MoveDown(gameObject1, (sprSpeed / 10));
      }
    }
```

Лістинг 4.4. Фрагмент коду, який використовує функцію IsCollide()

У цьому фрагменті ми переміщаємо об'єкт на кількість пікселів, що задаються параметром sprSpeed в напрямку «Вгору». Після переміщення ми викликаємо функцію IsCollide (). Якщо вона повернула false - ніяких дій більше не робимо. Але якщо вона повернула true - переміщаємо об'єкт в зворотному напрямку (викликаючи процедуру MoveDown), змінюючи координати зі швидкістю, що дорівнює 1/10 звичайного зміни координат. У прикладі швидкість зміни позиції спрайту дорівнює 5. Якщо спрайт на цій швидкості зайшов за кордону іншого спрайту - дана ситуація обробляється як зіткнення. Після цього спрайт переміщається в зворотному напрямку зі швидкістю 0,5 до тих пір, поки не виявиться, що зіткнення більше немає. Таким чином при зіткненні об'єкти зупиняються строго близько кордонів один одного, не перетинаючи їх, але і не перебуваючи на деякій відстані один від одного при швидкості переміщення, яка менше, ніж розміри об'єктів.

На рис. 4.3. ви можете бачити приклад зіткнення об'єктів.

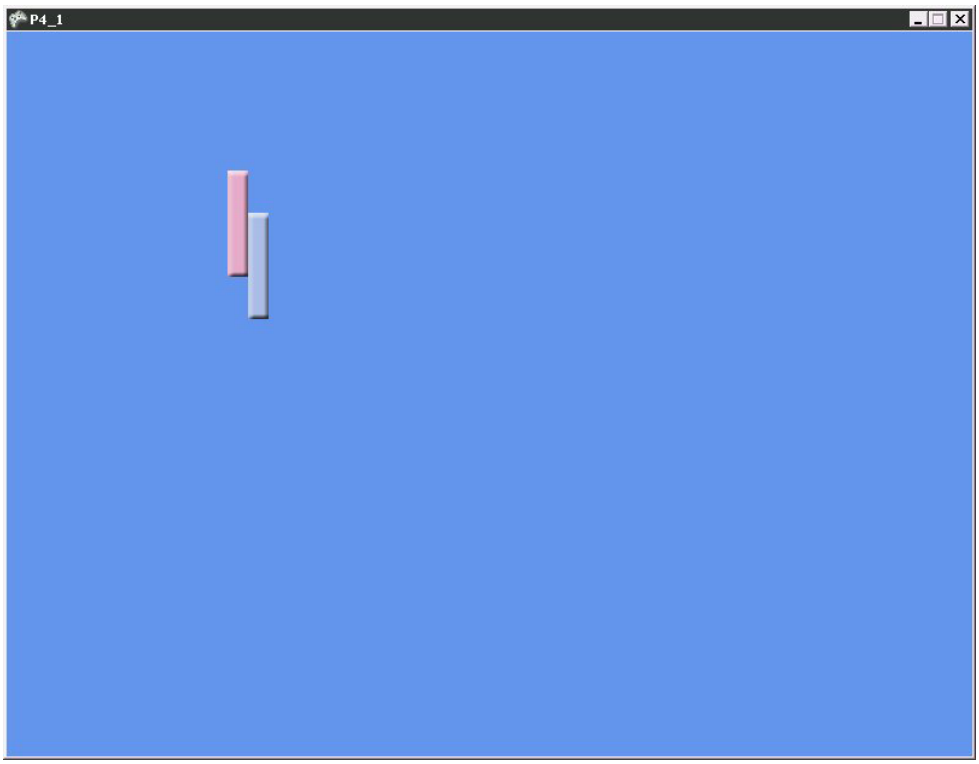

Рис. 4.3. Зіткнення об'єктів

Тепер розглянемо приклад обробки зіткнень об'єктів при їх автоматичному переміщенні із завданням швидкості переміщення об'єктів.

ОБРОБКА ЗІТКНЕНЬ АВТОМАТИЧНО ПЕРЕМІЩУВАНИХ ОБ'ЄКТІВ

Створимо новий ігровий проект P4_2, взявши за основу проект P3_7. Будемо автоматично переміщати об'єкти з деякою випадково заданою швидкістю. При зіткненні об'єкта з межею екрану змінимо швидкість по X і по Y на протилежну. Так само зробимо при зіткненні об'єктів один з одним.

У лістингу 4.5. ви можете бачити приклад коду класу Game1.

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;
namespace P4_2
{
```

```
/// <summary>
/// This is the main type for your game
/// </summary>
public class Game1 : Microsoft.Xna.Framework.Game
{
    GraphicsDeviceManager graphics;
    SpriteBatch spriteBatch;
    Texture2D texture;
    Random randNum;
    public Game1()
    {
        graphics = new GraphicsDeviceManager(this);
        Content.RootDirectory = «Content»;
    }

    protected override void Initialize()
    {
        // TODO: Add your initialization logic here
        base.Initialize();
    }
    protected override void LoadContent()
    {
        // Create a new SpriteBatch, which can be used to draw textures.
        spriteBatch = new SpriteBatch(GraphicsDevice);
        Services.AddService(typeof(SpriteBatch), spriteBatch);
        texture = Content.Load<Texture2D>(«BallandBats»);
        randNum = new Random();
        CreateNewObject();
        // TODO: use this.Content to load your game content here
    }
    protected void CreateNewObject()
    {
        //Цикл от 1 до случайного числа в диапазоне 20,200
        for (int i = 0; i < randNum.Next(20, 200); i++)
```

```
        {
            //Добавляем в список компонентов новый компонент класса spriteComp
            Components.Add(new spriteComp(this, ref texture,
            new Rectangle(16, 203, 17, 17), i));
        }
    }
    protected override void UnloadContent()
    {
        // TODO: Unload any non ContentManager content here
    }
    protected override void Update(GameTime gameTime)
    {
        // Allows the game to exit
        if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
            this.Exit();
        // TODO: Add your update logic here
        base.Update(gameTime);
    }
    protected override void Draw(GameTime gameTime)
    {
        graphics.GraphicsDevice.Clear(Color.CornflowerBlue);
        // TODO: Add your drawing code here
        spriteBatch.Begin();
        base.Draw(gameTime);
        spriteBatch.End();
    }
  }
}
```

Лістинг 4.5. Код класу Game1

Можна помітити, що в основному ігровому класі ми лише створюємо випадкове кількість нових об'єктів класу spriteComp. Управління об'єктами і обробка зіткнень ведеться безпосередньо в коді самих об'єктів. У лістингу 4.6. ви можете бачити код об'єкта spriteComp.

```
using System;
using System.Collections.Generic;
```

```
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Storage;
using Microsoft.Xna.Framework.Content;
namespace P4_2
{
    /// <summary>
    /// This is a game component that implements IUpdateable.
    /// </summary>
    public class spriteComp : Microsoft.Xna.Framework.DrawableGameComponent
    {
        protected Texture2D sprTexture;
        protected Rectangle sprRectangle;
        protected Vector2 sprPosition;
        protected Rectangle scrBounds;
        //Для генерування випадкових чисел
        protected Random randNum;
        //Об'єкт для доступу до основного ігрового об'єкту
        protected Game myGame;
        //Колір спрайту
        protected Color sprColor;
        //Швидкість переміщення
        public Vector2 speed;
        public spriteComp(Game game, ref Texture2D newTexture,
            Rectangle newRectangle, int Seed)
            : base(game)
        {
            sprTexture = newTexture;
            sprRectangle = newRectangle;
            //Ініціалізуємо лічильник
            randNum = new Random(Seed);
```

```
        myGame = game;
        scrBounds = new Rectangle(0, 0,
           game.Window.ClientBounds.Width,
           game.Window.ClientBounds.Height);
        //Встановлюємо стартову позицію спрайту
        setSpriteToStart();
        //Якщо спрайт зтикається з яким-небудь спрайтом - змінимо його позицію
        while (howManyCollides() > 0)
        {
            setSpriteToStart();
        }
        //Задамо випадковий колір для додання зображенню
        //відповідного відтінку
          sprColor = new Color((byte)randNum.Next(0, 255), (byte)randNum.Next(0, 255),
(byte)randNum.Next(0, 255));
        //Змінна для зберігання швидкості поки порожня
        speed = new Vector2((float)randNum .Next (-5,5),(float )randNum .Next (-5,5));
        // TODO: Construct any child components here
    }
   //Перевірка, чи не встановлені спрайти в позиції з перекриттям інших спрайтів
    int howManyCollides()
    {
        int howMany = 0;
        foreach (spriteComp spr in myGame.Components)
        {
            if (this != spr)
            {
                if (this.sprCollide(spr))
                {
                    howMany++;
                }
            }
        }
        return howMany;
```

```
    }
    //Встановка спрайту у випадкову стартову позицію
    void setSpriteToStart()
    {
        sprPosition.X = (float)randNum.NextDouble() * (scrBounds.Width - sprRectangle.
Width);
        sprPosition.Y = (float)randNum.NextDouble() * (scrBounds.Height - sprRectangle.
Height);
    }
    public override void Initialize()
    {
      // TODO: Add your initialization code here
      base.Initialize();
    }
    //Переміщення спрайту
    public virtual void Move()
    {
      sprPosition += speed;
    }
    //Перевірка допустимості переміщення
    void Check()
    {
      if (sprPosition.X < scrBounds.Left)
      {
        sprPosition.X = scrBounds.Left;
        speed.X *= -1;
      }
      if (sprPosition.X > scrBounds.Width - sprRectangle.Width)
      {
        sprPosition.X = scrBounds.Width - sprRectangle.Width;
        speed.X *= -1;
      }
      if (sprPosition.Y < scrBounds.Top)
      {
        sprPosition.Y = scrBounds.Top;
```

```
            speed.Y *= -1;
        }
        if (sprPosition.Y > scrBounds.Height - sprRectangle.Height)
        {
            sprPosition.Y = scrBounds.Height - sprRectangle.Height;
            speed.Y *= -1;
        }
    }
    public bool sprCollide(spriteComp spr)
    {
        return (this.sprPosition.X + this.sprRectangle.Width > spr.sprPosition.X &&
                this.sprPosition.X < spr.sprPosition.X + spr.sprRectangle.Width  &&
                this.sprPosition.Y + this.sprRectangle.Height > spr.sprPosition.Y &&
                this.sprPosition.Y < spr.sprPosition.Y + spr.sprRectangle .Height);
    }
    /// <summary>
    /// Allows the game component to update itself.
    /// </summary>
    /// <param name=»gameTime»>Provides a snapshot of timing values.</param>
    public override void Update(GameTime gameTime)
    {
        // TODO: Add your update code here
        //Виклик методу для переміщення спрайту
        Move();
        //Перевірка на зіткнення з кордонами екрану
        Check();
        //Виклик перевірки на зіткнення з іншими спрайтами
        IsSpriteCollide();
        //Можливо, при корекції спрайту віідносно іншого спрайту,
        //відбулося перекриття з іншим спрайтом або спрайтами
        //це призводить до «зависання» спрайтів - вони залишаються на одному
        //місці в «зчепленому» стані. Для того, щоб цього уникнути,
        //ми коригуємо позиції спрайтів до тих пір, поки кожен з них
        //гарантовано не опиниться поза інших спрайтів
```

```
        while (howManyCollides() > 0)
        {
            IsSpriteCollide();
        }
        base.Update(gameTime);
    }
    //Якщо спрайт перекрив інший спрайт - змінити його швидкість і
    //застосувати зміни до позиції спрайту
    void IsSpriteCollide()
    {
        foreach (spriteComp spr in myGame.Components)
        {
            if (spr != this)
            {
                if (this.sprCollide(spr))
                {
                    this.speed *= -1;
                    this.sprPosition += this.speed;
                }
            }
        }
    }
    public override void Draw(GameTime gameTime)
    {
        SpriteBatch sprBatch =
            (SpriteBatch)Game.Services.GetService(typeof(SpriteBatch));
        sprBatch.Draw(sprTexture, sprPosition, sprRectangle, sprColor);
        base.Draw(gameTime);
    }
  }
}
```

Лістинг 4.6. Код об'єкту spriteComp

При створенні екземпляра класу ми встановлюємо кожен спрайт в випадкову позицію на екрані. При цьому можливе перекриття спрайтів, тому здійснюється пошук

допустимої позиції (процедура howManyCollides ()) для кожного спрайту до тих пір, поки не буде з'ясовано, що поточна позиція не перекриває інші позиції.

Мінлива speed зберігає швидкість переміщення по X і по Y. У процедурі Move () ми додаємо значення цієї змінної до координати спрайту. Після цього проводимо дві перевірки. Перша - на зіткнення з межею екрану - за допомогою процедури Check (). Якщо об'єкт стосується однієї зі сторін кордону екрану, позиція об'єкта фіксується на кордоні і відповідна складова швидкості об'єкта змінюється на протилежну. Далі слід перевірка на зіткнення з іншими спрайтами - за допомогою процедури IsSpriteCollide (). Тут проводиться обхід всіх спрайтів з перевіркою на зіткнення. Якщо зіткнення сталося - швидкість спрайту по обидва складовим змінюється на протилежну, зміни тут же застосовуються, далі відбувається перевірка на зіткнення з усіма спрайтами (за допомогою вже згаданої процедури howManyCollides), коригування положення спрайту проводиться до тих пір, поки не буде з'ясовано, що він не перекриває жодного з інших спрайтів. На рис. 4.4. ви можете бачити ігровий екран проекту P4_2.

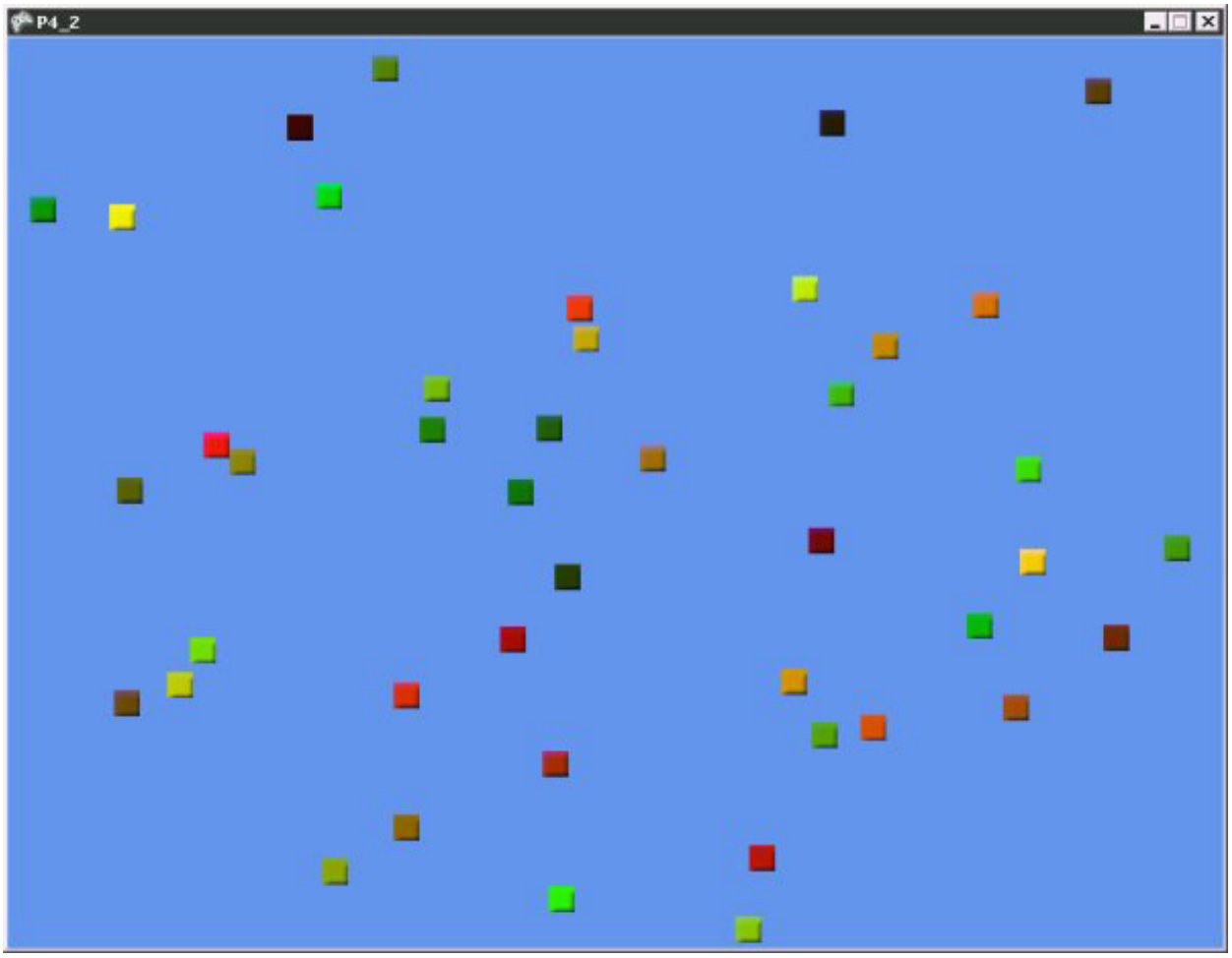

Рис. 4.4. Ігровий екран проекту P4_2.

Розглянемо ще один приклад обробки зіткнень.

ОБРОБКА ПОПАДАННЯ ТОЧКИ В МЕЖІ ОБ'ЄКТА.

Створимо просту гру на базі проекту P4_2. Гравець повинен за мінімальний час

потрапити по кожному з 100 знаходяться на екрані об'єктів, мишею. Об'єкт, за яким потрапили мишею, знищується.

Тут нам знадобиться алгоритм перевірки попадання точки в межі спрайту. Як і раніше, спрайт представлені прямокутниками. Для перевірки попадання точки в прямокутник, про який відома координата його лівої верхньої точки, ширина і висота, скористаємося таким алгоритмом (лістинг 4.7.). А - це прямокутник, В - це точка.

Якщо (А.Х+А.Ширина > В.Х И

А.Х < В.Х И

А.Y+А.Висота>В.Y И

А.Y<В.Y)

Тоді

Є зіткнення

Інакше

Немає зіткнення

Лістинг 4.7. Алгоритм перевірки попадання точки в прямокутник.

Створимо новий проект P4_3 на базі проекту P4_2. У лістингу 4.8. ви можете бачити код об'єкта Game1.

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;
namespace P4_3
{
  public class Game1 : Microsoft.Xna.Framework.Game
  {
    GraphicsDeviceManager graphics;
    SpriteBatch spriteBatch;
    Texture2D texture;
    public int Score;
```

```
bool IsWin = false;

public Game1()
{
   graphics = new GraphicsDeviceManager(this);
   Content.RootDirectory = «Content»;
}
protected override void Initialize()
{
   // TODO: Add your initialization logic here
   base.Initialize();
   this.IsMouseVisible = true;
   Score = 0;
}
protected override void LoadContent()
{
   // Create a new SpriteBatch, which can be used to draw textures.
   spriteBatch = new SpriteBatch(GraphicsDevice);
   Services.AddService(typeof(SpriteBatch), spriteBatch);
   texture = Content.Load<Texture2D>(«BallandBats»);
   CreateNewObject();
   // TODO: use this.Content to load your game content here
}
protected void CreateNewObject()
{

   for (int i = 0; i < 10; i++)
   {
      //Додаємо в список компонентів новий компонент класу spriteComp
      Components.Add(new spriteComp(this, ref texture,
      new Rectangle(16, 203, 17, 17), i, this));
   }
}
protected override void UnloadContent()
```

```
        {
            // TODO: Unload any non ContentManager content here
        }
        protected override void Update(GameTime gameTime)
        {
            // Allows the game to exit
            if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
                this.Exit();
           //Якщо мета гри не досягнута - виводимо в заголовок вікна
             //інформацію про поточний стан гри
            if (IsWin == false)
             Window.Title = «Знищено « + Score.ToString() + « за « + gameTime.TotalRealTime.
Seconds + « с.»;
           //Якщо мета досягнута - виводимо повідомлення про це
            if (Score == 10&&IsWin ==false )
            {
                Window.Title = «Ви знищили всіх за « + gameTime.TotalRealTime.Seconds+» с.»;
                IsWin = true;
            }
            // TODO: Add your update logic here

                base.Update(gameTime);
        }
        /// <summary>
        /// This is called when the game should draw itself.
        /// </summary>
        /// <param name=»gameTime»>Provides a snapshot of timing values.</param>
        protected override void Draw(GameTime gameTime)
        {
            graphics.GraphicsDevice.Clear(Color.CornflowerBlue);
            // TODO: Add your drawing code here
            spriteBatch.Begin();
            base.Draw(gameTime);
            spriteBatch.End();
```

```
        }
    }
}
```

Лістинг 4.8. Код об'єкту Game1

Тут ми створюємо 10 об'єктів класу spriteComp і перевіряємо, чи не досягнута мета гри - знищення всіх об'єктів. Для контролю кількості знищених об'єктів використовуємо змінну Score, для контролю за досягненням мети - змінну IsWin. Основна робота ведеться в об'єктах класу spriteComp. У лістингу 4. 8. ви можете знайти відповідний їм код.

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Storage;
using Microsoft.Xna.Framework.Content;
namespace P4_3
{
    public class spriteComp : Microsoft.Xna.Framework.DrawableGameComponent
    {
        protected Texture2D sprTexture;
        protected Rectangle sprRectangle;
        protected Vector2 sprPosition;
        protected Rectangle scrBounds;
        //Для генерування випадкових чисел
        protected Random randNum;
        // Об'єкт для доступу до основного ігрового об'єкту
        protected Game1 myGame;
        //Колір спрайту
        protected Color sprColor;
        //Швидкість переміщення
        public Vector2 speed;
        //Стан миші
```

```
MouseState mouse;
public spriteComp(Game game, ref Texture2D newTexture,
   Rectangle newRectangle, int Seed, Game1 game1)
   : base(game)
{
   sprTexture = newTexture;
   sprRectangle = newRectangle;
   //Ініціалізуємо лічильник
   randNum = new Random(Seed);
   myGame = game1;
   scrBounds = new Rectangle(0, 0,
      game.Window.ClientBounds.Width,
      game.Window.ClientBounds.Height);
   //Встановлюємо стартову позицію спрайту
   setSpriteToStart();
   //Якщо спрайт стикається з яким-небудь спрайтом - змінимо його позицію
   while (howManyCollides() > 0)
   {
      setSpriteToStart();
   }
   //Задамо випадковий колір для додання зображенню
   //відповідного відтінку
      sprColor = new Color((byte)randNum.Next(0, 255), (byte)randNum.Next(0, 255),
(byte)randNum.Next(0, 255));
   //Змінна для зберігання швидкості поки порожня
   speed = new Vector2((float)randNum.Next(-5, 5), (float)randNum.Next(-5, 5));
   // TODO: Construct any child components here
}
//Перевірка, чи не встановлені спрайт в позиції з перекриттям інших спрайтів
int howManyCollides()
{
   int howMany = 0;
   foreach (spriteComp spr in myGame.Components)
   {
```

```
            if (this != spr)
            {
               if (this.sprCollide(spr))
               {
                  howMany++;
               }
            }
         }
         return howMany;
      }
      //Встановлення спрайту в випадкову стартову позицію
      void setSpriteToStart()
      {
         sprPosition.X = (float)randNum.NextDouble() * (scrBounds.Width - sprRectangle.
Width);
         sprPosition.Y = (float)randNum.NextDouble() * (scrBounds.Height - sprRectangle.
Height);
      }

      public override void Initialize()
      {
         // TODO: Add your initialization code here
         base.Initialize();
      }
      //Переміщення спрайту
      public virtual void Move()
      {
         sprPosition += speed;
      }
      //Перевірка допустимості переміщення
      void Check()
      {
         if (sprPosition.X < scrBounds.Left)
         {
            sprPosition.X = scrBounds.Left;
```

```
            speed.X *= -1;
        }
        if (sprPosition.X > scrBounds.Width - sprRectangle.Width)
        {
            sprPosition.X = scrBounds.Width - sprRectangle.Width;
            speed.X *= -1;
        }
        if (sprPosition.Y < scrBounds.Top)
        {
            sprPosition.Y = scrBounds.Top;
            speed.Y *= -1;
        }
        if (sprPosition.Y > scrBounds.Height - sprRectangle.Height)
        {
            sprPosition.Y = scrBounds.Height - sprRectangle.Height;
            speed.Y *= -1;
        }
    }
    public bool sprCollide(spriteComp spr)
    {
        return (this.sprPosition.X + this.sprRectangle.Width > spr.sprPosition.X &&
                this.sprPosition.X < spr.sprPosition.X + spr.sprRectangle.Width &&
                this.sprPosition.Y + this.sprRectangle.Height > spr.sprPosition.Y &&
                this.sprPosition.Y < spr.sprPosition.Y + spr.sprRectangle.Height);
    }
    //Функція повертає true якщо вказівник знаходився в
    //межах поточного об'єкта при натисканні мишею
    bool MouseCollide()
    {
        return (this.sprPosition .X +this.sprRectangle.Width >mouse .X &&
            this .sprPosition.X<mouse .X &&
            this.sprPosition .Y+this.sprRectangle .Height >mouse.Y &&
            this.sprPosition .Y <mouse .Y);
    }
```

```
public override void Update(GameTime gameTime)
{
    mouse = Mouse.GetState();
    if (mouse.LeftButton == ButtonState.Pressed)
    {
        //Якщо при натисканні мишею покажчик знаходився в межах
        //поточного об'єкта збільшуємо кількість набраних
        //очок і знищуємо об'єкт.
        if (MouseCollide())
        {
            myGame.Score++;
            this.Dispose();
        }
    }
    // TODO: Add your update code here
    //Виклик методу для переміщення спрайту
    Move();
    //Перевірка на зіткнення з межами екрану
    Check();
    //Виклик перевірки на зіткнення з іншими спрайтами
    IsSpriteCollide();
    //Можливо, при корекції спрайту щодо іншого спрайту,
    //відбулося перекриття з іншим спрайтом або спрайтами
    //це призводить до «зависання» спрайтів - вони залишаються на одному
    //місці в «зчепленому» стані. Для того, щоб цього
    //ми коригуємо позиції спрайтів до тих пір, поки кожен з
    // них гарантовано не опиниться поза інших спрайтів
    while (howManyCollides() > 0)
    {
        IsSpriteCollide();
    }
    base.Update(gameTime);
}
//Якщо спрайт перекрив інший спрайт - змінити його швидкість і
```

```
    //застосувати зміни до позиції спрайту
    void IsSpriteCollide()
    {
      foreach (spriteComp spr in myGame.Components)
      {
        if (spr != this)
        {
          if (this.sprCollide(spr))
          {
            this.speed *= -1;
            this.sprPosition += this.speed;
          }
        }
      }
    }
    public override void Draw(GameTime gameTime)
    {
      SpriteBatch sprBatch =
        (SpriteBatch)Game.Services.GetService(typeof(SpriteBatch));
      sprBatch.Draw(sprTexture, sprPosition, sprRectangle, sprColor);
      base.Draw(gameTime);
    }
  }
}
```

Лістинг 4.9. Код об'єкту spriteComp

Тут ми спочатку розставляємо спрайт таким чином, щоб вони не перетиналися, переміщаємо їх, з огляду на зіткнення один з одним і з межами екрану, а так само - відстежуємо стан миші. При натисканні мишею перевіряємо, чи не знаходиться позиція, в якій в цей момент знаходився покажчик, в межах поточного спрайту. Якщо це так - збільшуємо кількість очок і знищуємо поточний об'єкт. На рис. 4.5. ви можете бачити ігровий екран проекту P4_3.

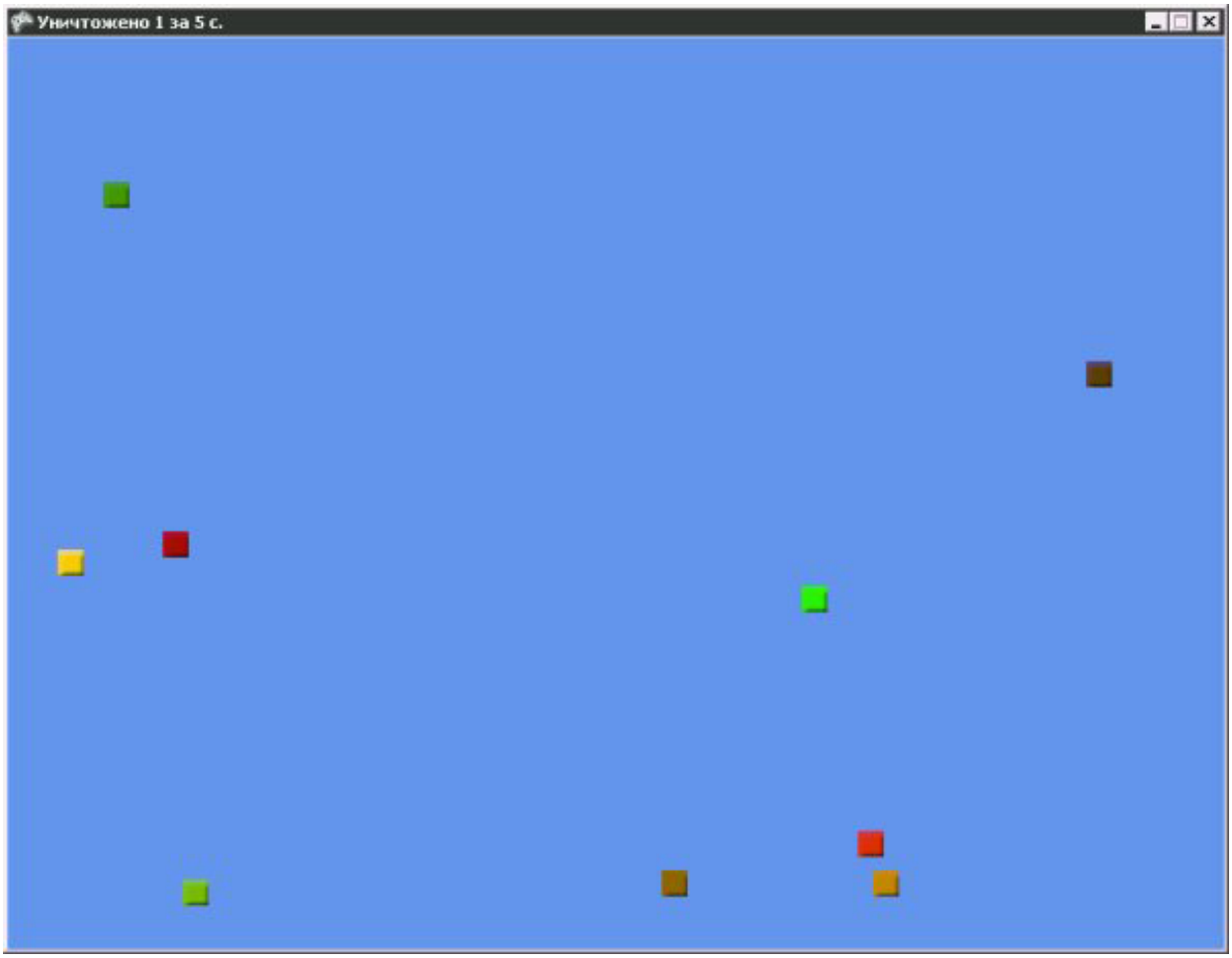

Рис. 8.5. Ігровий екран проекту P4_3

Розглянемо ще один приклад перевірки на «попадання» точки в певну область..

ОБРОБКА ПОПАДАННЯ ТОЧКИ В КРУГОВУ ОБЛАСТЬ.

Ідея перевірки попадання точки в межі кола така. Потрібно розрахувати відстань між точкою і центром кола. Якщо ця відстань не перевищить радіус кола - значить точка знаходиться всередині неї. Створимо новий стандартний ігровий проект P4_4, намалюємо коло і заміряємо її радіус. У нашому випадку радіус кола склав 250 пікселів. Додамо зображення в проект і виведемо його на екран. Включимо відображення покажчика миші в ігровому вікні. При натисканні лівою кнопкою миші обчислимо відстань від центру кола до точки, в якій стався клацання. Якщо ця відстань менше, ніж радіус кола - виведемо в заголовок вікна повідомлення «Ви потрапили в мішень!», В іншому випадку виведемо повідомлення «Ви не потрапили в мішень». Для реалізації даного прикладу ми обмежилися класом Game1, що не розробляючи окремих ігрових компнентов. У лістингу 4.10. ви можете бачити код класу Game1.

```
using System;
using System.Collections.Generic;
using Microsoft.Xna.Framework;
using Microsoft.Xna.Framework.Audio;
```

```
using Microsoft.Xna.Framework.Content;
using Microsoft.Xna.Framework.GamerServices;
using Microsoft.Xna.Framework.Graphics;
using Microsoft.Xna.Framework.Input;
using Microsoft.Xna.Framework.Net;
using Microsoft.Xna.Framework.Storage;
namespace P4_4
{
   /// <summary>
   /// This is the main type for your game
   /// </summary>
   public class Game1 : Microsoft.Xna.Framework.Game
   {
      GraphicsDeviceManager graphics;
      SpriteBatch spriteBatch;
      private Texture2D MySprite;
      private Vector2 position = new Vector2(150, 30);
      MouseState mouse;
      Vector2 Center;
      public Game1()
      {
         graphics = new GraphicsDeviceManager(this);
         Content.RootDirectory = «Content»;

      }
      protected override void Initialize()
      {
         // TODO: Add your initialization logic here
         base.Initialize();
         Center.X = position.X + 250+8;
         Center.Y = position.Y + 250+8;
         this.IsMouseVisible = true;
      }
      protected override void LoadContent()
```

```
{
    // Create a new SpriteBatch, which can be used to draw textures.
    spriteBatch = new SpriteBatch(GraphicsDevice);
    MySprite = Content.Load<Texture2D>(«circle»);
    // TODO: use this.Content to load your game content here
}
protected override void UnloadContent()
{
    // TODO: Unload any non ContentManager content here
}
protected override void Update(GameTime gameTime)
{
    // Allows the game to exit
    if (GamePad.GetState(PlayerIndex.One).Buttons.Back == ButtonState.Pressed)
        this.Exit();
    mouse = Mouse.GetState();
    if (mouse.LeftButton == ButtonState.Pressed)
    if (IsPointInCircle()) Window.Title = «Вы попали в мишень!»;
    else Window.Title = «Вы не попали в мишень!»;
    // TODO: Add your update logic here
    base.Update(gameTime);
}
bool IsPointInCircle()
{
  double length = Math.Pow(( Math.Pow((mouse.X - Center.X), 2) + Math.Pow((mouse.Y
- Center.Y), 2)),0.5);
        if (length <= 251) return true;
        else return false;
}
protected override void Draw(GameTime gameTime)
{
    graphics.GraphicsDevice.Clear(Color.CornflowerBlue);
    // TODO: Add your drawing code here
    spriteBatch.Begin();
```

```
            spriteBatch.Draw(MySprite, position, Color.White);
            spriteBatch.End();
            base.Draw(gameTime);
        }
    }
}
```

Лістинг 4.10. Код класу Game1

На рис. 4.6. ви може бачити ігровий екран проекту P4_4.

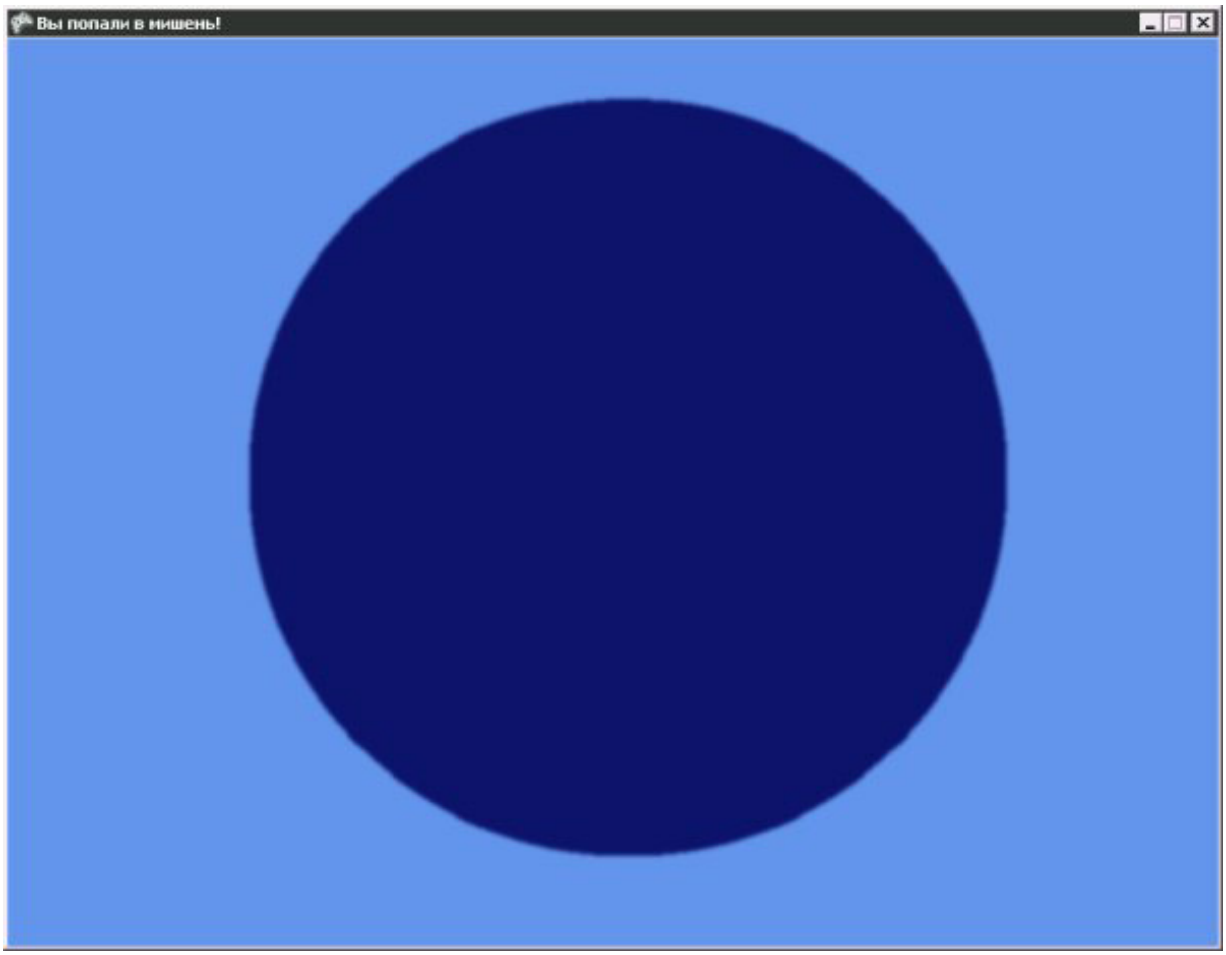

Рис. 4.6. Ігровий екран проекту P4_6

Навчальне видання

БРЕСЛАВЕЦЬ Віталій Сергійович

ТЕХНОЛОГІЇ РОЗРОБКИ КОМП'ЮТЕРНИХ ІГОР

Роботу до видання рекомендував *О.А. Сєрков*

Редактор *Л.П. Гобельовська*
Комп'ютерна верстка *Ю.Ф. Власова*
Коректор *Г.В. Сільченко*

Підписано до друку 07.02.2018 р.
Формат 60x84 1/16. Папір офсетний.
Цифровий друк.
Гарнітура DINPro. Ум. друк. арк. 9,42.
Наклад 50 прим. Зам. № ГЛ00-000476.

Видавець і виготовлювач:
ТОВ «Друкарня Мадрид»
61024, м. Харків, вул. Максиміліанівська, 11
Тел.:(057)756-53-25
www.madrid.in.ua info@madrid.in.ua

Свідоцтво суб'єкта видавничої справи:
ДК № 4399 від 27.08.2012 р.


9 786177 683048