КЛАСИЧНИЙ ПРИВАТНИЙ УНІВЕРСИТЕТ

ДО ЗАХИСТУ ДОПУЩЕНО

Зав. кафедри системного аналізу

та програмної інженерії

\_\_\_\_\_\_\_\_\_\_\_\_\_\_В.І. Горбенко

(підпис) (ініціали, прізвище)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

КВАЛІФІКАЦІЙНА РОБОТА

РОЗРОБКА ГРИ «GAME SHOOT» НА МОВІ JAVA

|  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- |
| Виконав  ст. групи | ЗI - 111 |  |  |  | Цаплюк О.С. |
|  | (шифр) |  | (підпис) |  | (ініціали, прізвище) |
| Керівник | ст. викл. |  |  |  | Р.Я Шумада |
|  | (посада) |  | (підпис) |  | (ініціали, прізвище) |
| Нормоконтролер | ст. викл. |  |  |  |  |
|  | (посада) |  | (підпис) |  | (ініціали, прізвище) |
| Рецензцнт |  |  |  |  |  |
|  | (посада) |  | (підпис) |  | (ініціали, прізвище) |

Запоріжжя

2016

КЛАСИЧНИЙ ПРИВАТНИЙ УНІВЕРСИТЕТ

Інститут управління

Кафедра системного аналізу та програмної інженерії

ЗАТВЕРДЖУЮ

Зав. кафедри системного аналізу

та програмної інженерії

\_\_\_\_\_\_\_\_\_\_\_\_\_\_В.І. Горбенко

(підпис) (ініціали, прізвище)

\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(дата)

## ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ

студенту групи

ЗІ-111

(шифр)

Цаплюк Олександр Сергійович

(прізвище, ім’я та по батькові)

Галузь знань: 0501 – Інформатика та обчислювальна техніка

Спеціальність: 6.050103 – Програмна інженерія

Освітньо-кваліфікаційний рівень – бакалавр

Форма навчання – денна (заочна)

1. Тема дипломної роботи: Розробка гри «game shoot» на мові java.

Затверджена наказом по університету № 34-зс від ”09” грудня 2015 р.

2. Термін здачі дипломної роботи ”12” травня 2016 р.

3. Цільова установка та загальний напрямок дипломної роботи: літературні дані за темою дослідження.

Метою даної дипломної роботи є створення арканної гри розважального характеру.

Об’єкт дослідження:Технологія розробки додатку на платформі Java.

4. Вихідні дані: літературні дані за темою дослідження.

5. Зміст дипломної роботи:

5.1. На основі аналізу літературних даних визначити основні властивості програмних продуктів та їх основні характеристики

5.2. Розробити додаток.

5.3. Розробити інтерфейси користувача.

5.4. Перевірити працездатність створеної програми.

5.5. Сформулювати висновки.

6. Календарний план виконання роботи:

|  |  |  |  |
| --- | --- | --- | --- |
| №  з/п | Назви частин роботи | Термін виконання | Позначка про виконання |
| 6.1. | Складання бібліографії та вивчення літературних джерел | 19.12.15р. |  |
| 6.2. | Збирання матеріалу. Написання вступу | 05.01.16р |  |
| 6.3. | Виконання розділу 1 | 06.02.16р |  |
| 6.4. | Виконання розділу 2 | 20.02.16р |  |
| 6.5. | Виконання розділу 3 | 02.03.16р |  |
| 6.6. | Виконання розділу 4 | 01.04.16р |  |
| 6.7. | Формулювання висновків. Оформлення пояснювальної записки дипломної роботи | 25.04.16р |  |
| 6.8. | Подання роботи до захисту | 19.05.16р. |  |

Дата видачі завдання ”19” грудня 2015 р.

Керівник роботи \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис керівника)

Завдання прийняв до виконання \_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_\_

(підпис студента)

РЕФЕРАТ

Звіт складається з 102 сторінок, 8 рисунків, 12 джерел, 5 таблиць.

Об’єкт дослідження — проектування програми на мові Java «Game shoot».

Предметом дослідження – розробка програми на мові Java «Game shoot».

Актуальність теми проектної практики полягає в тому, що комп'ютерні технології на сучасному етапі займають дуже важливу роль у житті кожної людини. Кожна сучасна людина вже не уявляє свого життя без комп’ютера або мобільного телефону, планшету, а також сьогодні неможливо уявити собі людину, яка не користується мобільним телефоном. А адже ще якихось 25-30 років тому цей пристрій був величезною розкішшю. Зараз мобільній телефон вже невід’ємна частина життя багатьох людей. І звичайно же з дуже великої швидкістю з’являються все нові та нові телефони з більшою функціональністю та максимальною зручністю. Найбільш сучасними зараз є смартфони.

При виконанні завдань практики використовувались: операційна система Windows, мова програмування Java, cсередовище програмування Eclipse.

КЛЮЧОВІ СЛОВА Windows, Java, Eclipse

ЗМІСТ

[ЗАВДАННЯ НА КВАЛІФІКАЦІЙНУ РОБОТУ 2](#_Toc450860352)

[ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ 8](#_Toc450860353)

[ВСТУП 9](#_Toc450860354)

[1. ОГЛЯД РІШЕНЬ 14](#_Toc450860355)

[1.1 Гра як невід’ємний атрибут комп’ютера. 14](#_Toc450860356)

[1.2 Огляд програмного забезпечення розважального характеру 15](#_Toc450860357)

[2 ІНСТРУМЕНТІ ТА ТЕХНОЛОГІЇ РОЗРОБКИ ІГОР 19](#_Toc450860358)

[2.1 Обумовлення вибору мови для розробки гри. 19](#_Toc450860359)

[2.2 Java - платформа 20](#_Toc450860360)

[2.3 Переваги Java-Платформи 23](#_Toc450860361)

[2.4 Внутрішня будова Java-Платформи 25](#_Toc450860362)

[2.5 Java Virtual Machine (Віртуальна машина Java) 27](#_Toc450860363)

[2.6 Середа розробки Eclipse 28](#_Toc450860364)

[2.7 Візуалізація 2D зображення за допомогою технології Java 31](#_Toc450860365)

[2.8 Використання Спрайтів для створення анімації 33](#_Toc450860366)

[3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ 35](#_Toc450860367)

[3.1 Постановка задачі до створюваної гри «Gameshoot» 35](#_Toc450860368)

[3.2 Побудова моделі гри. 37](#_Toc450860369)

[3.3 Створення проекту в середовищі розробки Java 2D 39](#_Toc450860370)

[3.4 Створення елементів коду гри 41](#_Toc450860371)

[3.5 Умови виконання програми 44](#_Toc450860372)

[3.6 Демонстрація роботи програми 45](#_Toc450860373)

[3.7 Критерій якості програми 48](#_Toc450860374)

[4. РОЗРАХУНОК ЕКОНОМІЧНОГО ЕФЕКТУ ВІД РОЗРОБКИ І ПРОДАЖУ ПРОГРАМИ 50](#_Toc450860375)

[4.1 Визначення витрат на розробку програми 50](#_Toc450860376)

[4.1 Розрахунок вартості програми 52](#_Toc450860377)

[4.2 Розрахунок економічного ефекту від продажу програми 52](#_Toc450860378)

[5 ОХОРОНА ПРАЦІ КОРИСТУВАЧІВ ПК 54](#_Toc450860379)

[5.1 Вимоги до виробничих приміщень для експлуатації ПК 54](#_Toc450860380)

[5.2 Мікроклімат виробничих приміщень 55](#_Toc450860381)

[5.3 Гігієнічні вимоги до організації та обладнання робочих місць 60](#_Toc450860382)

[5.4 Вимоги до режимів праці та відпочинку при роботі з ПК 61](#_Toc450860383)

[5.5 Вимоги до профілактичних медичних оглядів 63](#_Toc450860384)

[ВИСНОВКИ 65](#_Toc450860385)

[СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ 66](#_Toc450860386)

[ДОДАТОК А 67](#_Toc450860387)

# ПЕРЕЛІК УМОВНИХ СКОРОЧЕНЬ

IDE - Integrated Development Environment;

SDK - Software Development Kit;

API - Application Programming Interface;

JRE - Java Runtime Environment;

JDK - Java Development Kit;

JVM – JAVA VIRTUAL MACHINE;

MMORPG – MASSIVELY MULTIPLAYER ONLINE ROLE-PLAYING GAME;

MMOG – MASSIVELY MULTIPLAYER ONLINE GAME;

MOBA – MULTIPLAYER ONLINE BATTLE ARENA .

# ВСТУП

Зараз ХХІ століття світ новітніх технологій. ХХІ століття - час  народження нових ідей. Щоб засвоєні знання переходили в  уміння й навички, раціональні способи розв’язання завдань, щоб навчальний результат уроку поєднувався з розвитком пізнавальних здібностей учнів, їх прагненням учитися, потрібно здійснювати різнобічний розвиток учнівської молоді,  удосконалювати досвід творчої діяльності, зміцнювати світоглядні уявлення і поняття, цілеспрямовано розвивати процеси сприймання інформації. Сьогодні настав час  перевести навчання  на інноваційну основу, змінивши сам підхід до здобуття знань. Не виключенням є і розвок комютерних ігор. Комп'ютерні ігри беруть свій початок у 50-ті роки двадцятого століття. Сьогодні ігрова громадськість до кінця не визначилася, хто є їх початковим творцем. Історія знає трьох осіб, які в 1950-ті роки починали працювати над даним питанням. Першим з цих людей є Ральф Баер. Будучи інженером в 1951 році він запропонував громадськості ідею інтерактивного телебачення. Другою людиною є А.С. Дуглас. У 1952 році їм була написана гра, що отримала назву «ОХО». Дана гра являла собою програмну реалізацію всім відомих «хрестиків-нуликів». І третьою людиною, якого можна назвати батьком комп'ютерних ігор, є Вільям Хігінботем. У 1958 році він створив комп'ютерну гру «Теніс». У неї могли грати дві людини. У першому десятилітті 21-го століття ігрова індустрія йде вперед семимильними кроками. Щорічно випускаються тисячі ігор, що розходяться по всьому світу мільйонами копій, а оборот складає десятки мільярдів доларів. Найбільш успішні ігрові проекти збирають сотні мільйонів доларів від продажу. Цей відбувається на наших очах процес повинен бути осмислений у відповідних наукових категоріях, тому що дозволить більшою мірою зрозуміти специфіку і різноманітність актуальною сучасної культури. Дослідження комп'ютерних ігор в сучасних соціальних і гуманітарних науках - порівняно новий напрямок. Рідкісні дослідники звертаються до історії розвитку цієї категорії ігор (і то, більшою мірою в зарубіжній науці - Кент, Л.Стівен та ін.) Дещо більше зроблено в галузі соціально-психологічного вивчення проблем ігрової адикції, девіації і вікового розвитку (Е.Кафаі, Ю.Провензо). Досить багато рекламних матеріалів та виступів у ЗМІ. Серед серйозних праць домінують прикладні розробки по створенню комп'ютерних ігор - від наративу, сюжету і героїв до програмних засобів та інструментів штучного інтелекту. В цілому ж, слід визнати, що незважаючи на повсюдне поширення комп'ютерних ігор в сучасній культурі, наукове освоєння цього явища (особливо у вітчизняних науках) було недостатнім, позасистемним, «пунктирним», або ж - стосувалося лише окремих приватних його проявів.

Також, незважаючи на наявність ряду цікавих вітчизняних аналітичних робіт, заснованих на проголошеному міждисциплінарному підході (наприклад, роботи Д.В.Галкіна), дослідження будується більшою мірою тільки на історико-культурному підході і не зовсім актуальних прикладах.

У спробі визначити об'єкт дослідження ми повинні зафіксувати деякі важливі передумови. По-перше, як вірно зазначає Д.В. Галкін «комп'ютерні ігри є частиною найширшого світу ігор зі своєю історією ігрових форм і особливим змістом досвіду, знаходимо людиною в грі. Віхою в історії цього ігрового світу, безумовно, стала поява нового виду ігор, більш загальним і більш коректним терміном для яких слід вважати термін «електронні ігри» (або «відеоігри»), оскільки комп'ютерні ігри виникли пізніше буму електронних ігрових приставок в 70-х роках ХХ століття. По-друге, комп'ютерні ігри відносяться до більш загальній групі явищ, пов'язаних з гібридизацією художніх і технологічних об'єктів. Нарівні з комп'ютерною графікою і веб-дизайном, комп'ютерні ігри є техно-художніми гібридами, в яких технологічна основа служить не тільки інструментом створення художнього продукту, але включена в художній зміст та естетичні властивості твору ». У цьому зв'язку, а також зважаючи на те, що багато комп'ютерні ігри сьогодні претендують на звання творів мистецтва (це і згадана American McGee's Alice, Fallout 3 і безліч інших) актуально зафіксувати естетичний вимір сучасних комп'ютерних ігор, їх художній аспект. В якості можливих перспектив естетичного дослідження можна буде, наприклад, спробувати визначити трансформації категорій класичної естетики у віртуальних світах комп'ютерних ігор. Великий інтерес представляє недооцінений культурологічний ресурс сучасних комп'ютерних ігор. Необхідність цього дослідження полягає в тому, що вони є репрезентативним феноменом т. н. «Електронної культури» (Digital Culture, або E-culture) - новітній галузі сучасної культури, безпосередньо пов'язаної з інтенсивним розвитком інформаційно-комп'ютерних технологій в кінці XX - початку XXI ст., Яка поряд з комп'ютерними іграми включає в себе такі актуальні феномени як комп'ютерна «віртуальна реальність», численні феномени Інтернету, мистецтво (net-art), нові інтерактивні аудіовізуальні художні твори, різні нові види мистецтва, безпосередньо що з'являються разом з винаходом новітніх інформаційних технологій.

У той же час, незважаючи на очевидну актуальність, затребуваність явищ електронної культури, її феномени мало вивчені в сучасній вітчизняній науці, практично не існує серйозних досліджень «електронної культури» в культурології. У цьому контексті ретельний аналіз віртуальних світів комп'ютерних ігор дозволить глибше дослідити цю нову область сучасної культурології.

Важливий аспект культурологічного аналізу сучасних комп'ютерних ігор пов'язаний з дослідженням акутальнейшего феномена «віртуальної реальності». Широка популярність цього терміна призвела до того, що «віртуальна реальність» стала збірним позначенням безлічі феноменів кінця XX в.: Телебачення, електронних комунікацій, мультимедіа. На даний момент не існує цілісної концепції віртуальної реальності, в широкому сенсі вона може розглядатися як «будь змінені стани свідомості» або навіть як «реальна» життя, у вужчому - як вид екранного мистецтва або як закінчене втілення стилю і настроїв постмодерністської культури, і , нарешті, в самому вузькому сенсі - як реальність, створювана при безпосередній взаємодії з комп'ютером.

При всьому різноманітті систем віртуальної реальності, «їх об'єднує вироблений ефект занурення (immersion). Він полягає в тому що користувач перестає відчувати себе зовнішнім спостерігачем і включається в віртуальне оточення, починає сприймати його як «справжнє», тобто, по суті, віртуальна реальність намагається репрезентувати ефект достовірності і тотальності досвіду. Як і у випадку віртуальна реальність стимулює ефект чуттєвої достовірності, причому не тільки візуальної, а й, наприклад, дотиковий. Комп'ютерні віртуальні світи, як і світи переживання, володіють власною і каузальністю: час у віртуальній реальності оборотно, що, з урахуванням динаміки віртуальної реальності, може призводити до порушень причинно-наслідкових зв'язків. «Будучи результатом наукових досліджень і розвитку техніки в рамках раціональної традиції, технологія віртуальної реальності виявляє її заперечення, протиставляючи досвід потоку суб'єктно-об'єктної розірваності, довіру чуттєвим даними - аналітиці, релятивізм - струнким ієрархічним структурам». Як і віртуальна реальність, з одного боку, є формою сприйняття; з іншого - конструювання, причому ця здатність виявляється культурно-детермінованої, наповненою образами тієї іншої культурної традиції (наприклад, середньовічний антураж фентезійних комп'ютерних ігор). У прикладному контексті комп'ютерні ігри можуть знайти застосування в інноваційно-освітньому процесі, наприклад, в якості спеціальних ігрових навчальних програм, що використовуються як у ході лекцій, так і в ході заліків, екзаменаційних тестів. Інший продуктивний варіант - використання ігрових навчальних програм студентами під час профорієнтаційної практики.

У світлі заохочення урядом РФ розвитку актуальних інноваційно-модернізаційних проектів, видається актуальним дослідити модернізаційно-технологічний ресурс сучасних комп'ютерних ігор і їх значення як рушійної сили технологічного прогресу. Необхідно відзначити, що нові комп'ютерні ігри постійно задають усе більш високу візуальну (графічну) планку, яка вимагає від творців hardware (апаратного забезпечення) все більш досконалих технічних продуктів. Як приклади можна привести знаменитий Unreal (у 1998 р. його поява стала справжньою графічної революцією), TES4: Oblivion, Dragon Age або Napoleon: Total War. Популярність же і масовість таких високобюджетних ігрових продуктів, в свою чергу, змушує кінцевих користувачів частіше робити upgrade свого hardware, підвищуючи продажу і роблячи економічно доцільним подальший розвиток апаратного комп'ютерного забезпечення.

У підсумку, сучасні комп'ютерні ігри виявляють широкий спектр інноваційних можливостей, які можуть бути актуальні не тільки в світлі технологічної модернізації країни, а й у контексті перспективних соціально-гуманітарних досліджень. Було вирішено перевірити це на практиці.

# ОГЛЯД РІШЕНЬ

## Гра як невід’ємний атрибут комп’ютера.

Коли з'явилися перші комп'ютерні ігри, хто б міг ним передбачити того фурору, що вони справили згодом. І це не дивно, адже перші комп'ютерні ігри були аніскільки не схожі на гри сьогоднішні, які представляють хіба що повноцінне продовження реальності. Перші гри були схематичними й навіть досить незграбні. Один із перших ігор називалася “SрасеWаг”. Цілком можливо, нинішнє покоління гравців збирався б, що й невміло розігрують, якби хтось запропонував зіграти їм у неї. Отже, “SрасеWаг” — “Зоряний війни”. Усе дуже просто. Два квадратики зображують космічні кораблі. У тому завдання входить стріляти один одного (на поразка) мерехтливими точками і відхилятись від схематичних астероїдів.

З того часу комп'ютерні ігри пережили справжню еволюцію, перетворившись із простеньких, невигадливих істот в розкішні створенні з чудовою графікою, продуманим сюжетом, реалістичним звуком. Однією з результатів еволюції є розмаїття видів. Еволюція ігор стала винятком серед інших еволюції. Сьогодні комп'ютерні ігри містять у собі різні жанри та напрями. Єдине ж, що об'єднує, те що з них грають у вигляді комп'ютера, а що грає у яких називають «геймером».

Разом з приходом комп'ютерів з'явилися комп'ютерні ігри, що ж знайшли масу шанувальників. Ці ігри робилися із дитинства супроводжують підростаюче покоління, викликаючи, з одного боку, уповільнення розвитку і атрофію опорно-рухового апарату і м'язової мускулатури і, з іншого боку - швидко розвиваючи інтелект, логічне мислення та уяву людини. Комп'ютерний гравець звикає переміщатися вже з віртуального світу у інший, швидко сприймати незнайомі ситуації та адаптуватися до них. У бурхливо змінюваному суспільстві ХХІ сторіччя розвинена інтелектуальна гнучкість забезпечить пристосування до нових, несподіваним реаліям. Комп'ютерні гри виконують, в такий спосіб, функцію соціалізації у постіндустріальному суспільстві. Розвиток дослідницько-експериментальної і вдосконалення ігор був із розвитком комп'ютерного забезпечення і технологій. Тепер чимало людей складові комп'ютерів розробляються, хіба що спеціально для ігор. Наприклад, дорогі відеокарти, вартість яких сягає половини вартості задовільного комп'ютера до роботи на офісі. Усі гри розробляються з урахуванням останніх новинок комп'ютерна техніка, реагуючи попри всі досягнення і ближче підходячи до реальності зображення звуку. Сьогодні існують які своєю правдоподібністю ігри робилися із хорошим графічним і звуковим оформленням, майже зовсім імитуючим життя. Є й постійно виникає дуже багато фірм, які дають нові і призначає нові гри вкрай різноманітного характеру. Хоча історія комп'ютерних і відеоігор охоплює п'ять десятиліть, частиною поп-культури вони почали аж наприкінці 1970-х, 1950-х і 1960-х роках. Вони на таких платформах, як осциллографи, університетськ імейнфрейми і комп'ютери ESDAC. Винахід відеоігор зазвичай приписують комусь із трьох людей: РальфуБаеру, інженеру,запропонувавшому 1951-го ідею інтерактивного телебачення, О.С. Дугласу, створеним ним в 1952 “OXO” - комп'ютерну реалізацію “ крестиков-ноликов”, чи Вільяму Хигинботему, хто у 1958 року створив гру “TennisForTwo”.

## Огляд програмного забезпечення розважального характеру

Більш ніж 90% сучасних дітей і безліч дорослих грають у відеоігри, тому безсумнівно на сьогоднішній день розробка комп’ютерних ігор є найбільш розповсюдженою в IT-сфері. Вони є повсюди: web-ігри, flash-ігри: мобільні ігри для смартфонів і планшетів, консольні ігри, глобальні ігри типу MMORPG, MMO, MMOG, MOBA; сервісні ігрові станції, звичайні ігри для ПК і багато іншого. Конструювання якісних комп’ютерних ігор входить у клас найскладніших задач розробки ПЗ в цілому.

Відеоігри можуть бути корисними, а можуть і нашкодити. Якщо діти займаються ними в розумних межах, це сприяє, поряд з іншою їхньою діяльністю, розумовому розвитку і допомагає засвоювати нові навички. Перш за все, дитина набуває і вдосконалює навички роботи з комп'ютером, що стане в нагоді йому при подальшому навчанні. Правильно підібрані комп'ютерні ігри розвивають у дітей мислення, увагу і логіку, пам'ять і здатність приймати рішення. Управління грою покращує координацію рухів дитини. Навчальні ігри допомагають підготувати його до школи. Групові відеоігри вчать терпінню і взаємодії в групі. Деякі ігри розвивають творчу уяву.

Отже, якщо неправильно підходити до розробки комп’ютерних ігор, можна натрапити на наслідки руйнівного характеру, тому необхідно спрямовувати цю енергію в потрібне русло. Саме тому однією з головних задач є розробка продукту, який би вміло поєднував корисні якості і водночас ніс у собі розважальний характер.

Розробка комп’ютерних ігор дуже широка галузь програмування, тому ігри поділяються на велику кількість різновидів (таблиця 1), та більшість з них класифікуються за наступним жанрами:

Екшен - жанр комп'ютерних ігор, в яких успіх гравця залежить, насамперед від його швидкості реакції і здатності швидко приймати тактичні рішення. Дія у таких іграх розвивається дуже динамічно і вимагає напруження уваги і швидкої реакції на події, що відбуваються у грі.

Симулятори – жанр комп'ютерних ігор, які імітують управління яким-небудь процесом, апаратом або транспортним засобом. За допомогою комп'ютерно-механічних симуляторів, абсолютно точно відтворюють інтер'єр кабіни апарату, а це, в свою чергу, дозволяє тренуватися пілотам, космонавтам, машиністам високошвидкісних поїздів та іншим.

Стратегії – ігри, що вимагають планування і вироблення певної стратегії для досягнення якоїсь конкретної мети, наприклад, перемоги у військовій операції. Гравець управляє не одним персонажем, а цілим підрозділом, підприємством або навіть всесвітом. Розрізняють покрокові стратегічні ігри, де гравці по черзі роблять ходи, і кожному гравцеві відводиться необмежений або обмежений (залежно від типу і складності гри) час на свій хід, та стратегічні ігри в реальному часі, в яких всі гравці виконують свої дії одночасно, і хід часу не переривається. Щоб здобути перемогу в таких іграх, потрібно мислити на кілька кроків вперед, випереджаючи суперника. Тому Стратегічні ігри розвивають аналітичне мислення на рівні з грою в шахи.

Пригоди – гра-розповідь у якій керований гравцем герой просувається по сюжету і взаємодіє з ігровим світом за допомогою застосування предметів, спілкування з іншими персонажами і вирішення логічних завдань. Розвивають уяву і фантазію, а також здібність приймати рішення.

Музичні ігри – жанр комп'ютерних ігор, де на перше місце ставиться музична складова, тому допомагають розвивати почуття ритму та слух.

Головоломка – ігри, метою яких є вирішення логічних завдань, що вимагають від гравця задіяння логіки, стратегії та інтуїції.

Метою мого дипломного проекту є розробка програми, яка б несла, насамперед, розважальний характер, але включала б компоненти розвитку зорової пам’яті. Призначена для людей різних вікових категорію.

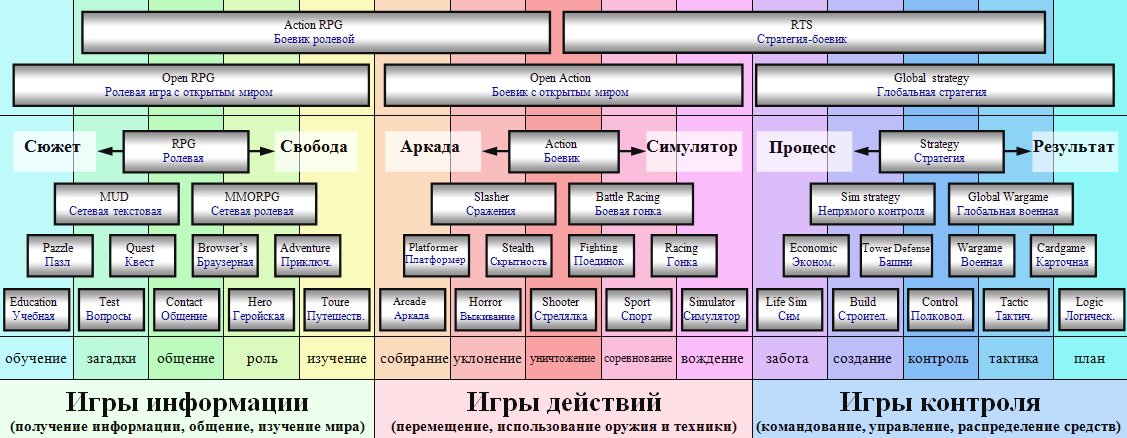


Рисунок 1 Різновиди ігор за категоріями

Було переглянуто і проаналізовано безліч комп’ютерних ігор різних жанрів, але майже не знайшлося тих, які б задовольняли цим умовам.

Тому ця розробка є актуальною, вона допомагає відпочити після важкого робочого дня, водночас розвиває зорову пам'ять, та при цьому не займає забагато часу, тому не відволікає надовго від важливих справ.

# ІНСТРУМЕНТІ ТА ТЕХНОЛОГІЇ РОЗРОБКИ ІГОР

## Обумовлення вибору мови для розробки гри.

На данный момент існує велика кількість мов програмування. В кожної з них є свої переважні сфери використання та недоліки.

C# і Java - дві найзатребуваніші мови, які сильно вплинули на усе сучасне програмування. Обидві мови мають C++ подібний синтаксис.

Обидві мови - об’єктно-орієнтовані, з синтаксисом, успадкованим від C++, та найбільш розповсюдженими у сфері розробки ігор. Для створення гри «Gameshoot» головними критеріями є: кросплатформеність готового проекту, легкість та практичність написання коду гри а саме:

Інкапсуляція. В Java модифікатор protected в описі, крім доступу з класів-нащадків, вирішує доступ зі всіх класів, що входять в той же пакет, що і клас-власник.

Внутрішні класи. Внутрішні класи Java мають доступ до нестатичних членів батьківського класу, тобто «знають про this»; крім того, усередині методів можна визначати локальні класи, що мають доступ по читанню до локальних змінних, і безіменні (анонімні) локальні класи, які фактично дозволяють створювати екземпляри об’єктів і інтерфейсів, що перекривають методи свого класу. На цьому механізмі в Java-програмах може будуватися обробка подій..

Методи. Тіло методу розташовується усередині опису класу. Підтримуються статичні методи, абстрактні методи. У Java параметри методу передаються тільки по значенню, але оскільки для екземплярів класів передається посилання, ніщо не заважає змінити в методі екземпляр, переданий через параметр.

Примітивні типи. Мова підтримую ідею примітивних типів, і обидва для трансляції примітивних типів в об’єктні забезпечують їх автоматичне згортання в об’єкти

В Java відмовилися від більшості беззнакових типів ради спрощення мови. Одна з відомих проблем з такими типами - складність визначення типу результату арифметичних операцій над двома аргументами, один з яких є знаковим, інший - беззнаковим.

Перевантаження операцій. Java не включає перевантаження операцій щоб уникнути зловживань нею і для підтримки простоти мови..

Умовна компіляція. Java версій 1.4 і вище включає в мову можливість перевірки допущень, що включається під час виконання. Крім того, конструкції if з константними умовами можуть розгортатися на етапі компіляції.

Таким чином ми бачимо, що і Java є потужною мовою, а також потужною платформою.

## Java - платформа

На теперішній час комп'ютерному світу відомо багато різних платформ, серед яких такі як: Microsoft Windows, Macintosh, OS/2, UNIX ® і система Netware ®. Для встановлення на кожну з них програмного забезпечення, воно повинне бути відкомпільоване окремо. Двійковий файл додатка, що виконується на одній платформі, не може бути запущений на іншій, тому як двійковий файл – специфічний, залежно від платформи.

Платформа Java – нова програмна платформа для транспортування й виконання високо інтерактивних, динамічних і безпечних аплетів і додатків на системах мережевих комп'ютерів. Основною якістю Java-Платформи, що виділяє її серед інших, є те, що вона розташовується на самому верхньому рівні в інших платформах, що дозволяє їй робити компіляцію в байт-коди, не прив'язані до кожної з фізичних машин і представляють собою машинні інструкції для віртуальної машини (virtual machine). Програма, написана мовою Java, компілюється у файл байт-коду, що може працювати скрізь, де присутня Java-Платформа, на кожній з основних операційних систем. Інакше кажучи, той самий файл буде виконуватися на будь-якій операційній системі, на якій присутня Java-Платформа. Подібна мобільність стає можливою завдяки тому, що в основі Java-Платформи лежить віртуальна Java-Машина.

У той час як кожна платформа має у своєму розпорядженні свою власну реалізацію віртуальної Java-Машини, всі віртуальні машини задовольняють вимогою єдиної специфікації. Завдяки цьому, платформа Java може реалізовувати єдиний стандарт – універсальний програмний інтерфейс для аплетів і додатків на будь-яких апаратних засобах. Тому Платформа Java є ідеальною для Інтернету, де та сама програма повинна бути здатна до виконання на будь-якому комп'ютері у світі.

Розробники використовують мову Java при написанні вихідного коду для Java-додатків. Вони компілюють свій код один раз і позбуваються тим самим від необхідності компілювати його для кожної системи окремо. Вихідний текст мови Java компілюється в проміжну, переносну форму байт-коду, що запуститься скрізь, де є присутнім Java-Платформа.

Розробники можуть писати объектно-оріентовані, багатопоточні, динамічно зв'язані додатки, використовуючи мову Java. Платформа має вбудовані системи захисту, обробки виняткових ситуацій, і автоматичного «збору сміття». Крім того, існує можливість використовувати JIT (just-in-time) компілятори (компілятори «на льоту») і прискорити виконання програм за допомогою перетворення байт – кодів Java у машинну мову. Також, розробники можуть записувати й викликати так звані нативні методи – методи C, C++ або інших мов, відкомпільовані для певної операційної системи – для підвищення швидкості виконання або для застосування спеціальних функціональних можливостей.

Програми, написані мовою Java і потім відкомпільовані, будуть запускатися на Java-Платформі. Платформа Java має дві основних частини:

* Java Virtual Machine (віртуальна Java-Машина);
* Java API (прикладний програмний інтерфейс Java).

У сукупності, ці частини забезпечують оперативні засоби керування роботою програми для кінцевого користувача при установці інтернет-додатків.

The Java Base Platform – «мінімальна» Java платформа, створена для запуску Java-Аплетів і додатків, що розробники можуть без проблем встановити й використовувати. Дана платформа призначається для мережевих, настільних комп'ютерів і робочих станцій. Платформа містить у собі ту ж віртуальну машину, що описувалася вище й при цьому має мінімальний комплект API, необхідним для запуску основних аплетів і додатків. Згаданий мінімальний комплект відомий, як Java Applet API або Java Base API. Розробники, які пишуть для цього комплекту можуть бути впевнені в тім, що програма запуститься скрізь без необхідності в підключенні додаткових бібліотек класів.

Деякі ліцензіати платформи Java уклали контракти про включення приватних реалізацій Java Base API в Java – платформі. В міру розробки бібліотек класів, Java Base Platform розростаються й нові класи регулярно мігрують у встановлену на кожну ліцензійну операційну систему Java Base Platform.

Інший набір API, що називається Standard Extension API, визначений JavaSoft у партнерстві з провідними промисловими компаніями створений для розширення основних функціональних можливостей. Найближчим часом планується мігрувати деяку підмножину Standard Extension API в Java Base Platform.

Embedded Java Platform була розроблена для споживача, що використовує прилади з малими ресурсами й з більш спеціалізованою функціональністю, ніж мережевий комп'ютер. Наприклад, принтери, ксерокси, мобільні телефони й ін. Подібна апаратура може мати деякі специфічні властивості, а саме невеликий обсяг пам'яті, відсутність дисплея або неможливість зв'язку по мережі.

API, розроблений для такої платформи, називається Java Embedded API. Java Embedded API – найменший із прикладних програмних інтерфейсів, які можуть бути впроваджені в описані вище прилади й при цьому ефективно працювати. Оскільки дана платформа усе ще допрацьовується, Java Embedded API дотепер не може розглядатися як стандарт. Тому існує деяка невизначеність, пов'язана із складом API. Приблизно, він буде містити в собі пакети java.lang і java.util. Java-Додатки, написані для одного окремого пристрою, зберігають працездатність на широкому діапазоні подібних по своїй специфіці пристроїв.

## Переваги Java-Платформи

Переваги для кінцевого користувача: На сьогоднішній день, Java-Платформа забезпечує підтримку оперативного, інтерактивного наповнення World Wide Web, за допомогою «just-in-time» доступу до програмних ресурсів. Додатки легко доступні на всіх операційних системах одночасно, що звільняє користувача від необхідності вибору операційної системи по факту доступності додатків. Більш маленькі, менш дорогі спеціалізовані системи будуть, в остаточному підсумку, доступні для спеціалізованих додатків.

Переваги для розробника: Мова програмування Java – невелика, проста для вивчення система, оснащена всебічно, що розширюється набором, API. Розробники можуть «написавши один раз, запускати всюди», що дає мові Java величезну перевагу перед іншими мовами на ринку. Крім того, програми на Java на всіх операційних системах при компіляції перетворяться до того самого двійкового формату. Виграш у порівнянні із ситуацією, коли для установки програми на декількох платформах потрібно писати й компілювати код окремо для кожної платформи, очевидний. Програміст може працювати над додатком під однією платформою, скорочуючи при цьому час і вартість розробки, і бути впевненим, що його код буде працювати скрізь. Можливість «написавши один раз, запускати всюди», для багатьох програмістів є достатньою причиною для переходу до мови Java від таких мов як C або С++, на яких працездатність додатків залежить від платформи а, також, додатки яких «немережеві».

На додаток до цього, можливе створення додатків, що багаторазово використовують загальнодоступні об'єкти, що ще більше зменшує вартість розробок і дозволяє розробникам концентрувати свої зусилля тільки на створенні чогось нового.

Переваги при адмініструванні й підтримці: Java-Платформа має переваги, пов'язані з адмініструванням корпоративних комп'ютерних систем. Контроль версії й оновлення спрощуються, тому що Java-Додатки можуть зберігатися в центральному архіві й тут обслуговуватися для кожного окремо взятого користувача. У багатотипному, багатоплатформенному оточенні кількість Java-Платформ, яким необхідна технічна підтримка зменшується до однієї. У сукупності з тенденцією до зниження вартості мережеих комп'ютерів це приводить до зменшення як основних витрат, так і витрат на обслуговування. З подібними мережевими комп'ютерами керування даними може здійснюватися централізовано, у той час як обробка даних здійснюється локально.

Компанії з великими внутрішніми комп'ютерними мережами, які можуть не приділяти достатньо уваги для встановлення на всіх машинах новітнього програмного забезпечення або останніх операційних систем, можуть запускати Java-Додатки на всіх комп'ютерах у мережі. Крім того, впроваджуючи утримування корпоративних даних у форматах «розпізнаваних» Java додатками, корпорації дає своїм працівникам можливість доступу до будь-яких даних незалежно від платформи.

Якщо на Java-Платформі запустити клієнт, компанії можуть скористатися интерактивністю інтернету й перекласти необхідність відсилання завдань працівникам на клієнтський додаток. Також компанії можуть зменшити час, що витрачається на облік замовлень за допомогою впровадження спеціальних клієнтів, що містять бланки замовлень на Web сторінках. При цьому клієнти можуть бути запущені на будь-яких платформах.

## Внутрішня будова Java-Платформи

Java-Платформа складається із двох основних частин, Java Virtual Machine (віртуальна машина Java) і Java API.

Java Virtual Machine – Java Virtual Machine – це «запрограмований» комп'ютер, що може бути реалізований у програмному забезпеченні або в апаратних засобах. Це абстрактний пристрій, спроектований так, щоб бути реалізованим на як можна більшому числі сучасних процесорів. Інтерфейс переносів і адаптери дають JVM можливість бути перенесеною на нові операційні системи без необхідності в повнім переписуванні

Java API – Java API визначає стандартний інтерфейс для аплетів і додатків, не звертаючи уваги на встановлену на комп'ютер операційну систему. Java API – основа, каркас при розробці додатка. Даний API визначає набір стандартних інтерфейсів для використання в ключових областях, кількість котрих збільшується, у яких програмісти звичайно вибудовують свої Java додатки.

Java Base API (Основний прикладний програмний інтерфейс Java) забезпечує можливість роботи з різними допоміжними класами, з уведенням / виведенням, з мережею, з GUI (графічний інтерфейс користувача), і аплетами. Компанії-Виробники операційних систем, які мають ліцензію Java, підписали контракти, що зобов'язують їх включати Java Base API у будь-яку Java-Платформу, що вони встановлюють.

Java Standard Extension API (Стандартний розширений прикладний програмний інтерфейс Java). Є розширенням описаного вище Java Base API. Передбачається, що деякі розширення будуть згодом мігровані в Java Base API. Інші нестандартні API будуть підтримуватися в додатках, аплетах і основних операційних системах. При публікації специфікації будь-якого розширювального API до його остаточного виходу, в обов'язковому порядку публікуються промислові огляди з можливістю зворотного зв'язка з розроблювачами.

На представленому малюнку, Java Base Platform представлена частинами, зафарбованими чорним, включаючи блоки, підписані «Адаптер». Java API включає як Java Base API так і Java Standard Extension API. Класи є реалізаціями API. Java Virtual Machine лежить в основі платформи. Інтерфейс переносу розташовується між Java Virtual Machine і операційною системою (ОС) або броузером. Інтерфейс переносу складається із платформонезалежної частини (зафарбована чорним) і залежної від платформи частини з написом «Адаптер». ОС і JavaOS забезпечують роботу з віконним інтерфейсом, зберіганням даних і взаємодією по мережі. Як показано, різні машини можуть приєднуватися по мережі.

Основна частина Java API відкрита й розширювана. Специфікації для кожного інтерфейсу розвиваються всегалузевими фахівцями у всіх областях. Підготовлювані специфікації видаються й відкриваються для рецензування різними галузями промисловості. Реалізації специфікацій API надходять від JavaSoft і інших підприємств. У сучасному середовищі постійних інновацій, структура Java API дозволяє будь-якому нововведенню легко існувати у вигляді розширення до Платформи Java.

API розбивається по групами, або наборам. Кожний набір API може бути реалізований у вигляді одного або декількох пакетів (просторів імен). Кожний пакет поєднує в собі набір класів і інтерфейсів, які визначають набір полів, конструкторів і методів.

## Java Virtual Machine (Віртуальна машина Java)

Java Virtual Machine є ключем до незалежності від основної операційної системи й апаратних засобів. Це платформа, що приховує основну операційну систему від Java аплетів і додатків. Крім того, перенос Java Virtual Machine на броузер або іншу операційну систему не становить ніяких труднощів.

Віртуальна Машина визначає машинно-незалежний формат для двійкових файлів, що зберігаються з розширенням class. Цей формат включає команди для віртуального комп'ютера у формі байт-кодів. Подання байт-коду будь-якої програми, написаної мовою Java, символічне, у тому розумінні, що зсуви й індекси усередині методів не є константами, а представляються символічно, як строкові імена. При першому виклику методу, він відшукується у файлі class по імені, і в цей момент визначається числове значення його зсуву для більш швидкого доступу при наступних викликах. Тому, будь-який новий метод або метод, що перевантажує старий, може бути визначений у будь-якому місці в структурі класу. Він буде переданий у символічній формі, і програма належним чином виконається без порушень при виконанні коду.

Байт-Коди – високорівневі подання програми, які проходять оптимізацію й машинну генерацію об'єктного коду за допомогою just-in-time компілятора, після чого можуть бути виконані. Крім того, всередині Virtual Machine може запуститися «збір сміття», під час якої змінні втримуються в стеках в адресному просторі Java-Платформи.

## Середа розробки Eclipse

Якщо ви уважно стежите за розвитком OpenSource проектів або програмуванням із застосуванням Java, ви вже могли чути частина того шуму, який супроводжує Eclipse. Eclipse - це розширювана середу розробки (Integrated Development Environment, далі IDE) з відкритим вихідним кодом (далі OpenSource). Проект був запущений в листопаді 2001 року, коли IBM передала вихідний код свого Websphere Studio Workbench вартістю 40 мільйонів доларів на OpenSource і сформувала консорціум Eclipse для управління триваючої розробкою.

Мета створення Eclipse була сформульована таким чином: " розробити багату, повнофункціональну індустріальну платформу комерційної якості для розробки сильно - інтегрованих інструментів ". Для досягнення цієї мети консорціум націлений на три головних проекту:

1) проект The Eclipse безпосередньо Eclipse IDE (" платформи", що містить і виконуючою інструменти Eclipse), інструментів розробки для Java (Java Development Tools, далі JDT) і середовища розробки Plug -In (Plug - In Development Environment, далі PDE), що дозволяють розширювати платформу;

2) проект Eclipse Tools має своєю метою створення інструментів для платформи Eclipse (у поточній розробці знаходяться підпроекти створення IDE для Cobol, IDE для C / C + +, а також інструменту для побудови EMF моделей).

3) платформа Eclipse в поєднанні з JDT включає багато з можливостей, які включаються в комерційні IDE : редактор з підсвічуванням синтаксису, інкрементальних компіляція коду, потокобезпечна відладчик, навігатор по класах, менеджери файлів і проектів, а також інтерфейси до стандартних систем контролю вихідних текстів, таким як CVS і ClearCase.

4) проект Eclipse являє собою першу настільки потужно підтриману світовим IT -спільнотою спробу створення єдиної відкритої інтегрованої платформи розробки додатків, що володіє надійністю, функціональністю і рівнем якості комерційного продукту. Фактично ця платформа призначена для всього і ні для чого конкретно : вона являє собою основу, що має блочну структуру і інтегруючу інструменти розробки ПО різних виробників для створення додатків на будь-якій мові, з використанням будь-яких технологій і для будь програмної платформи. Навколо проекту Eclipse в даний час сформовано співтовариство найбільших IT -компаній, серед яких Borland, IBM, SAP AG, RedHat та інші.

Також Eclipse пропонує безліч унікальних можливостей, наприклад рефакторінг коду, автоматичне оновлення встановлення коду (за допомогою Менеджера Оновлень), список поточних завдань, налагодження модулів за допомогою JUnit та інтеграцію з інструментом компоновки Jakarta Ant.

Незважаючи на велику кількість стандартних можливостей, Eclipse відрізняється від традиційних IDE по ряду особливостей. Напевно найцікавіше в Eclipse те, що вона повністю незалежна від платформи і мови. Крім мов, підтримуваних консорціумом зараз (Java, Cobol, C / C + +), ведуться розробки по додаванню в Eclipse підтримки таких мов, як Python, Eiffel, PHP, Ruby, і C #.

Консорціум надає готові виконувані файли для Windows, Linux, Solaris, HP- UX, AIX, QNX і Mac OS X. Великий інтерес в Eclipse являє plug - in архітектура, а також багатий API, що надається PDE, що дозволяє розширювати Eclipse. Додавання підтримки для нового редактора, подання або мови програмування є досить простим, завдяки грамотно розробленим API і великим будівельним блокам, що надаються Eclipse.

Переваги:

Eclipse служить в першу чергу платформою для розробки розширень, чим він завоював популярність: будь-який розробник може розширити Eclipse своїми модулями. Вже існують Java Development Tools (JDT), C/C++ Development Tools (CDT), розробляються інженерами QNX спільно з IBM, і кошти для мови Ada (GNATbench, Hibachi), COBOL, FORTRAN, PHP і пр. від різних розробників.

Безліч розширень доповнює середовище Eclipse менеджерами для роботи з базами даних, серверами додатків і др.Eclipse JDT (Java Development Tools) - найбільш відомий модуль, націлений на групову розробку: середа інтегрована з системами керування версіями - CVS, GIT в основній поставці, для інших систем (наприклад, Subversion, MS SourceSafe) існують плагіни. Також пропонує підтримку зв'язку між IDE і системою управління завданнями.

У основне постачання включена підтримка трекера помилок Bugzilla, також є безліч розширень для підтримки інших трекерів (Trac, Jira і ін). В силу безкоштовності і високої якості, Eclipse у багатьох організаціях є корпоративним стандартом для розробки додатків. Eclipse написана на Java, тому є платформо-незалежною продуктом, за винятком бібліотеки SWT, яка розробляється для всіх поширених платформ. Бібліотека SWT використовується замість стандартної для Java бібліотеки Swing. Вона повністю спирається на нижележащую платформу (операційну систему), що забезпечує швидкість і натуральний зовнішній вигляд користувацького інтерфейсу, але іноді викликає на різних платформах проблеми сумісності та стійкості додатків.

Враховуючи сотні проектів по розробці plug - in, що ведуться в даний час, таких індустріальних гігантів як IBM, HP і Rational, що надають ресурси, а також проектних важковаговиків начебто Erich Gamma, що допомагають спрямовувати процес еволюції в потрібне русло - у Eclipse велике майбутнє.

## Візуалізація 2D зображення за допомогою технології Java

Методи комп'ютерної графіки знаходять широке застосування в сучасному проектуванні високотехнологічних промислових виробів. Двовимірна графіка, в свою чергу ділиться на растрову і векторну. Двовимірна графіка (2D) графіка використовується більшою зокрема при роботі з фото-банками, фотографіями, ілюстраціями та інколи з флеш - кінцевий результат рендера виводиться у форматі JPG, а не як вихідний файл. У кожного виду є свої гідність і недоліки при роботі з фото-банками. Растрова графіка - це класичний спосіб представлення комп'ютерної графіки, за своєю суттю - це масив різнокольорових пікселів, кожен з яких вимагає зберігання для нього кольору і координат (прообразом растрової графіки можна вважати вишивку хрестом, де зображення складається з кольорових хрестиків). Векторна графіка - це спосіб представлення зображення за допомогою математичних символів і примітивів. Растрова і векторна графіки в силу своїх відмінностей не є взаємозамінними, так як за допомогою растрової графіки найзручніше представляти фото-реалістичні зображення, наприклад, фотографії, а за допомогою векторної - малюнки та схеми. З одного боку - безперечною перевагою векторної графіки є можливість масштабування без втрати якості і невелика вага файлу з простими об'єктами, а з іншого - реалістичне зображення з дрібними деталями буде неймовірно великим за рахунок складності фігур, за допомогою яких вона створена.

Середа розробки програмного забезпечення, в якій найбільш поширені блоки програмного коду представляють у вигляді графічних об'єктів і застосовуються вони в основному для створення прикладних програм і розробки графічного інтерфейсу користувача (GUI). Слід враховувати, що деякі візуальні середовища розробки мають власний формат зберігання проекту і при переході на іншу середу може виникнути непереносимість властивостей проекту і деяких частин проекту, таких, як власні бібліотеки використовуваного середовища розробки. Середовищами розробки являються : HiAsm, Eclipse, Borland Delphi, Borland C + + Builder, Qt Creator, Dev - C + + і NetBeans.

Останні кілька років розробники докладали масу зусиль, щоб інтегрувати графіку і анімацію в свої аплети і додатки Java. Однак спочатку включені в Java графічні пакети AWT Java мали обмежені кошти для вирішення таких завдань. Зараз, використовуючи інтерфейси прикладного програмування Java 2D і Java 3D, розробники можуть реалізовувати набагато більш складні графічні додатки, включаючи ігри, зберігачі екрану, екранні заставки та тривимірний графічний користувальницький інтерфейс.

У технології Java графіка ускладнюється тим, що додатки Java повинні працювати з потрібною бібліотекою класів, незалежно від конкретної графічної системи. Класи бібліотеки AWT реалізують ці інтерфейси для створення додатків. Програми Java використовують ці методи для розміщення і переміщення графічних об'єктів, зміни їх розмірів, взаємодії об'єктів. Бібліотека класів Java, заснована на peer-інтерфейсах, отримала назву AWT (Abstract Window Toolkit). При виведенні об'єкта, створеного в додатку Java і заснованого на peer-інтерфейсі, на екран створюється парний йому (peer-to-peer) об'єкт графічної підсистеми операційної системи, який і відображається на екрані. Бібліотека "легких" компонентів Java, названа Swing. може використовуватися самостійно, незважаючи на те, що всі класи з неї розширюють класи бібліотеки AWT. Всі ці засоби Java 2: AWT, Swing, Java 2D, DnD, Input Method Framework і Accessibility склали бібліотеку графічних засобів Java, названу JFC (Java Foundation Classes). Створення коду гри було здійснено в середовищі розробки Еclipce на мові програмування Java. Для потрібного розташування об'єктів на карті та їх кількості, а так же характеристик спочатку був створений файл Bitmaps для експорту графічних елементів присвоївши їм назви і їх можливості. Для створення графіки елементів графічного інтерфейсу потрібні нам елементи графіки було взято з gif файлу. Все було укладено в один пакет com.mojang.tower.

## Використання Спрайтів для створення анімації

Графічний об'єкт в комп'ютерній графіці. Найчастіше - растрове зображення, вільно переміщається по екрану. Спостереження спрайта під невідповідним кутом призводить до руйнування ілюзії. Тобто найлегше сприймати спрайт як переміщатися в просторі проекцію якогось об'ємного тіла так, що різниця непомітна. Спочатку під спрайтами розуміли невеликі малюнки, які виводилися на екран із застосуванням апаратного прискорення. На деяких машинах (MSX 1, NES) програмна промальовування приводила до певних обмежень, а апаратні спрайт цього обмеження не мали. Згодом із збільшенням потужності центрального процесора, від апаратних спрайтів відмовилися, і поняття «спрайт» поширилося на всіх двовимірних персонажів. Зокрема, у відеоіграх Super Mario і Heroes of Might and Magic вся графіка спрайтові. До апаратно прискореним спрайт повернулися в середині 1990 -х років - коли розвиток мультимедіа та вибуховий підвищення дозволу і глибини кольору зажадало спеціалізований процесор в відеоплаті. Саме тоді, як обгортка над апаратним 2D - прискорювачем, вийшов DirectDraw. DirectX 8 ввів загальний API для двох - і тривимірної графіки, і в сучасних спрайтові іграх двомірні спрайт виводяться точно так само, як і тривимірні - як текстурований прямокутник. Спрайт сам розріже його на ту кількість кадрів, яке ми вкажемо йому при створенні. Насправді можна було звичайно трохи задурити і хоча б обійти вимогу на вибудовування кадрів в лінію, а й так працює цілком не погано). Отже, переходимо до конструктора. Він у нас отримує декілька параметрів - перше - це AssetManager, потім ім'я файлу в папці assets нашого проекту. Ну і звичайно число кадрів на які треба розбити спрайт. Конструктор завантажує спрайт використовуючи метод батьківського класу mSimpleSprite а потім викликає метод MakeFrames () - який і розіб'є картинку на деяку кількість окремих картинок і заповнить ними масив bmpList. Там же, для кожного кадру встановлюється час кількість показів за замовчуванням (за замовчуванням кадри показуються по одному разу).

У методі draw () крім простий отрисовки, ще й перемикається лічильник кадрів відповідно зі значеннями з масиву frameLen. Таким чином, що якщо для якогось i - того фрейма в i -том елементі масиву frameLen встановлено наприклад 5, то і сам фрейм буде відмалювали 5 разів, перш, ніж фрейм зміниться.

Встановити кількість показів для кожного кадру можна використовуючи метод setFrameLen (int [] frameLen). Якщо ми хочемо щоб кожен кадр нашої анімації з трьох кадрів перемальовувати наприклад по сім разів, то ми повинні передати такий масив : { 7, 7, 7 }. При цьому якщо нам потрібно щоб якісь кадри відображалися довше ми можемо підрегулювати це наприклад так : { 7, 1, 7 }, і тоді другий кадр анімації Отріс всього 1 раз. Знаючи фреймрейт і то скільки милисекунд повинен відображатися кадр не складно розрахувати відповідні значення з цього масиву.

# 3. РОЗРОБКА ПРОГРАМНОГО ЗАБЕЗПЕЧЕННЯ

## Постановка задачі до створюваної гри «Gameshoot»

Для того щоб ігрове застосування було зрозумілим для користувача, з досвіду розробників, були підкреслені наступні вимоги мінімалізму.

Мінімалізм ідеї. Ідея гри повинна бути простою і зрозумілою користувачу.

Проста ідея повинна запасти в душу гравцеві і запам'ятатися надовго.

Мінімалізм дизайну. Геймплей (ігровий процес) повинен бути максимально простим і зрозумілим без підказок і навчання.

Управління повинне бути простим і підходящим більшості гравців.

Геймлуп (цикл гри) і ігрова сесія мають бути короткими, але геймплей повинен давати можливість знову і знову перегравати (endless mode або можливість постійного поліпшення результатів).

Рівень повинен бути влаштований так, щоб завжди легко можна було б внести зміни і протестувати їх.

Меню повинно бути максимально простим, як можна менше переходів.

У грі повинно бути якомога менше тексту (назва гри і автори). Все інше - візуально (іконки, стрілочки і т.д.) і інтуїтивно зрозуміло. У дизайні повинно бути якомога менше цифр - це спрощує настройку. І, звичайно, як можна менше цифр (крім статистики) має бути надано на обробку гравцеві.

Маленька кількість об'єктів на екрані.

Об'єкти не повинні бути величезними. В крайньому випадку повинні будуватися з більш дрібних.

Простий і запам'ятовується зовнішній вигляд головного героя.

Головний герой повинен мати такий вигляд і форму, щоб анімація його була простою, але веселою. Нечіткий, без деталей задній фон.

Елементи графічного інтерфейсу повинні використовувати загальноприйняті стандарти. По максимуму використовувати поточні напрацювання.

Простий дизайн повинен привести до простих алгоритмів та структур даних.

По максимуму використовувати поточні напрацювання.

Простий дизайн повинен привести до простих алгоритмів та структур даних.

Чим менше коду, тим менше розмір ігрового застосування.

Мінімалізм звукового оформлення. Легка фонова музика, яка сподобається всім. Маленька кількість звуків.

Звуки повинні бути простими, не відволікаючими і не навантажувати, а занурювати в геймплей.

• старт програми має тривати не більше 10 секунд;

• перехід між рівнями має тривати не більше 10 секунд

Інтерфейс користувача – компонент, що відображає хід гри користувачеві та використовується для введення необхідних даних.

Інтерфейс програми має бути простим та зрозумілим навіть якщо користувач не прочитав інструкцію до програми:

• Основні елементи керування повинні бути доступні в будь- якому стані програми.

• Елементи керування повинні бути доступними на моніторах будь-якого розміру.

• Користувач має легко знаходити необхідні елементи керування.

• Кількість елементів керування, яка необхідна для успішної роботи з програмою, не повинна перевищувати 7 одиниць

• Користувач повинен мати можливість зупинити поточну операцію або можливість вимкнути програму.

• Має існувати можливість керування програмою лише за допомогою клавіатури, не користуючись мишею.

## Побудова моделі гри.

Діаграма варіантів використання (Use-Cases Diagram) - це UML діаграма за допомогою якої в графічному вигляді можна зобразити вимоги до розроблюваної системі.

Діаграма варіантів використання - це вихідна концептуальна модель проектованої системи, вона не описує внутрішній устрій системи. Діаграми варіантів використання призначені для:

Визначення спільного кордону функціональності проектованої системи

Сформулювати загальні вимоги до функціонального поведінки проектованої системи

Розробка вихідної концептуальної моделі системи

Створення основи для виконання аналізу, проектування, розробки і тестування.

Діаграма варіантів використання складається з ряду елементів. Основними елементами є: варіанти використання або прецедент (use case), актор або дійова особа (actor) і відносини між акторами і варіантами використання (relationship).

Як ми бачимо, на рисунку 4.2. зображена діаграма варіантів використання нашого ігрового застосування «Game shoot».

Актори:

Актором називається будь-який об'єкт, суб'єкт або система, що взаємодіє з розроблюваною бізнес-системою ззовні для досягнення своїх цілей або вирішення певних завдань. Це може бути людина, технічний пристрій, програма або будь-яка інша система, яка служить джерелом впливу на розроблювану систему. Актори взаємодіють з системою за допомогою передачі і прийому повідомлень від варіантів використання. Повідомлення являє собою запит актором сервісу від системи і отримання цього сервісу.

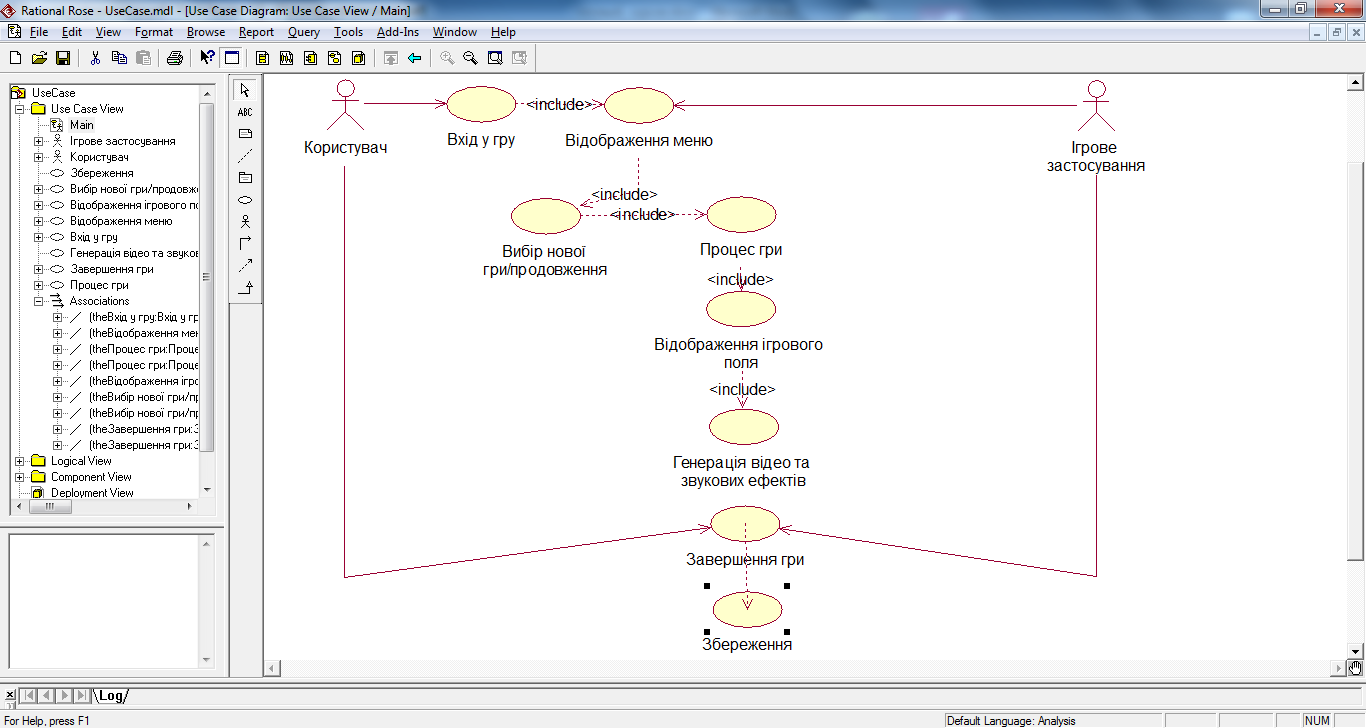


Рис. 3.1 - Діаграма варіантів використання

Актор «Користувач» - відіграє роль головного актору у всьому ігровому застосуванні. У реальному житті являє собою людину/гравця, зацікавлену у проходженні усіх етапів гри та отриманні головного призу/кубку. Користувач, використовуючи головне меню застосування має можливість входити у гру, продовжувати раніше початий процес або обирати новий, настроювати ефекти звуку, закінчувати процес гри.

Актор «Ігрове застосування» - актор, який являє собою програмні модулі, що відповідають за реалізацію ігрового застосування «NLO». Ігрове застосування має відображати головне меню гри, для вибору типу гри та налаштувань звукових ефектів. Також відображати ігрове поле, генерувати відео та звукові ефекти, зберігати кожен етап гри при виході.

Діаграма класів (class diagram) служить для представлення статичної структури моделі системи в термінології класів об'єктно-орієнтованого програмування. Діаграма класів може відбивати, зокрема, різні взаємозв'язки між окремими сутностями предметної області, такими як об'єкти і підсистеми, а також описує їхню внутрішню структуру і типи відносин. На даній діаграмі не вказується інформація про тимчасові аспекти функціонування системи. З цієї точки зору діаграма класів є подальшим розвитком концептуальної моделі проектованої системи.

Діаграма класів є деякий граф, вершинами якого є елементи типу "класифікатор", які пов'язані різними типами структурних відносин. Слід зауважити, що діаграма класів може також містити інтерфейси, пакети, відносини і навіть окремі екземпляри, такі як об'єкти і зв'язки. Коли говорять про даній діаграмі, мають на увазі статичну структурну модель проектованої системи. Тому діаграму класів прийнято вважати графічним представленому таких структурних взаємозв'язків логічної моделі системи, які не залежать або інваріантні від часу. Діаграма класів складається з безлічі елементів, які в сукупності відображають декларативні знання про предметної області. Ці знання інтерпретуються в базових поняттях мови UML, таких як класи, інтерфейси і відносини між ними та їх складовими компонентами. При цьому окремі компоненти цієї діаграми можуть утворювати пакети для представлення більш загальної моделі системи. Якщо діаграма класів є частиною деякого пакета, то її компоненти повинні відповідати елементам цього пакета, включаючи можливі посилання на елементи з інших пакетів.

## Створення проекту в середовищі розробки Java 2D

Для розробки було обрано середу розробки Eclipse. Для того що б створити проект в необхідно запустити середовище розробки і пройти за наступним шляхом File-> New Project внаслідок чого відкриється вікно.

Тут можна налаштувати назву програми, його розташування. Після визначення всіх необхідних параметрів приступаємо до створення проекту. Створюємо потрібні нам папки та файли і призначаємо між ними зв'язок.

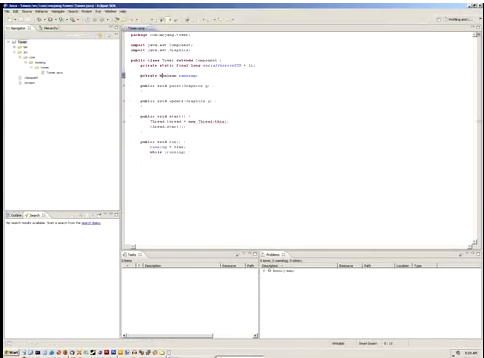


Рис. 3.2. Створення пакету

Тут бачимо шаблон, так звану основу з чого ми будемо починати додатки. Створюємо спочатку зв'язковий файл.

Це те з чого все й почалося. У даній програмі просто безмежні можливості для створення таких додатків у випадку з розробкою гри в самий раз.

## Створення елементів коду гри

Потрібно створити код для потрібного розташування об'єктів на карті їх кількості та характеристик. Спочатку створив фаїл Bitmaps для експорту всього що потрібно присвоївши їм назви і їх можливості.

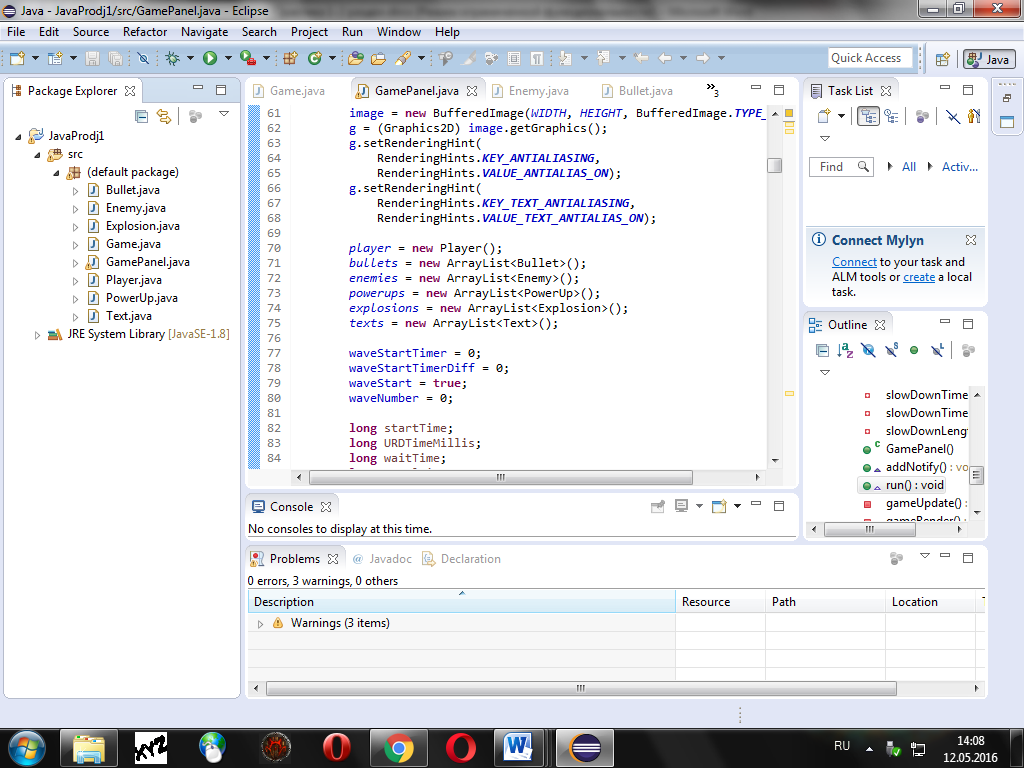


Рис. 3.3. Усі елементи коду гри

Таблиця3.1

Опис основних элементів коду гри

|  |  |
| --- | --- |
| Назва елемента | Опис елемента |
| Bullet.java | Інформація про кулі |
| Enemy.java | Інформація про ворогів |
| Explosion.java | Функціональність знищення ворога |
| Game.java | Загальні характеристики елементів гри |
| GamePanel.java | Елементи функціональності ігрового поля |
| Player.java | Функціональність гравця |
| Power.java | Інформація про збільшення потужності гравця |
| Text.java | Функціональність текстових надписів |

Підключення та наповнення пакетів гри

import javax.swing.JPanel;

import java.awt.\*;

import java.awt.image.\*;

import java.awt.event.\*;

import java.util.\*;

import javax.swing.JFrame;

import java.awt.\*;

public class Game {

public static void main(String[] args) {

JFrame window = new JFrame("Game Shoot");

window.setDefaultCloseOperation(JFrame.*EXIT\_ON\_CLOSE*);

window.setContentPane(new GamePanel());

window.pack();

window.setVisible(true);

}

}

Головні параметри классу :

public class Enemy {

// FIELDS

private double x;

private double y;

private int r;

private double dx;

private double dy;

private double rad;

private double speed;

private int health;

private int type;

private int rank;

private Color color1;

private boolean ready;

private boolean dead;

private boolean hit;

private long hitTimer;

private boolean slow;

Метод Draw, який власне б малював зображення ігрових елементів. Для цього до кожного з класів додамо наш конструктор

public void draw(Graphics2D g) {

g.setColor(color1);

g.fillOval(x - r, y - r, 2 \* r, 2 \* r);

g.setStroke(new BasicStroke(3));

g.setColor(color1.darker());

g.drawOval(x - r, y - r, 2 \* r, 2 \* r);

g.setStroke(new BasicStroke(1));

}

Реалізація взаємодії керування ігровим персонажем.

public void keyTyped(KeyEvent key) {}

public void keyPressed(KeyEvent key) {

int keyCode = key.getKeyCode();

if(keyCode == KeyEvent.*VK\_LEFT*) {

*player*.setLeft(true);

}

if(keyCode == KeyEvent.*VK\_RIGHT*) {

*player*.setRight(true);

}

if(keyCode == KeyEvent.*VK\_UP*) {

*player*.setUp(true);

}

if(keyCode == KeyEvent.*VK\_DOWN*) {

*player*.setDown(true);

}

if(keyCode == KeyEvent.*VK\_Z*) {

*player*.setFiring(true);

}

}

Формування рядка виводу інформації

g.setColor(new Color(0, 100, 255));

g.fillRect(0, 0, *WIDTH*, *HEIGHT*);

g.setColor(Color.*WHITE*);

g.setFont(new Font("Century Gothic", Font.*PLAIN*, 16));

s = "Final Score: " + *player*.getScore();

length = (int) g.getFontMetrics().getStringBounds(s, g).getWidth();

g.drawString(s, (*WIDTH* - length) / 2, *HEIGHT* / 2 + 30);

gameDraw();

## Умови виконання програми

Програма може працювати під управлінням операційної системи Windows 2000, а також інших, новіших, версіях ОС цього сімейства.

Вимогою до апаратного та програмного забезпечення є наявність будь-якого IBM-сумісного комп'ютера з наявністю накопичувача на гнучких або жорстких магнітних дисках і наявністю достатньої кількості оперативної пам'яті.

## Демонстрація роботи програми

Щоб розпочати гру, запустити Eclipse і перейти в проекти .

Щоб відкрити програму без середовища розробки знадобитися ПК з ОС, та установленною платформою Java.

 Для запуску програми необхідно запустити файл Shoot.jar, після чого на екрані з'явиться головне вікно програми. Управління грою здійснюється за допомогою клавіш, описаних в таблиці 4.1.

Завдання гравця - знищити 5 супротивників. Для їх знищення необхідно підібрати зброю - жовтий куб

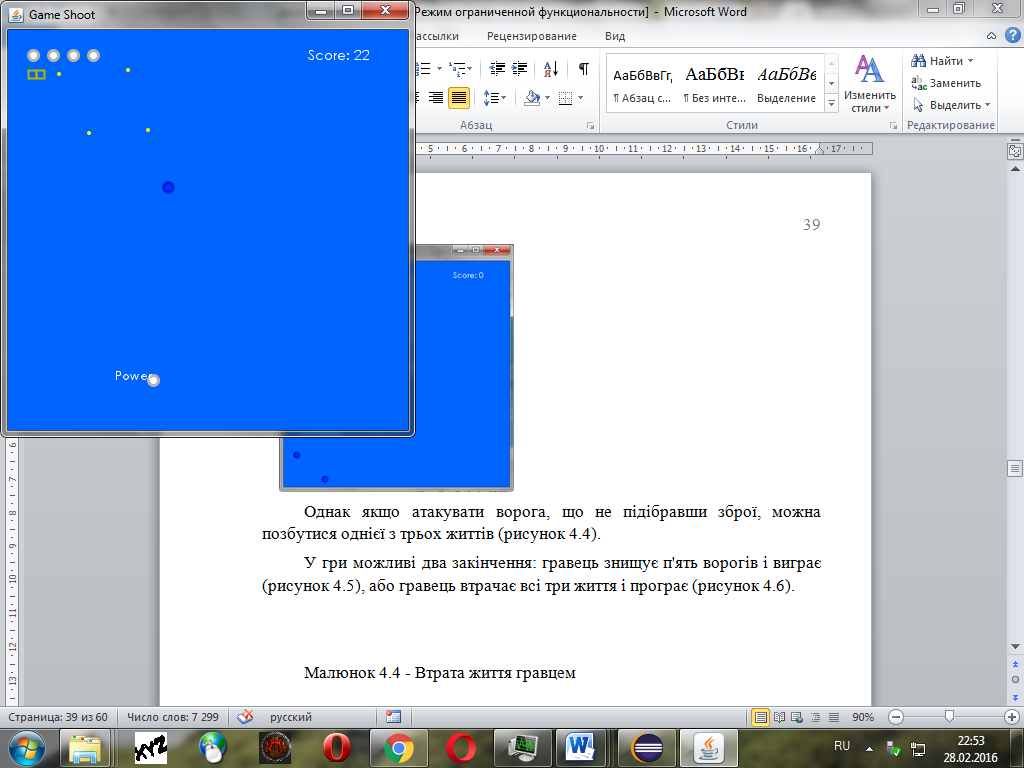


Рис 3.4 - Отримання бонусів гравцем

Однак якщо атакувати ворога, не підібравши бонусів у вигляди збільшення потужності куль та їх кількості, можна позбутися однієї з трьох життів (рисунок 4.4).

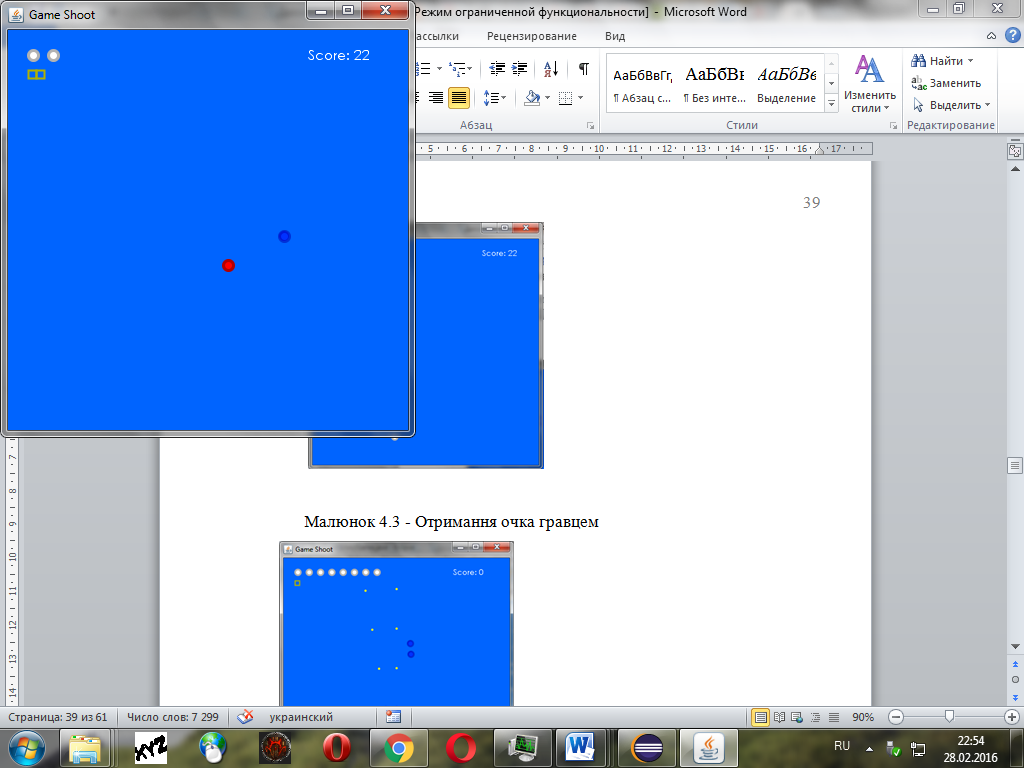
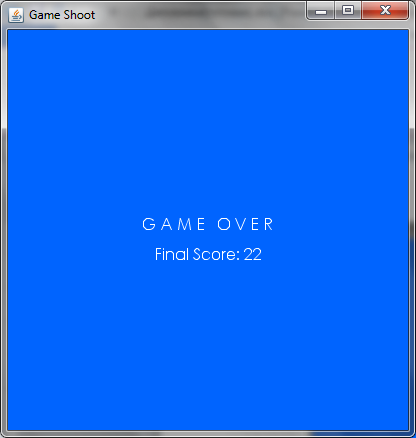


Рис 3.5 - Втрата життя гравцем

 У гри можливі два закінчення: гравець знищує усіх ворогів і виграє, або гравець втрачає всі три життя і програє (рисунок 4.5).

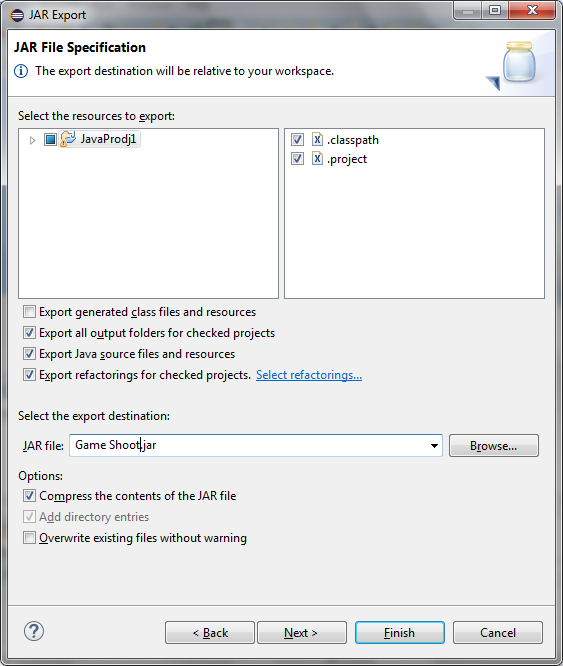


Зображення 3.6 - Кінець гри, гравець програв

Таблиця 4.1 - Призначення клавіш

|  |  |
| --- | --- |
| Назва клавіші | Виконувана функція |
| Стрілка вправо | Рух гравця вправо |
| Стрілка вліво | Рух гравця вліво |
| Стрілка вгору | Рух гравця вгору |
| Z | Вогонь |

Гра повністю функціональна та може запускатись через головним класом командну строку. Для надання біль зручного використання скомпілюємо усі файлі один. Насамперед виберемо потрібні опції та задамо назву



Зображення 3.7 Створення единного файлу гри

Перевіримо та запустимо програму як єдиний файл.

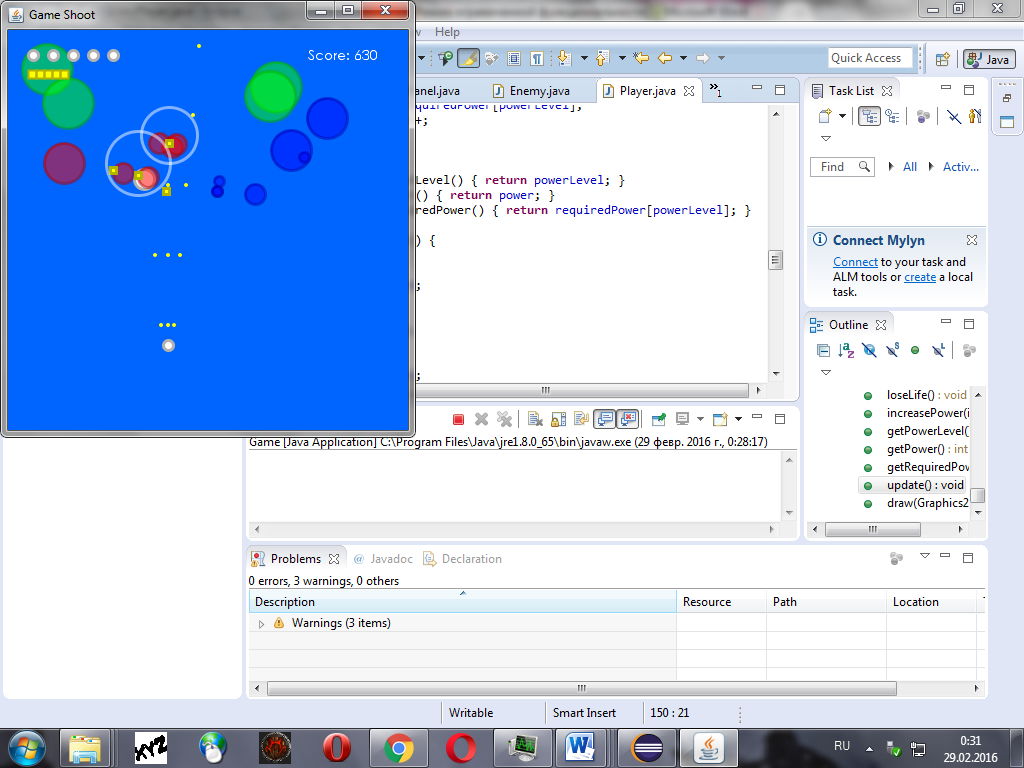


Рисунок 3.8 - Інтерфейс програми

## Критерій якості програми

1. Критерії якості з точки зору виконання критеріїв об'єктно-орієнтованого підходу:

- Розроблені об'єкти містять інкапсульовані дані і функції, згруповані разом, що дозволяє захистити дані;

- Програма відповідає призначенню, тобто, призначена для гри;

- Завершеність, тобто даний продукт має всі необхідні рисами, необхідними для гри.

- Програма в достатній мірі протестована, зроблені виправлення виявлених помилок.

- Програма має інтуїтивно зрозумілим і простим інтерфейсом;

- Програма супроводжена описом, яке допоможе користувачеві зрозуміти принципи її роботи.

- Даний програмний продукт володіє високою швидкодією і коротким часом відгуку;

- Програмний продукт має достатньою ефективністю за приладами, тобто економічністю використання пристроїв машини для вирішення поставленого завдання.

- Програма придатна до змін, тобто при появі нових вимог є можливість модифікування програмного коду;

- Програма має стабільність.

- Програма має гарну адаптованість і незалежністю від пристроїв (здатна працювати на різноманітному апаратному забезпеченні);

- Продукт не вимагає інсталяції, тобто здійснюється запуск .jar-файлу.

# РОЗРАХУНОК ЕКОНОМІЧНОГО ЕФЕКТУ ВІД РОЗРОБКИ І ПРОДАЖУ ПРОГРАМИ

## Визначення витрат на розробку програми

Витрати на розробку програми визначаються по формулі:

З = ФОТ + Отч + Накл + Зпевм., (4.1)

де ФОТ – фонд оплати праці, грн;

Отч – відрахування до позабюджетних фондів, грн;

Накл – накладні витрати, пов'язані із створенням програми, грн;

Зпевм – витрати, пов'язані з експлуатацією програми, грн.

З = 52,90 + 14,50,66 + 14,90+ 22 = 104,30 (грн).

Розмір фонду оплати праці розробника розраховується по формулі:

ФОТ = Зппрям × (1 + Кр / 100) (4.2)

де Зппрям – пряма заробітна плата, грн;

Кр – районний коефіцієнт % (Кр = 15 %).

ФОТ = 46 × (1+15/100)= 52,90(грн)

Зппрям=СР × Окл., (4.3)

де СР – повний термін розробки програми, міс. (днів);

Окл – оклад (денна тарифна ставка) розробника, грн.

Оклад = 40 / 20 = 2 (грн)

Зппрям = 22 × 2 = 44 (грн)×

Час, що реально витрачається розробником на створення системи приведено в табл. 4.1.

Таблиця 4.1

Терміни на розробку програми

|  |  |  |
| --- | --- | --- |
| Стадія розробки | Термін, відведений на стадію розробки (днів) | |
|  | Всього | В т.ч. машинного часу |
| 1. Передпроектна стадія | 2 |  |
| 2. Проектування | 4 | 3 |
| 3. Програмування | 14 | 16 |
| 4. Випробування | 3 | 3 |
| Разом: | 22 | 22 |

Розрахунок страхових внесків до позабюджетних фондів проводиться по формулі:

Отч = ФОТ × (Котч / 100), (4.4)

де Котч – відсоток відрахувань % (Котч = 26%);

Отч = 52,90 × (26 / 100) = 13,75 (грн).

Накладні витрати, пов'язані із створенням програми, складають 35% від ФОТ і розраховуються по формулі:

Накл = ФОТ × 0,35. (4.5)

Накл = 52,90 × 0,35 = 18,51 (грн)

Витрати пов'язані з експлуатацією ПЕВМ, розраховуються по формулі:

Зпевм = tмаш × Смаш.час (4.6)

де tмаш – повний машинний час, витрачений в процесі розробки програми;

Смаш. година – вартість машино-години, грн/година.

Зпевм = 23 × 1 = 23 (грн)

## Розрахунок вартості програми

Вартість розробленої програми визначається по формулі:

Ціна = З × (1 + Нормпр / 100) (4.7)

де Нормпр – норма прибутку %, (Нормпр = 25 %).

Ціна = 104,30× (1 + 25 / 100) = 130,40 (грн)

## Розрахунок економічного ефекту від продажу програми

Економічний ефект від продажу програми визначається за формулою:

ЕФ = Кіл × Ціна – Кіл × З – Кіл × Ціна Зрекл / 100 (4.8)

де Зрекл – витрати на рекламу % на одиницю проданої продукції (Зрекл = 1 %).

Кіл – гадана кількість проданих одиниць продукту, шт/рік.

ЕФ = 100 × 130,40 – 100 × 104,30 – 100 × 130,40 × 1 / 100 = 1867,1(грн).Таким чином економічний ефект від продажу розробленої програми складає 1867,1 гривень на рік.

# 5 ОХОРОНА ПРАЦІ КОРИСТУВАЧІВ ПК

Однією із характерних особливостей сучасного розвитку суспільства є зростання сфер діяльності людини, в яких використовуються інформаційні технології. Широке розповсюдження отримали персональні комп'ютери. Однак їх використання загострило проблеми збереження власного та суспільного здоров'я, вимагає вдосконалення існуючих та розробки нових підходів до організації робочих місць, проведення профілактичних заходів для запобігання розвитку негативних наслідків впливу ПК на здоров'я користувачів [1, c. 232].

Зараз у нашій країні проводиться розробка національних нормативних документів, спрямованих на охорону праці користувачів ПК. Найбільш повним нормативним документам щодо забезпечення охорони пращ користувачів ПК є «Державні санітарні правила й норми роботи з візуальними дисплейними терміналами (ВДТ) електронно-обчислювальних машин» ДСанШН 3.3.2.007–98.

## Вимоги до виробничих приміщень для експлуатації ПК

Об'ємно-планувальні рішення будівель та приміщень для роботи з ПК мають відповідати вимогам ДСанПіН 3.3.2.007–98.

Розміщення робочих місць з ПК у підвальних приміщеннях, на цокольних поверхах заборонено.

Площа на одне робоче місце становить не менше ніж 6,0 м3, а об'єм – не менше ніж 20,0 м3.

Приміщення для роботи з ПК повинні мати природне та штучне освітлення відповідно до СНиП П-4–79/

Природне освітлення має здійснюватись через світлові прорізи, орієнтовані переважно на північ чи північний схід, і забезпечувати коефіцієнт природної освітленості (КПО) не нижче, ніж 1,5%.

Виробничі приміщення повинні обладнуватись шафами для зберігання документів, магнітних дисків, полицями, стелажами, тумбами тощо, з урахуванням вимог до площі приміщень.

У приміщеннях з ПК слід щоденно робити вологе прибирання.

Приміщення із ПК мають бути оснащені аптечками першої медичної допомоги.

При приміщеннях із ПК мають бути обладнані побутові приміщення для відпочинку під час роботи, кімната психологічного розвантаження. В кімнаті психологічного розвантаження слід передбачити встановлення пристроїв для приготування й роздачі тонізуючих напоїв, а також місця для занять фізичною культурою (СНиП 2.09.04. – 87) [2, c. 17].

## Мікроклімат виробничих приміщень

Складові мікроклімату виробничих приміщень та їх вплив на працівника. На підприємствах на самопочуття, стан здоров’я людини впливає мікроклімат виробничих приміщень, який визначається дією на організм людини температури, вологості, рухомості повітря і теплового випромінювання.

Виробничий мікроклімат, як правило, відрізняється значною мінливістю, нерівномірністю по горизонталі та вертикалі, різноманітністю сполучень температури, вологості, рухомості повітря, інтенсивності випромінювання залежно від особливостей технології виробництва, кліматичних особливостей місцевості, конструкцій споруд, організації повітрообміну із зовнішнім середовищем.

Мікроклімат виробничих приміщень - умови внутрішнього середовища цих приміщень, що впливають на тепловий обмін працюючих з оточенням шляхом конвекції, кондукції, теплового випромінювання та випаровування вологи.

Мікроклімат виробничих приміщень характеризується температурою, вологістю повітря, швидкістю переміщення повітряних мас, а також тепловим випромінюванням від нагрітих обладнання, машин, предметів праці. Від комплексного впливу цих елементів залежать теплові відчуття і зумовлені ними фізіологічні та психічні стани працівників.

Визначальним метеорологічним елементом є температура повітря, дія якої може посилюватися або послаблюватися іншими факторами. Посилення несприятливого впливу одного фактора дією інших факторів характеризується як їхня взаємодія. При антагоністичній взаємодії несприятливий вплив одного фактора послаблюється іншим, що діє в цей час.

Мікроклімат виробничих приміщень зумовлюється технологіч­ним процесом і певною мірою зовнішніми метеорологічними умовами.

Цехи, в яких тепловиділення від обладнання, матеріалів, людей, сонця перевищують 20 ккал на 1 м3 за годину, відносяться до гарячих. Як правило, основними при цьому є теплові випромінювання від нагрітих поверхонь обладнання і матеріалів.

Так, 500-тонна мартенівська піч віддає 15 млн ккал/год, з яких 10 млн ккал припадає на випромінювання. Теплове навантаження складає 250—300 ккал на 1 м3 за годину. Біля 100 ккал на 1 м3 за годину становлять тепловиділення в сушильних цехах заводів будівельних матеріалів, на деяких дільницях виробництва штучного волокна, капрону, в легкій, нафтопереробній, харчовій промисловості. Виконання роботи в умовах теплового випромінювання і високих температур викликає різко виражені фізіологічні зрушення в організмі працюючих.

Таблиця. 5.1

Оптимальні відчуття залежно від температури і вологості повітря

|  |  |  |
| --- | --- | --- |
| Температура °С | Відносна вологість повітря,  % | Стан |
| 21 | 40 | Найбільш приємний стан |
|  | 75 | Відсутні неприємні відчуття |
| 91 | Втома, пригнічений стан |
| 24 | 20 | Неприємні відчуття |
|  | 65 | Потреба у відпочинку |
| 80 | Втома, пригнічений стан |
|  | 100 | Неможливе виконання важкої роботи |
| 34 | 25 | Відсутні неприємні відчуття |

Їхня працездатність в таких умовах знижується на 50 %. Фізіологами встановлено, що температура 22°С є тією межею, за якою починається прогресивне зниження працездатності. Так, при підвищенні температури до 26°С вона зменшується на 4 % з кожним градусом, а при подальшому її підвищенні до 30°С — на 6 %. А. В. Васильєвою виділені три температурних режими: від 0 до 25°С; від 25 до 35°С і від 35 до 50°С.

Кожному режиму властивий певний характер зрушень фізіологічних функцій працівника. При першому режимі ці зрушення знаходяться в зоні допустимих показників; при другому відмічаються підвищені показники, а при третьому — парадоксальні реакції. В умовах, коли температура повітря дорівнює або перевищує температуру тіла працівника, віддача тепла організмом відбувається шляхом випаровування вологи.

Так, при виконанні важкої фізичної роботи при високій температурі повітря кількість виділеного поту може сягати 1,0 – 1,5 л/год. Тепловий удар супроводжується раптовою втратою свідомості, підвищенням температури тіла до 40 – 41°С, слабим частим пульсом, припиненням потовиділення. Судомна хвороба характеризується порушенням водно-солевого обміну, судомами м’язів, кінцівок, діафрагми, потовиділенням, згущенням крові. Теплова рівновага працівника залежить також від вологості повітря, тобто вмісту у повітрі водяних парів. Найсприятливішою для організму є відносна вологість повітря від 35 до 60 %.

Таблиця. 5.2

Оптимальні норми в робочій зоні виробничих приміщень

|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| Категорія роботи | Холодний і перехідний періоди року (температура зовнішнього повітря нижча за +10°С) | | | Теплий період року (температура зовнішнього повітря вища за +10°С) | | |
|  | Температура  повітря, °С | Відносна вологість повітря,  % | Швидкість руху повітря, м/с | Температура  повітря, °С | Відносна вологість повітря,  % | Швидкість руху повітря, м/с |
| Легка | 20-22 | 60-30 | не більш як 0,2 | 22-25 | 60-30 | 0,2-0,5 |
| Середньої важкості | 17-19 | 60-30 | не більш як 0,3 | 20-23 | 60-30 | 0,2-0,5 |
| Важка | 16-18 | 60-30 | не більш як 0,3 | 18-21 | 60-30 | 0,3-0,7 |

Якщо вологість менша 35 %, то повітряне середовище характеризується сухістю, яка посилює випаровування води з поверхні шкіри. При підвищенні вологості повітря (понад 60 %) випаровування поту утруднене.

Так, при температурі 25°С в умовах дуже сухого повітря організм втрачає через шкіру і легені 75,4 г вологи за годину, а в умовах дуже вологого повітря – лише 23,9 г/год. Для оцінки комфортності умов праці залежно від температури і вологості повітря використовується показник ефективних температур. Ефективною вважається температура, яку відчуває людина при певній вологості повітря і відсутності його руху. Рух повітря у приміщенні також справляє різний вплив на організм працівника, посилюючи або послаблюючи дію інших метеорологічних факторів.

Так, при високій температурі і високій вологості повітря сприятливішою є вища швидкість руху повітря порівняно з комбінацією високої температури і низькою вологістю повітря. Для оцінки комфортності умов праці залежно від температури, вологості і руху повітря використовується показник ефективно-еквівалентної температури.

Ефективно-еквівалентною вважається температура, яка відчувається людиною при певній відносній вологості і швидкості руху повітря. Заходами, які забезпечують створення оптимальних мікрокліматичних умов на виробництві, є:

* механізація важких робіт у гарячих цехах;
* застосування дистанційного управління тепловипромінювальними процесами і апаратами;
* теплоізоляція гарячих поверхонь обладнання;
* застосування теплових повітряних завіс на вході до виробничих приміщень;
* вентиляція і кондиціювання повітря, регулювання вологості повітря.

## Гігієнічні вимоги до організації та обладнання робочих місць

Обладнання й організація робочого місця із БДТ мають забезпечувати відповідність конструкції всіх елементів робочого місця та їх взаємного розташування ергономічним вимогам з урахуванням характеру й особливостей трудової діяльності (ГОСТ 12,2.032–78, ГОСТ22.269–76, ГОСТ 21.889–76) [3, c. 177].

Конструкція робочого місця користувача ПК має забезпечити підтримання оптимальної робочої пози.

Робочі місця із ПК слід так розташовувати відносно світлових прорізів, щоб природне світло падало збоку, переважно зліва.

При розміщенні робочих столів із ПК слід дотримуватись таких відстаней: між бічними поверхнями БДТ – 1,2 м; від тильної поверхні одного ПК до екрана іншого – 2,5 м.

Екран ПК має розташовуватися на оптимальній відстані тещ очей користувача, що становить 600…700 мм, але не ближче ніж за 600 мм з урахуванням розміру літерно-цифрових знаків і символів.

Розташування екрана ПК має забезпечувати зручність зорового спостереження у вертикальній площині під кутом +30\* до нормальної лінії погляду працюючого.

Клавіатуру слід розташовувати на поверхні столу на відстані 100…300 мм від краю, звернутого до працюючого. У конструкції клавіатури має передбачатися опорний пристрій (виготовлений із матеріалу з високим коефіцієнтом тертя, що перешкоджає мимовільному її зсуву), який дає змогу змінювати кут нахилу поверхні клавіатури в межах 5… 15°.

Для забезпечення захисту і досягнення нормованих рівнів комп'ютерних випромінювань необхідно застосовувати при екранні фільтри, локальні світлофільтри (засоби індивідуального захисту очей) та інші засоби захисту, що пройшли випробування в акредитованих лабораторіях і мають щорічний гігієнічний сертифікат [3, c. 175].

При оснащенні робочого місця із ВДТ лазерним принтером параметри лазерного випромінювання повинні відповідати вимогам ДСанПіН 3.3.2.007–98.

## Вимоги до режимів праці та відпочинку при роботі з ПК

При організації праці, пов'язаної з використанням ПК, для збереження здоров'я працюючих, запобігання професійним захворюванням І підтримки працездатності передбачаються внутрішньо змінні регламентовані перерви для відпочинку [4, c. 220].

Внутрішньозмінні режими праці й відпочинку містять додаткові нетривалі перерви в періоди, що передують появі об'єктивних і суб'єктивних ознак стомлення й зниження працездатності.

При виконанні робіт, що належать до різних видів трудової діяльності, за основну роботу з ПК слід вважати таку, що займає не менше 50% робочого часу. Впродовж робочої зміни мають передбачатися:

·    перерви для відпочинку і вживання їжі (обідні перерви);

·    перерви для відпочинку й особистих потреб (згідно із трудовими нормами);

·    додаткові перерви, що вводяться для окремих професій з урахуванням особливостей трудової діяльності.

За характером трудової діяльності розрізняють три професійні групи, згідно з діючим класифікатором професій [4, c. 223–224]:

1) розробники програм інженери-програмісти) виконують роботу переважно з відеотерміналом та документацією при необхідності інтенсивного обміну Інформацією з ЕОМ і високою частотою прийняття рішень. Робота характеризується інтенсивною розумовою творчою працею з підвищеним напруженням зору, концентрацією уваги на фоні нервово-емоційного напруження, вимушеною робочою позою, загальною гіподинамією, періодичним навантаженням на кисті верхніх кінцівок. Робота виконується в режимі діалогу з ПК у вільному темпі з періодичним пошуком помилок в умовах дефіциту часу;

2) оператори електронно-обчислювальних машин виконують роботу, пов'язану з обліком інформації, одержаної із ВДТ за попереднім запитом, або тієї, що надходить з нього, супроводжується перервами різної тривалості, пов'язана з виконанням іншої роботи й характеризується напруженням зору, невеликими фізичними зусиллями, нервовим напруженням середнього ступеня та виконується у вільному темпі;

3) оператор комп'ютерного набору виконує одноманітні за характером роботи з документацією та клавіатурою і нечастими нетривалими переключеннями погляду на екран дисплея, з введенням даних з високою швидкістю. Робота характеризується як фізична праця з підвищеним навантаженням на кисті верхніх кінцівок на фоні загальної гіподинамії з напруженням зору (фіксація зору переважно на документи), нервово-емоційним напруженням.

Правилами встановлюються такі внутрішньозмінні режими праці та відпочинку при роботі з ПК при 8-годинній денній робочій зміні в залежності від характеру праці:

·      для розробників програм із застосуванням ПК слід призначати регламентовану перерву для відпочинку тривалістю 15 хвилин через кожну годину роботи за ПК;

·       для операторів із застосуванням ПК слід призначати регламентовані перерви для відпочинку тривалістю 15 хвилин через кожні дві години;

·      для операторів комп'ютерного набору слід призначати регламентовані перерви для відпочинку тривалістю 10 хвилин після кожної години роботи за ПК.

У всіх випадках, коли виробничі обставини не дозволяють застосувати регламентовані перерви, тривалість безперервної роботи з ПК не повинна перевищувати 4 години.

При 12-годинній робочій зміні регламентовані перерви повинні встановлюватися в перші 8 годин робота аналогічно перервам при 8-годинній робочій зміні, а протягом останніх 4-х годин роботи, незалежно від характеру трудової діяльності, через кожну годину тривалістю 15 хвилин.

Для зниження нервово-емоційного напруження, втомлення зорового аналізатора, поліпшення мозкового кровообігу, подолання несприятливих наслідків гіподинамії, запобігання втомі доцільно деякі перерви використовувати для виконання комплексу вправ, які наведені у Державних санітарних правилах і нормах роботи з візуальними дисплейними терміналами електронно-обчислювальних машин ДСаяПІН 3.3.2.007–98 [4, c. 226].

## Вимоги до профілактичних медичних оглядів

Працюючі з ПК підлягають обов'язковим медичним оглядам: попереднім – при влаштуванні на роботу і періодичним – протягом трудової діяльності, відповідно до наказу МЗ України N45 від 31.03.94 р.

Періодичні методичні огляди мають проводитися раз на два роки комісією в складі терапевта, невропатолога та офтальмолога.

До складу комісії, що проводить попередні та періодичні медичні огляди, при необхідності (за наявністю медичних показань), можуть залучатись до оглядів лікарі інших спеціальностей.

Основними критеріями оцінки придатності до роботи з ПК мають бути показники стану органів зору: гострота зору, показники рефракції, акомодації, стану бінокулярного апарату ока тощо. При цьому необхідно враховувати також стан організму в цілому [1, c. 239].

Жінки, що працюють з ВДТ, обов'язково оглядаються акушером-гінекологом один раз на два роки.

Жінки з часу встановлення вагітності та в період годування дитини грудьми до виконання всіх робіт, пов'язаних з використанням ПК, не допускаються.

Виконання вимог, наведених в Правилах, в комплексі з практичним здійсненням первинних та спеціальних заходів повинно стати нормою діяльності всіх фахівців, безпосередньо пов'язаних з навчальними та виробничими колективами.

# ВИСНОВКИ

Програма розроблена відповідно до постановкою завдання дипломного проектування з дисципліни «Об'єктно-орієнтоване програмування». При написанні були використані методичні вказівки по курсовому проектування з дисципліни «Об'єктно-орієнтоване програмування». Інтерфейс програми простий і зручний в користуванні. Тестування програми підтвердило, що програма коректно виконує взаємодія об'єктів різних класів відповідно до варіанта.

Застосування розроблено за допомогою мови програмування Java та середовища розробки Eclpse. Працює на комп'ютері з будь-якою ОС, на якій встановлена JAVA з вереї не менше 1.7

# СПИСОК ВИКОРИСТАНИХ ДЖЕРЕЛ

* + 1. Статті про програмуванні для [Електронний ресурс] / Режим доступу: URL http://flashbot.ru/ Java -dev
    2. Офіційна довідка середовищі програмування [Електронний ресурс] / Режим доступу: http://www.jetbrains.com
    3. Форум про програмування [Електронний ресурс] / Режим доступу: http://www.cyberforum.ru/ Java -dev/
    4. Підручник, уроки для початківців [Електронний ресурс] / Режим доступу:  http://startandroid.ru/ru/
    5. Хашими С. Разработка приложений для Java / Хашими С, Коматинени С, Маклин Д. — Пітер, 2011. — 265с.
    6. Майер Р. Программирование приложений для планшетных компьютеров и смартфонов / Майер Р. — Эксмо, 2011. — 671с.
    7. Khalid A. Mughal - A Programmer’s Guide to Java™ SCJP Certification - Addison-Wesley -2009. – 1089с.
    8. Michael Sicor-EJB 3 Developer Guide -Packt publishing -2008. -277с.
    9. Вязовик Н. А. Програмування на Java. Курс лекцій. - М. Інтернет-університет інформаційних технологій. -2003. -592с.
    10. Карабін П. Мова програмування Java: Створення інтерактивних додатків для Internet. - М. Прес-прес-бук-прес. -2006. -224с.
    11. Знакомьтесь: Java. Самоучитель -Аккуратов Е. Е. -2006. -230с.
    12. Применение шаблонов Java. - Стивен Стелтинг, Олав Маасен -2002. -576с.

# ДОДАТОК А

GamePanel.java

import javax.swing.JPanel;

import java.awt.\*;

import java.awt.image.\*;

import java.awt.event.\*;

import java.util.\*;

public class GamePanel extends JPanel implements Runnable, KeyListener {

// FIELDS

public static int WIDTH = 400;

public static int HEIGHT = 400;

private Thread thread;

private boolean running;

private BufferedImage image;

private Graphics2D g;

private int FPS = 30;

private double averageFPS;

public static Player player;

public static ArrayList<Bullet> bullets;

public static ArrayList<Enemy> enemies;

public static ArrayList<PowerUp> powerups;

public static ArrayList<Explosion> explosions;

public static ArrayList<Text> texts;

private long waveStartTimer;

private long waveStartTimerDiff;

private int waveNumber;

private boolean waveStart;

private int waveDelay = 2000;

private long slowDownTimer;

private long slowDownTimerDiff;

private int slowDownLength = 6000;

// CONSTRUCTOR

public GamePanel() {

// super();

setPreferredSize(new Dimension(WIDTH, HEIGHT));

setFocusable(true);

requestFocus();

}

// FUNCTIONS

public void addNotify() {

super.addNotify();

if (thread == null) {

thread = new Thread(this);

thread.start();

}

addKeyListener(this);

}

public void run() {

running = true;

image = new BufferedImage(WIDTH, HEIGHT, BufferedImage.TYPE\_INT\_RGB);

g = (Graphics2D) image.getGraphics();

g.setRenderingHint(RenderingHints.KEY\_ANTIALIASING, RenderingHints.VALUE\_ANTIALIAS\_ON);

g.setRenderingHint(RenderingHints.KEY\_TEXT\_ANTIALIASING, RenderingHints.VALUE\_TEXT\_ANTIALIAS\_ON);

player = new Player();

bullets = new ArrayList<Bullet>();

enemies = new ArrayList<Enemy>();

powerups = new ArrayList<PowerUp>();

explosions = new ArrayList<Explosion>();

texts = new ArrayList<Text>();

waveStartTimer = 0;

waveStartTimerDiff = 0;

waveStart = true;

waveNumber = 0;

long startTime;

long URDTimeMillis;

long waitTime;

long totalTime = 0;

int frameCount = 0;

int maxFrameCount = 30;

long targetTime = 1000 / FPS;

// hue = 0;

// GAME LOOP

while (running) {

startTime = System.nanoTime();

gameUpdate();

gameRender();

gameDraw();

URDTimeMillis = (System.nanoTime() - startTime) / 1000000;

waitTime = targetTime - URDTimeMillis;

try {

Thread.sleep(waitTime);

} catch (Exception e) {

}

frameCount++;

if (frameCount == maxFrameCount) {

averageFPS = 1000.0 / ((totalTime / frameCount) / 1000000);

frameCount = 0;

totalTime = 0;

}

}

g.setColor(new Color(0, 100, 255));

g.fillRect(0, 0, WIDTH, HEIGHT);

g.setColor(Color.WHITE);

g.setFont(new Font("Century Gothic", Font.PLAIN, 16));

String s = "G A M E O V E R";

int length = (int) g.getFontMetrics().getStringBounds(s, g).getWidth();

g.drawString(s, (WIDTH - length) / 2, HEIGHT / 2);

s = "Final Score: " + player.getScore();

length = (int) g.getFontMetrics().getStringBounds(s, g).getWidth();

g.drawString(s, (WIDTH - length) / 2, HEIGHT / 2 + 30);

gameDraw();

}

private void gameUpdate() {

// new wave

if (waveStartTimer == 0 && enemies.size() == 0) {

waveNumber++;

waveStart = false;

waveStartTimer = System.nanoTime();

} else {

waveStartTimerDiff = (System.nanoTime() - waveStartTimer) / 1000000;

if (waveStartTimerDiff > waveDelay) {

waveStart = true;

waveStartTimer = 0;

waveStartTimerDiff = 0;

}

}

// create enemies

if (waveStart && enemies.size() == 0) {

createNewEnemies();

}

// player update

player.update();

// bullet update

for (int i = 0; i < bullets.size(); i++) {

boolean remove = bullets.get(i).update();

if (remove) {

bullets.remove(i);

i--;

}

}

// enemy update

for (int i = 0; i < enemies.size(); i++) {

enemies.get(i).update();

}

// powerup update

for (int i = 0; i < powerups.size(); i++) {

boolean remove = powerups.get(i).update();

if (remove) {

powerups.remove(i);

i--;

}

}

// explosion update

for (int i = 0; i < explosions.size(); i++) {

boolean remove = explosions.get(i).update();

if (remove) {

explosions.remove(i);

i--;

}

}

// text update

for (int i = 0; i < texts.size(); i++) {

boolean remove = texts.get(i).update();

if (remove) {

texts.remove(i);

i--;

}

}

// bullet-enemy collision

for (int i = 0; i < bullets.size(); i++) {

Bullet b = bullets.get(i);

double bx = b.getx();

double by = b.gety();

double br = b.getr();

for (int j = 0; j < enemies.size(); j++) {

Enemy e = enemies.get(j);

double ex = e.getx();

double ey = e.gety();

double er = e.getr();

double dx = bx - ex;

double dy = by - ey;

double dist = Math.sqrt(dx \* dx + dy \* dy);

if (dist < br + er) {

e.hit();

bullets.remove(i);

i--;

break;

}

}

}

// check dead enemies

for (int i = 0; i < enemies.size(); i++) {

if (enemies.get(i).isDead()) {

Enemy e = enemies.get(i);

// chance for powerup

double rand = Math.random();

if (rand < 0.001)

powerups.add(new PowerUp(1, e.getx(), e.gety()));

else if (rand < 0.020)

powerups.add(new PowerUp(3, e.getx(), e.gety()));

else if (rand < 0.120)

powerups.add(new PowerUp(2, e.getx(), e.gety()));

else if (rand < 0.130)

powerups.add(new PowerUp(4, e.getx(), e.gety()));

player.addScore(e.getType() + e.getRank());

enemies.remove(i);

i--;

e.explode();

explosions.add(new Explosion(e.getx(), e.gety(), e.getr(), e.getr() + 30));

}

}

// check dead player

if (player.isDead()) {

// running = false;

}

// player-enemy collision

if (!player.isRecovering()) {

int px = player.getx();

int py = player.gety();

int pr = player.getr();

for (int i = 0; i < enemies.size(); i++) {

Enemy e = enemies.get(i);

double ex = e.getx();

double ey = e.gety();

double er = e.getr();

double dx = px - ex;

double dy = py - ey;

double dist = Math.sqrt(dx \* dx + dy \* dy);

if (dist < pr + er) {

player.loseLife();

}

}

}

// player-powerup collision

int px = player.getx();

int py = player.gety();

int pr = player.getr();

for (int i = 0; i < powerups.size(); i++) {

PowerUp p = powerups.get(i);

double x = p.getx();

double y = p.gety();

double r = p.getr();

double dx = px - x;

double dy = py - y;

double dist = Math.sqrt(dx \* dx + dy \* dy);

// collected powerup

if (dist < pr + r) {

int type = p.getType();

if (type == 1) {

player.gainLife();

texts.add(new Text(player.getx(), player.gety(), 2000, "Extra Life"));

}

if (type == 2) {

player.increasePower(1);

texts.add(new Text(player.getx(), player.gety(), 2000, "Power"));

}

if (type == 3) {

player.increasePower(2);

texts.add(new Text(player.getx(), player.gety(), 2000, "Double Power"));

}

if (type == 4) {

slowDownTimer = System.nanoTime();

for (int j = 0; j < enemies.size(); j++) {

enemies.get(j).setSlow(true);

}

texts.add(new Text(player.getx(), player.gety(), 2000, "Slow Down"));

}

powerups.remove(i);

i--;

}

}

// slowdown update

if (slowDownTimer != 0) {

slowDownTimerDiff = (System.nanoTime() - slowDownTimer) / 1000000;

if (slowDownTimerDiff > slowDownLength) {

slowDownTimer = 0;

for (int j = 0; j < enemies.size(); j++) {

enemies.get(j).setSlow(false);

}

}

}

}

private void gameRender() {

// draw background

g.setColor(new Color(0, 100, 255));

g.fillRect(0, 0, WIDTH, HEIGHT);

// draw slowdown screen

if (slowDownTimer != 0) {

g.setColor(new Color(255, 255, 255, 64));

g.fillRect(0, 0, WIDTH, HEIGHT);

}

// draw player

player.draw(g);

// draw bullet

for (int i = 0; i < bullets.size(); i++) {

bullets.get(i).draw(g);

}

// draw enemy

for (int i = 0; i < enemies.size(); i++) {

enemies.get(i).draw(g);

}

// draw powerups

for (int i = 0; i < powerups.size(); i++) {

powerups.get(i).draw(g);

}

// draw explosions

for (int i = 0; i < explosions.size(); i++) {

explosions.get(i).draw(g);

}

// draw text

for (int i = 0; i < texts.size(); i++) {

texts.get(i).draw(g);

}

// draw wave number

if (waveStartTimer != 0) {

g.setFont(new Font("Century Gothic", Font.PLAIN, 18));

String s = "- W A V E " + waveNumber + " -";

int length = (int) g.getFontMetrics().getStringBounds(s, g).getWidth();

int alpha = (int) (255 \* Math.sin(3.14 \* waveStartTimerDiff / waveDelay));

if (alpha > 255)

alpha = 255;

g.setColor(new Color(255, 255, 255, alpha));

g.drawString(s, WIDTH / 2 - length / 2, HEIGHT / 2);

}

// draw player lives

for (int i = 0; i < player.getLives(); i++) {

g.setColor(Color.WHITE);

g.fillOval(20 + (20 \* i), 20, player.getr() \* 2, player.getr() \* 2);

g.setStroke(new BasicStroke(3));

g.setColor(Color.WHITE.darker());

g.drawOval(20 + (20 \* i), 20, player.getr() \* 2, player.getr() \* 2);

g.setStroke(new BasicStroke(1));

}

// draw player power

g.setColor(Color.YELLOW);

g.fillRect(20, 40, player.getPower() \* 8, 8);

g.setColor(Color.YELLOW.darker());

g.setStroke(new BasicStroke(2));

for (int i = 0; i < player.getRequiredPower(); i++) {

g.drawRect(20 + 8 \* i, 40, 8, 8);

}

g.setStroke(new BasicStroke(1));

// draw player score

g.setColor(Color.WHITE);

g.setFont(new Font("Century Gothic", Font.PLAIN, 14));

g.drawString("Score: " + player.getScore(), WIDTH - 100, 30);

// draw slowdown meter

if (slowDownTimer != 0) {

g.setColor(Color.WHITE);

g.drawRect(20, 60, 100, 8);

g.fillRect(20, 60, (int) (100 - 100.0 \* slowDownTimerDiff / slowDownLength), 8);

}

}

private void gameDraw() {

Graphics g2 = this.getGraphics();

g2.drawImage(image, 0, 0, null);

g2.dispose();

}

private void createNewEnemies() {

enemies.clear();

Enemy e;

if (waveNumber == 1) {

for (int i = 0; i < 4; i++) {

enemies.add(new Enemy(1, 1));

}

}

if (waveNumber == 2) {

for (int i = 0; i < 8; i++) {

enemies.add(new Enemy(1, 1));

}

}

if (waveNumber == 3) {

for (int i = 0; i < 4; i++) {

enemies.add(new Enemy(1, 1));

}

enemies.add(new Enemy(1, 2));

enemies.add(new Enemy(1, 2));

}

if (waveNumber == 4) {

enemies.add(new Enemy(1, 3));

enemies.add(new Enemy(1, 4));

for (int i = 0; i < 4; i++) {

enemies.add(new Enemy(2, 1));

}

}

if (waveNumber == 5) {

enemies.add(new Enemy(1, 4));

enemies.add(new Enemy(1, 3));

enemies.add(new Enemy(2, 3));

}

if (waveNumber == 6) {

enemies.add(new Enemy(1, 3));

for (int i = 0; i < 4; i++) {

enemies.add(new Enemy(2, 1));

enemies.add(new Enemy(3, 1));

}

}

if (waveNumber == 7) {

enemies.add(new Enemy(1, 3));

enemies.add(new Enemy(2, 3));

enemies.add(new Enemy(3, 3));

}

if (waveNumber == 8) {

enemies.add(new Enemy(1, 4));

enemies.add(new Enemy(2, 4));

enemies.add(new Enemy(3, 4));

}

if (waveNumber == 9) {

running = false;

}

}

public void keyTyped(KeyEvent key) {

}

public void keyPressed(KeyEvent key) {

int keyCode = key.getKeyCode();

if (keyCode == KeyEvent.VK\_LEFT) {

player.setLeft(true);

}

if (keyCode == KeyEvent.VK\_RIGHT) {

player.setRight(true);

}

if (keyCode == KeyEvent.VK\_UP) {

player.setUp(true);

}

if (keyCode == KeyEvent.VK\_DOWN) {

player.setDown(true);

}

if (keyCode == KeyEvent.VK\_Z) {

player.setFiring(true);

}

}

public void keyReleased(KeyEvent key) {

int keyCode = key.getKeyCode();

if (keyCode == KeyEvent.VK\_LEFT) {

player.setLeft(false);

}

if (keyCode == KeyEvent.VK\_RIGHT) {

player.setRight(false);

}

if (keyCode == KeyEvent.VK\_UP) {

player.setUp(false);

}

if (keyCode == KeyEvent.VK\_DOWN) {

player.setDown(false);

}

if (keyCode == KeyEvent.VK\_Z) {

player.setFiring(false);

}

}

}

Game.java

**import** javax.swing.JFrame;

//import java.awt.\*;

**public** **class** Game {

**public** **static** **void** main(String[] args) {

JFrame window = **new** JFrame("Game Shoot");

window.setDefaultCloseOperation(JFrame.***EXIT\_ON\_CLOSE***);

window.setContentPane(**new** GamePanel());

window.pack();

window.setVisible(**true**);

}

}

Bullet.java

**import** java.awt.\*;

**public** **class** Bullet {

// FIELDS

**private** **double** x;

**private** **double** y;

**private** **int** r;

**private** **double** dx;

**private** **double** dy;

**private** **double** rad;

**private** **double** speed;

**private** Color color1;

// CONSTRUCTOR

**public** Bullet(**double** angle, **int** x, **int** y) {

**this**.x = x;

**this**.y = y;

r = 2;

rad = Math.*toRadians*(angle);

speed = 10;

dx = Math.*cos*(rad) \* speed;

dy = Math.*sin*(rad) \* speed;

color1 = Color.***YELLOW***;

}

// FUNCTIONS

**public** **double** getx() { **return** x; }

**public** **double** gety() { **return** y; }

**public** **double** getr() { **return** r; }

**public** **boolean** update() {

x += dx;

y += dy;

**if**(x < -r || x > GamePanel.*WIDTH* + r ||

y < -r || y > GamePanel.*HEIGHT* + r) {

**return** **true**;

}

**return** **false**;

}

**public** **void** draw(Graphics2D g) {

g.setColor(color1);

g.fillOval((**int**) (x - r), (**int**) (y - r), 2 \* r, 5 \* r);

}

}

Enemy.java

**import** java.awt.\*;

**public** **class** Enemy {

// FIELDS

**private** **double** x;

**private** **double** y;

**private** **int** r;

**private** **double** dx;

**private** **double** dy;

**private** **double** rad;

**private** **double** speed;

**private** **int** health;

**private** **int** type;

**private** **int** rank;

**private** Color color1;

**private** **boolean** ready;

**private** **boolean** dead;

**private** **boolean** hit;

**private** **long** hitTimer;

**private** **boolean** slow;

// CONSTRUCTOR

**public** Enemy(**int** type, **int** rank) {

**this**.type = type;

**this**.rank = rank;

// default enemy

**if**(type == 1) {

//color1 = Color.BLUE;

color1 = **new** Color(0, 0, 255, 128);

**if**(rank == 1) {

speed = 5;

r = 5;

health = 1;

}

**if**(rank == 2) {

speed = 4;

r = 10;

health = 2;

}

**if**(rank == 3) {

speed = 2.5;

r = 20;

health = 3;

}

**if**(rank == 4) {

speed = 1.5;

r = 30;

health = 4;

}

}

// stronger, faster default

**if**(type == 2) {

//color1 = Color.RED;

color1 = **new** Color(255, 0, 0, 128);

**if**(rank == 1) {

speed = 3;

r = 5;

health = 2;

}

**if**(rank == 2) {

speed = 3;

r = 10;

health = 3;

}

**if**(rank == 3) {

speed = 2.5;

r = 20;

health = 3;

}

**if**(rank == 4) {

speed = 2.5;

r = 30;

health = 4;

}

}

// slow, but hard to kill

**if**(type == 3) {

//color1 = Color.GREEN;

color1 = **new** Color(0, 255, 0, 128);

**if**(rank == 1) {

speed = 1.5;

r = 5;

health = 3;

}

**if**(rank == 2) {

speed = 1.5;

r = 10;

health = 4;

}

**if**(rank == 3) {

speed = 1.5;

r = 25;

health = 5;

}

**if**(rank == 4) {

speed = 1.5;

r = 45;

health = 5;

}

}

x = Math.*random*() \* GamePanel.*WIDTH* / 2 + GamePanel.*WIDTH* / 4;

y = -r;

**double** angle = Math.*random*() \* 140 + 20;

rad = Math.*toRadians*(angle);

dx = Math.*cos*(rad) \* speed;

dy = Math.*sin*(rad) \* speed;

ready = **false**;

dead = **false**;

hit = **false**;

hitTimer = 0;

}

// FUNCTIONS

**public** **double** getx() { **return** x; }

**public** **double** gety() { **return** y; }

**public** **int** getr() { **return** r; }

**public** **int** getType() { **return** type; }

**public** **int** getRank() { **return** rank; }

**public** **void** setSlow(**boolean** b) { slow = b; }

**public** **boolean** isDead() { **return** dead; }

**public** **void** hit() {

health--;

**if**(health <= 0) {

dead = **true**;

}

hit = **true**;

hitTimer = System.*nanoTime*();

}

**public** **void** explode() {

**if**(rank > 1) {

**int** amount = 0;

**if**(type == 1) {

amount = 3;

}

**if**(type == 2) {

amount = 3;

}

**if**(type == 3) {

amount = 4;

}

**for**(**int** i = 0; i < amount; i++) {

Enemy e = **new** Enemy(getType(), getRank() - 1);

e.setSlow(slow);

e.x = **this**.x;

e.y = **this**.y;

**double** angle = 0;

**if**(!ready) {

angle = Math.*random*() \* 140 + 20;

}

**else** {

angle = Math.*random*() \* 360;

}

e.rad = Math.*toRadians*(angle);

GamePanel.*enemies*.add(e);

}

}

}

**public** **void** update() {

**if**(slow) {

x += dx \* 0.3;

y += dy \* 0.3;

}

**else** {

x += dx;

y += dy;

}

**if**(!ready) {

**if**(x > r && x < GamePanel.*WIDTH* - r &&

y > r && y < GamePanel.*HEIGHT* - r) {

ready = **true**;

}

}

**if**(x < r && dx < 0) dx = -dx;

**if**(y < r && dy < 0) dy = -dy;

**if**(x > GamePanel.*WIDTH* - r && dx > 0) dx = -dx;

**if**(y > GamePanel.*HEIGHT* - r && dy > 0) dy = -dy;

**if**(hit) {

**long** elapsed = (System.*nanoTime*() - hitTimer) / 1000000;

**if**(elapsed > 50) {

hit = **false**;

hitTimer = 0;

}

}

}

**public** **void** draw(Graphics2D g) {

**if**(hit) {

g.setColor(Color.***WHITE***);

g.fillOval((**int**) (x - r), (**int**) (y - r), 2 \* r, 2 \* r);

g.setStroke(**new** BasicStroke(3));

g.setColor(Color.***WHITE***.darker());

g.drawOval((**int**) (x - r), (**int**) (y - r), 2 \* r, 2 \* r);

g.setStroke(**new** BasicStroke(1));

}

**else** {

g.setColor(color1);

g.fillOval((**int**) (x - r), (**int**) (y - r), 2 \* r, 2 \* r);

g.setStroke(**new** BasicStroke(3));

g.setColor(color1.darker());

g.drawOval((**int**) (x - r), (**int**) (y - r), 2 \* r, 2 \* r);

g.setStroke(**new** BasicStroke(1));

}

}

}

Explosion.java

**import** java.awt.\*;

**public** **class** Explosion {

// FIELDS

**private** **double** x;

**private** **double** y;

**private** **int** r;

**private** **int** maxRadius;

// CONSTRUCTOR

**public** Explosion(**double** x, **double** y, **int** r, **int** max) {

**this**.x = x;

**this**.y = y;

**this**.r = r;

maxRadius = max;

}

**public** **boolean** update() {

r += 2;

**if**(r >= maxRadius) {

**return** **true**;

}

**return** **false**;

}

**public** **void** draw(Graphics2D g) {

g.setColor(**new** Color(255, 255, 255, 128));

g.setStroke(**new** BasicStroke(3));

g.drawOval((**int**) (x - r), (**int**) (y - r), 2 \* r, 2 \* r);

g.setStroke(**new** BasicStroke(1));

}

}  
Player.java

**import** java.awt.\*;

**public** **class** Player {

// FIELDS

**private** **int** x;

**private** **int** y;

**private** **int** r;

**private** **int** dx;

**private** **int** dy;

**private** **int** speed;

**private** **boolean** left;

**private** **boolean** right;

**private** **boolean** up;

**private** **boolean** down;

**private** **boolean** firing;

**private** **long** firingTimer;

**private** **long** firingDelay;

**private** **boolean** recovering;

**private** **long** recoveryTimer;

**private** **int** lives;

**private** Color color1;

**private** Color color2;

**private** **int** score;

**private** **int** powerLevel;

**private** **int** power;

**private** **int**[] requiredPower = {

1, 2, 3, 4, 5

};

// CONSTRUCTOR

**public** Player() {

x = GamePanel.*WIDTH* / 2;

y = GamePanel.*HEIGHT* / 2;

r = 5;

dx = 0;

dy = 0;

speed = 5;

lives = 8;

color1 = Color.***WHITE***;

color2 = Color.***RED***;

firing = **false**;

firingTimer = System.*nanoTime*();

firingDelay = 200;

recovering = **false**;

recoveryTimer = 0;

score = 0;

}

// FUNCTIONS

**public** **int** getx() { **return** x; }

**public** **int** gety() { **return** y; }

**public** **int** getr() { **return** r; }

**public** **int** getScore() { **return** score; }

**public** **int** getLives() { **return** lives; }

**public** **boolean** isDead() { **return** lives <= 0; }

**public** **boolean** isRecovering() { **return** recovering; }

**public** **void** setLeft(**boolean** b) { left = b; }

**public** **void** setRight(**boolean** b) { right = b; }

**public** **void** setUp(**boolean** b) { up = b; }

**public** **void** setDown(**boolean** b) { down = b; }

**public** **void** setFiring(**boolean** b) { firing = b; }

**public** **void** addScore(**int** i) { score += i; }

**public** **void** gainLife() {

lives++;

}

**public** **void** loseLife() {

lives--;

recovering = **true**;

recoveryTimer = System.*nanoTime*();

}

**public** **void** increasePower(**int** i) {

power += i;

**if**(powerLevel == 4) {

**if**(power > requiredPower[powerLevel]) {

power = requiredPower[powerLevel];

}

**return**;

}

**if**(power >= requiredPower[powerLevel]) {

power -= requiredPower[powerLevel];

powerLevel++;

}

}

**public** **int** getPowerLevel() { **return** powerLevel; }

**public** **int** getPower() { **return** power; }

**public** **int** getRequiredPower() { **return** requiredPower[powerLevel]; }

**public** **void** update() {

**if**(left) {

dx = -speed;

}

**if**(right) {

dx = speed;

}

**if**(up) {

dy = -speed;

}

**if**(down) {

dy = speed;

}

x += dx;

y += dy;

**if**(x < r) x = r;

**if**(y < r) y = r;

**if**(x > GamePanel.*WIDTH* - r) x = GamePanel.*WIDTH* - r;

**if**(y > GamePanel.*HEIGHT* - r) y = GamePanel.*HEIGHT* - r;

dx = 0;

dy = 0;

// firing

**if**(firing) {

**long** elapsed = (System.*nanoTime*() - firingTimer) / 1000000;

**if**(elapsed > firingDelay) {

firingTimer = System.*nanoTime*();

**if**(powerLevel < 2) {

GamePanel.*bullets*.add(**new** Bullet(270, x, y));

}

**else** **if**(powerLevel < 4) {

GamePanel.*bullets*.add(**new** Bullet(270, x + 5, y));

GamePanel.*bullets*.add(**new** Bullet(270, x - 5, y));

}

**else** {

GamePanel.*bullets*.add(**new** Bullet(270, x, y));

GamePanel.*bullets*.add(**new** Bullet(275, x + 5, y));

GamePanel.*bullets*.add(**new** Bullet(265, x - 5, y));

}

}

}

**if**(recovering) {

**long** elapsed = (System.*nanoTime*() - recoveryTimer) / 1000000;

**if**(elapsed > 2000) {

recovering = **false**;

recoveryTimer = 0;

}

}

}

**public** **void** draw(Graphics2D g) {

**if**(recovering) {

g.setColor(color2);

g.fillOval(x - r, y - r, 2 \* r, 2 \* r);

g.setStroke(**new** BasicStroke(3));

g.setColor(color2.darker());

g.drawOval(x - r, y - r, 2 \* r, 2 \* r);

g.setStroke(**new** BasicStroke(1));

}

**else** {

g.setColor(color1);

g.fillOval(x - r, y - r, 2 \* r, 2 \* r);

g.setStroke(**new** BasicStroke(3));

g.setColor(color1.darker());

g.drawOval(x - r, y - r, 2 \* r, 2 \* r);

g.setStroke(**new** BasicStroke(1));

}

}

}

PowerUp.java

**import** java.awt.\*;

**public** **class** PowerUp {

// FIELDS

**private** **double** x;

**private** **double** y;

**private** **int** r;

**private** **int** type;

**private** Color color1;

// 1 -- +1 life

// 2 -- +1 power

// 3 -- +2 power

// 4 -- slow down time

// CONSTRUCTOR

**public** PowerUp(**int** type, **double** x, **double** y) {

**this**.type = type;

**this**.x = x;

**this**.y = y;

**if**(type == 1) {

color1 = Color.***PINK***;

r = 3;

}

**if**(type == 2) {

color1 = Color.***YELLOW***;

r = 3;

}

**if**(type == 3) {

color1 = Color.***YELLOW***;

r = 5;

}

**if**(type == 4) {

color1 = Color.***WHITE***;

r = 3;

}

}

// FUNCTIONS

**public** **double** getx() { **return** x; }

**public** **double** gety() { **return** y; }

**public** **double** getr() { **return** r; }

**public** **int** getType() { **return** type; }

**public** **boolean** update() {

y += 2;

**if**(y > GamePanel.*HEIGHT* + r) {

**return** **true**;

}

**return** **false**;

}

**public** **void** draw(Graphics2D g) {

g.setColor(color1);

g.fillRect((**int**) (x - r), (**int**) (y - r), 2 \* r, 2 \* r);

g.setStroke(**new** BasicStroke(3));

g.setColor(color1.darker());

g.drawRect((**int**) (x - r), (**int**) (y - r), 2 \* r, 2 \* r);

g.setStroke(**new** BasicStroke(1));

}

}

Text.java

**import** java.awt.\*;

**public** **class** Text {

// FIELDS

**private** **double** x;

**private** **double** y;

**private** **long** time;

**private** String s;

**private** **long** start;

// CONSTRUCTOR

**public** Text(**double** x, **double** y, **long** time, String s) {

**this**.x = x;

**this**.y = y;

**this**.time = time;

**this**.s = s;

start = System.*nanoTime*();

}

**public** **boolean** update() {

**long** elapsed = (System.*nanoTime*() - start) / 1000000;

**if**(elapsed > time) {

**return** **true**;

}

**return** **false**;

}

**public** **void** draw(Graphics2D g) {

g.setFont(**new** Font("Century Gothic", Font.***PLAIN***, 12));

**long** elapsed = (System.*nanoTime*() - start) / 1000000;

**int** alpha = (**int**) (255 \* Math.*sin*(3.14 \* elapsed / time));

**if**(alpha > 255) alpha = 255;

g.setColor(**new** Color(255, 255, 255, alpha));

**int** length = (**int**) g.getFontMetrics().getStringBounds(s, g).getWidth();

g.drawString(s, (**int**) (x - (length / 2)), (**int**) y);

}

}