МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
ОДЕСЬКИЙ ДЕРЖАВНИЙ ЕКОЛОГІЧНИЙ УНІВЕРСИТЕТ

Факультет Магістерської та
аспірантської підготовки
Кафедра інформаційних технологій

МАГІСТЕРСЬКА КВАЛІФІКАЦІЙНА РОБОТА

на тему: «Графічне моделювання та пошарова анімація об’єктів»

Виконав студент 2 року групи
МІС- 18 спеціальності 122
Комп’ютерні науки
Фунтова Ірина Леонідівн
Керівник д.х.н. проф..
Кругляк Юрій Олексійович
Рецензент
Мещеряков Володимир Іванович

Одеса 2018

АНОТАЦІЯ

на магістерську роботу «Дослідження технологій компонування та тестування програмного продукту»,
студентки Фунтової Ірини Леонідівни

В магістерській роботі проведено тестування програмного продукту «NoSleep» методом виявлення недоліків та їх усунення. Розглянуто класифікації помилок, які допомогли в подальшому тестуванні гри.

Об'єкт дослідження – комп'ютерна гра «NoSleep».

Предмет дослідження – тестування програмного продукту.

Мета роботи полягає у перевірці функціонування гри, виявлення помилок та їх усунення.

Методи дослідження – опис характеристик тестування і тестування продукту з результатами.

Структура магістерської роботи складається з анотації, вступу, чотирьох розділів, висновків, переліку посилань на 15 найменувань, додатків. Повний обсяг роботи становить 85 сторінок, містить 17 рисунків.

Ключові слова: баги, тестування програмного продукту, QA.

ABSTRACT

The master's thesis tested the NoSleep software by detecting deficiencies and correcting them. The bug classifications that helped further test the game are discussed.

Object of Study - NoSleep computer game.

The subject of the study is software testing.

The purpose of the job is to check the functioning of the game, detecting soap and removing them.

Research Methods - A description of the testing and testing characteristics of a product with results.

The structure of the master's work consists of an abstract, introduction, four or four sections, conclusions, a list of references to 15 titles, applications. The volume of the work is 85 pages, contains 17 figures.

Keywords: bugs, software testing, QA.

ЗМІСТ

СКОРОЧЕННЯ ТА УМОВНІ ПОЗНАЧКИ

IBM	– International Business Machines
ENIAC	– Electronic Numerical Integrator and Computer
EDVAC	– Electronic Discrete Variable Automatic Computer
ПЗ	– Програмне забезпечення
ОС	– Операційна система
ЕОМ	– Електронно-обчислювальна машина
ACM	– Association for Computing Machinery
RPG	– Role-playing game
QA	– Quality assurance
США	– Сполучені Штати Америки
Fortran	– FORmula TRANslator
ALGOL	– Algorithmic language
LISP	– LISt Processing language
COBOL	– COmmon Business Oriented Language
MIT	– Massachusetts Institute of Technology
ПК	– Персональний компьютер
PC	– Personal computer
PDP	– Programmed Data Processor
MOBA	– Multiplayer Online Battle Arena
NPC	– Non-Player Character
3D	– 3-dimensional
CMS	– Construction and management simulation
CRPG	– Computer Role-Playing Game
ПЗ	– Програмне забезпечення
TDD	– Test-driven development
DDOS	– Distributed Denial of Service
MMO	– Massively Multiplayer Online Game
FPS	– First-person shooter

ВСТУП

Основний пік інтересу до тестування програмного забезпечення припав на дев'яності роки в США. Швидкий розвиток систем автоматизованої розробки програмного забезпечення та мережевих технологій привело до збільшення виробництва на ринку програмного забезпечення. Посилення конкуренції між виробниками програмного забезпечення вимагало підвищеної уваги до якості продукції. Оскільки асортимент продукції сильно розширився, а ціни стали доступнішими, споживачі почали звертати більшу увагу на якість програмного забезпечення.

В даний час практично всі сфери життя схильні до комп'ютеризації. Мало того, що комп'ютери, використовуються в повсякденному житті для звичайних цілей, вони також необхідні, коли мова йде про набагато більш значущих сферах, таких як медицина, транспорт, будівництво, безпеку і багато інших. Таким чином, питання про якість програмного забезпечення стає особливо важливим, оскільки це не тільки питання комфорту, а й безпеки.

Усвідомлюючи вищезазначене, велика кількість компаній по всьому світу почали інвестувати в підвищення якості програмного забезпечення - почали створюватися відділи контролю якості і застосовуватися нові технології, які дозволили компаніям збільшити свою конкурентну перевагу, за рахунок підвищення якості своїх програмних продуктів.

Незабаром після цього тестування програмного забезпечення стало невід'ємною частиною виробництва програмного забезпечення. Тестування необхідно для того, щоб зрозуміти, чи працює програма, як очікується і чи відповідає вона висунутим до неї вимогам. Своєчасне виявлення і виправлення помилок і недоробок має величезне значення в процесі розробки програмного продукту, оскільки це зменшує ризики і при цьому відбувається зниження витрат на розробку програмного забезпечення. Завдяки тестуванню, компанії здатні підтримувати якість своїх продуктів на дуже високому рівні. Часто

процес тестування ПО може бути автоматизований, що в деяких випадках може позитивно відіб'ється на швидкості і якості тестування, це дозволяє ще більше знизити витрати компанії і підвищити якість продукту.

На даний момент пильна увага приділяється процесам тестування, способам мінімізувати витрати і автоматизувати процес тестування. Зараз існує досить велика кількість книг і статей на різні теми, будь то загальні поняття в сфері тестування, або дослідження вузької спрямованості.

Метою данної магістерської роботи є опис характеристик тестування і тестування продукту з результатами.

Для досягнення поставленої мети в роботі необхідно вирішити наступні завдання:

- створити теоретичну базу актуальних способів і методів ПЗ;
- розробити методики тестування програмного продукту під конкретні завдання розробника;
- провести сам процес тестування, ознайомитися з його результатами;
- зробити висновок про якість тестованого продукту.

1 ТЕОРЕТИЧНЕ ВВЕДЕННЯ

1.1 Виникнення обчислювальної техніки

Обчислювальна техніка є найважливішим компонентом процесу обчислень і обробки даних. Першими пристосуваннями для обчислень були, ймовірно, всім відомі рахункові палички, які і сьогодні використовуються в початкових класах багатьох шкіл для навчання рахунку. Розвиваючись, ці пристосування ставали більш складними, наприклад, такими як фінікійські глиняні фігурки, також призначаються для наочного уявлення кількості рахованих предметів. Такими пристосуваннями, схоже, користувалися торговці і рахівники того часу.

Поступово з найпростіших пристосувань для рахунку народжувалися все більш і більш складні пристрої: абак (рахунки), логарифмічна лінійка, арифмометр, комп'ютер. Одним із прикладів можна назвати палички Непера. Заслуга їх винаходу приписується шотландському математику Джону Непером (автор логарифмів) і описується в його трактаті від 1617 року.

Незважаючи на простоту ранніх обчислювальних пристроїв, досвідчений рахівник може отримати результат за допомогою простих рахунків навіть швидше, ніж нерозторопний власник сучасного калькулятора. Природно, продуктивність і швидкість рахунку сучасних обчислювальних пристроїв вже давно перевершують можливості найвидатнішого рахувальника-людини.
Потреба в складних розрахунках в XVI столітті швидко росла. Значна частина труднощів була пов'язана з множенням і діленням багатозначних чисел.

Це призвело до появи на протязі найкоротшого часу (1614-1623 рр.) відразу чотирьох нових типів обчислювачів:

- логарифмічних таблиць;
- логарифмічних лінійок;

- механічних арифмометрів (швидше за перевідкриття, бо існували в античності);
- паличок Непера.

Пізніше вже в XIX столітті на базі логарифмів і логарифмічних лінійок виникла і їх графічний аналог – номограми, які стали використовуватися для обчислення самих різних функцій.

У 1623 році Вільгельм Шиккард придумав «Розрахунковий годинник» - перший арифмометр, що вмів виконувати чотири арифметичних дії. Вважають годинами пристрій було названо тому, що, як і в справжніх годинниках, робота механізму була заснована на використанні зірочок і шестерень. Цей винахід знайшло практичне використання в руках одного Шиккарда, філософа і астронома Йоганна Кеплера. За цим послідували машини Блеза Паскаля («Паскаліна», 1642 г.) і Готфріда Вільгельма Лейбніца – арифмометр Лейбніца.

Лейбніц також описав двійкову систему числення – один з ключових принципів побудови всіх сучасних комп'ютерів. Однак, аж до 1940-х багато наступних розробки (включаючи машини Чарльза Беббіджа і навіть ЕНІАК 1945 року) були засновані на більш складної в реалізації десятковій системі.

У 1804 році Жозеф Марі Жаккар розробив ткацький верстат, в якому вишивали узор визначався перфокартами. Серія карт могла бути замінена, і зміна візерунка не вимагала змін в механіці верстата. Це було важливою віхою в історії програмування.

У 1832 році Семен Корсаков застосував перфоровані карти в конструкції розроблених ним «інтелектуальних машин», механічних пристроїв для інформаційного пошуку, які є прообразами сучасних баз даних та, в якійсь мірі, – експертних систем.

У 1838 році Чарльз Беббідж перейшов від розробки різницевих машини до проектування складнішою аналітичної машини, принципи програмування якій безпосередньо сходять до перфокарт Жаккар.

У 1890 році Бюро Перепису США використовувало перфокарти і механізми сортування (табулятори), розроблені Германом Холлеритом, щоб обробити потік даних десятирічної перепису, переданий під мандат відповідно до Конституції. Компанія Холлерита в кінцевому рахунку стала ядром IBM. Ця корпорація розвинула технологію перфокарт в потужний інструмент для обробки ділових даних і випустила велику лінію спеціалізованого обладнання для їх запису. До 1950 року технологія IBM стала всюдисущої в промисловості й уряді. Попередження, надруковане на більшості карт, «не звертати, не скручувати і не рвати», стало девізом післявоєнної ери.

У багатьох комп'ютерних рішеннях перфокарти використовувалися до (і після) кінця 1970-х. Наприклад, студенти інженерних і наукових спеціальностей у багатьох університетах у всьому світі могли відправити їх програмні команди в локальний комп'ютерний центр у формі набору карт, одна карта на програмну рядок, а потім повинні були чекати черги для обробки, компіляції та виконання програми. Згодом, після роздруківки будь-яких результатів, зазначених ідентифікатором заявника, вони містилися в випускний лоток поза комп'ютерного центру. У багатьох випадках ці результати включали в себе виключно роздруківку повідомлення про помилку в синтаксисі програми, вимагаючи іншого циклу редагування – компіляція – виконання.

До 1900 року арифмометри, касові апарати та рахункові машини були перепроектовані з використанням електричних двигунів з поданням положення змінної як позиції шестерні. З 1930-х настільні арифмометри, які могли складати, віднімати, множити і ділити, почали випускати такі компанії як Friden, Marchant і Monro. Словом «computer» (буквально – «обчислювач») називалася посада – це були люди, які використовували калькулятори для виконання математичних обчислень.

У 1939 році в Endicott laboratories в IBM почалася робота над Harvard Mark I. Офіційно відомий як Automatic Sequence Controlled Calculator, Mark I був електромеханічним комп'ютером загального призначення, створеним з фінансуванням IBM і за допомогою з боку персоналу IBM під керівництвом гарвардського математика Говарда Айкена. Проект комп'ютера був створений під впливом Аналітичної машини Ч. Беббіджа з використанням десяткової арифметики, коліс для зберігання даних і поворотних перемикачів на додаток до електромагнітних реле. Машина програмувались за допомогою перфострічки і мала кілька обчислювальних блоків, які працювали паралельно. Пізніші версії мали кілька зчитувачів з перфострічки, і машина могла перемикатися між зчитувачами в залежності від стану. Проте, машина була не зовсім Тьюринг-повна. Mark I був перенесений до Гарвардського університету і почав роботу в травні 1944 року.

Американський ENIAC, який часто називають першим електронним комп'ютером загального призначення, публічно довів застосовність електроніки для масштабних обчислень. Це стало ключовим моментом у розробці обчислювальних машин, перш за все через величезну приросту в швидкості обчислень, але також і через що з'явилися можливостей для мініатюризації. Створена під керівництвом Джона мокли і Дж. Преспера Екерта, ця машина була в 1000 разів швидше, ніж всі інші машини того часу. Розробка «ЕНІАК» тривала з 1943 до 1945 року.

У той час, коли був запропонований даний проект, багато дослідників були переконані, що серед тисяч тендітних електровакуумних ламп багато будуть згоряти настільки часто, що «ЕНІАК» буде занадто багато часу простоювати в ремонті, і, тим самим, буде практично марний. Проте, на реальній машині вдавалося виконувати кілька тисяч операцій в секунду протягом декількох годин, до чергового збою через згорілої лампи.

«ЕНІАК», безумовно, задовольняє вимогу повноти по Тьюрингу. Але «програма» для цієї машини визначалася станом сполучних кабелів і

перемикачів – величезна відмінність від машин з програмою, що зберігається, що з'явилися у Конрада Цузе в 1940 році[1][1)]. Проте, в той час обчислення, що виконувалися без допомоги людини, розглядалися як досить велике досягнення, і метою програми було тоді рішення тільки однією єдиною завдання. (Покращення, які були завершені в 1948 році, дали можливість виконання програми, записаної в спеціальній пам'яті, що зробило програмування більш систематичним, менш «одноразовим» досягненням).

Переробивши ідеї Екерта і мокли, а також оцінивши обмеження «ЕНІАК», Джон фон Нейман, написав широко цитований звіт, що описує проект комп'ютера (EDVAC), в якому і програма, і дані зберігаються в єдиній універсальній пам'яті. Принципи побудови цієї машини стали відомі під назвою «архітектура фон Неймана» і послужили основою для розробки перших по-справжньому гнучких універсальних цифрових комп'ютерів.

У той час в СССР У 1944 році Лебедєв повертається до Москви, а потім приїжджає до Києва і стає директором Інституту електротехніки АН УРСР. У 1945 р. С.О.Лебедєв був обраний академіком і призначений директором Інституту енергетики. Через півроку інститут розділили на дві частини - Інститут електротехніки та Інститут теплотехніки. С.О.Лебедєв став директором Інституту електротехніки. У той час пріоритетними напрямками в науці вважалися ракетна техніка, атомна енергетика, дослідження космосу. Але Лебедєв залишається вірним своєму довоєнному задуму - створити цифрову електронну обчислювальну машину. Машина займала найбільшу кімнату площею 60 м2 у лівому крилі лабораторії в Феофанії і працювала з небувалою на ті часи швидкістю - 3 тисячі операцій за хвилину (для порівняння, сучасні комп'ютери виконують мільйони операцій за секунду) і могла виконувати операції віднімання, додавання, множення, ділення, зсуву,

[1] [1)] Історія виникнення комп’ютера URL: https://sites.google.com/site/komputertvijpomicnik/komputer--tvij-pomicnik/istoria-viniknenna-komputera (дата звернення: 11.11.2019)

порівняння з урахуванням знака, порівняння за абсолютною величиною, передачі керування, передачі чисел з магнітного барабану, складання команд. У машині були використані електронні лампи (6000 штук!) загальною потужністю споживання в 25 кВт. Перший запуск «МЭСМ» запам'ятався саме «ламповою проблемою»: при включенні машини 6000 ламп, що запрацювали одночасно, перетворили приміщення в тропіки. Технікам довелося терміново розбирати стелю, щоб відвести з кімнати хоча б частину тепла.[2][2)]

1.2. Виникнення язиків програмування

Прогрес комп'ютерних технологій визначив процес появи нових різноманітних знакових систем для запису алгоритмів мов програмування. Сенс появи такої мови – спрощення програмного коду.

З кожним днем наш світ стає більш мобільним і інформаційним. Все більше і більше комп'ютери вступають в наше повсякденне життя, і щоб полегшити наше спілкування з ними створюється нове ПЗ за допомогою різних мов програмування.

Мови програмування прийнято ділити на п'ять поколінь. У перше покоління входять мови, створені на початку 50-х років, коли перші комп'ютери тільки з'явилися на світло. Це був перший мову асемблера, створений за принципом «одна інструкція – один рядок».

Фізичні принципи роботи електронних пристроїв ЕОМ такі, що комп'ютер може сприймати команди, що складаються тільки з одиниць і нулів – послідовність перепаду напруги, тобто машинний код. На початковій стадії розвитку ЕОМ людині було необхідно складати програми на мові, зрозумілій комп'ютеру, в машинних кодах. Кожна команда складалася з коду операцій і адрес операндів, виражених у вигляді різних поєднань одиниць і нулів. Отже,

---

[2] [2)] Як народжувався перший комп'ютер URL: http://ua.uacomputing.com/stories/mesm/ (дата звернення: 13.11.2019)

будь-яка програма для процесора виглядала на той час як послідовність одиниць і нулів.

Як показала надалі практика спілкування з комп'ютером, така мова громіздкий і незручний. При користуванні їм легко припуститися помилки, записавши не в тій послідовності 1 або 0. Програму дуже важко контролювати. Крім того, при програмуванні в машинних кодах треба добре знати внутрішню структуру ЕОМ, принцип роботи кожного блоку. І найгірше в такій мові, що програми на даній мові – дуже довгі послідовності одиниць і нулів є машинно залежними, тобто для кожної ЕОМ необхідно було складати свою програму, а так само програмування в машинних кодах вимагає від програміста багато часу, праці, підвищеного уваги.

Досить скоро стало зрозуміло, що процес формування машинного коду можна автоматизувати. Уже в 1950 році для запису програм почали застосовувати мнемонічний мову – мову assembly. Мова асемблера дозволив уявити машинний код в більш зручній для людини формі: для позначення команд і об'єктів, над якими ці команди виконуються, замість двійкових кодів використовувалися букви або скорочені слова, які відображали суть команди. Наприклад, на мові асемблера команда складання двох чисел позначається словом add, тоді як її машинний код може бути таким: 000010.

Асемблер – мова програмування низького рівня. Мова програмування низького рівня – це мова програмування, який орієнтований на конкретний тип процесора і враховує його особливості. В даному випадку «низький рівень» не означає «поганий». Мається на увазі, що оператори мови близькі до машинного коду і орієнтовані на конкретні команди процесора. Поява мови асемблера значно полегшило життя програмістів, так як тепер замість рябящіх в очах нулів і одиниць, вони могли писати програму командами, що складаються з символів наближених до звичайної мови. Для того часу ця мова була нововведенням і користувався популярністю так як дозволяв писати програми невеликого розміру, що при тих машинах критерій значний.

Але складність розробки в ньому великих програмних комплексів призвела до появи мов третього покоління – мов високого рівня. Але на цьому застосування асемблера не закінчилася, він користується популярністю у вузьких колах і до цього дня. Зараз його використовують в написанні окремих фрагментів програм або іноді в написанні самих програм. Прикладів може бути багато, але найяскравіші це використання асемблера в написанні драйверів, ігр та завантажувачів ОС.

Не варто забувати, що у хакерів цю мову так само користується популярністю, в зв'язку з тим, що швидкість роботи отриманої програми значно вище швидкості програми написаної на мові програмування високого рівня. Це пояснюється тим, що вийшов розмір програми дуже малий. Розробники антивірусів так само використовують асемблер в деяких модулях своїх програм, що так само забезпечує їх швидкодія.

Середина 50-х рр. характеризується стрімким прогресом в області програмування. Роль програмування в машинних кодах стала зменшуватися, стали з'являтися мови програмування нового типу, що виступають в ролі посередника між машинами і програмістами. Настав час другого і третього поколінь мов програмування.

З середини 50-х рр. XX ст. почали створювати перші мови програмування високого рівня (high-level programming languages). Ці мови не були прив'язані до певного типу ЕОМ. Для кожного з них були розроблені власні компілятори. Компіляція – трансляція програми, складеної мовою оригіналу високого рівня, в еквівалентну програму на низкорівневою мовою, близькою машинного коду (абсолютний код, об'єктний модуль, іноді мова асемблера). Яскравими представниками перших мов програмування є Fortran, ALGOL 68 і LISP.

Перша мова високого рівня Фортран був створений в період з 1954 по 1957 рік групою програмістів під керівництвом Джона Бекуса в корпорації

IBM. Він призначався для наукових і технічних розрахунків. Назва Fortran є скороченням від FORmula TRANslator (перекладач формул).

Оскільки Фортран виявився настільки успішним мовою, в Європі виникли побоювання, що IBM буде домінувати в комп'ютерній галузі. Німецьке Товариство прикладної математики і механіки (GAMM) створило комітет з розробки універсальної мови. У той же час Асоціація обчислювальної техніки (ACM) організувала схожий комітет в США. Незважаючи на те, що у європейців було деяке занепокоєння з приводу панування американців, обидва цих комітету злилися в один.

Алгол був розроблений в 1958 році на тижневої конференції в ETH (Цюріх, Швейцарія) як універсальна мова програмування для широкого кола застосувань, а потім доопрацьований комітетом, створеним Міжнародною федерацією з обробки інформації[3][3)]. До комітету увійшли ряд провідних європейських і американських вчених і інженерів-розробників мов, серед яких були Джон Бекус, Джон Маккарті, Петер Наур, Едсгер Дейкстра і Джозеф Уегстіл, згодом очолив комітет із розробки мови Кобол.

Кобол був розроблений в 1959 році і призначався насамперед для написання програм для розробки бізнес-додатків, а так само для роботи в економічній сфері.

Специфікація мови була створена в 1959 році. Творці мови ставили собі за мету зробити його машінонезавісімим і максимально наближеним до природного англійської мови. Обидві цілі були успішно досягнуті; програми на COBOL вважаються зрозумілими навіть неспеціалістам, оскільки тексти на цій мові програмування не потребують будь-яких спеціальних коментарів (самодокументірующіеся програми).

Мова Лісп був запропонований Дж. Маккарті в роботі в 1960 році і орієнтований на розробку програм для вирішення завдань не чисельного харак-

---

[3][3)] Роберт У. Себеста. Основные концепции языков программирования.5-е изд. М.: Вильямс, 2001.672 с.ISBN 5-8459-0192-8.

теру. Англійська назва цієї мови – LISP є абревіатурою виразу LISt Processing (обробка списків) і добре підкреслює основну область його застосування. Поняття «список» виявилося дуже ємним.

У вигляді списків зручно представляти алгебраїчні вирази, графи, елементи кінцевих груп, безлічі, правила виведення і багато інших складні об'єкти. Списки є найбільш гнучкою формою подання інформації в пам'яті комп'ютерів. Тому не дивно, що зручний мову, спеціально призначений для обробки списків, швидко завоював популярність.

На зорі комп'ютеризації (на початку 1950-х р.), машинний мова була єдиною мовою, більшого людина до того часу не придумав. Мови низького рівня мало схожі на нормальний, звичний людині мову. Великі, громіздкі програми на таких мовах пишуться рідко. Зате якщо програма буде написана такою мовою, то вона буде працювати швидко, займаючи маленький обсяг і допускаючи мінімальну кількість помилок. Чим нижче і ближче до машинного рівень мови, тим менше і конкретніше завдання, які ставляться перед кожною командою.

Для порятунку програмістів від суворого машинного мови програмування, були створені мови високого рівня (тобто немашінние мови), які стали своєрідною сполучною мостом між людиною і машинним мовою комп'ютера. Мови високого рівня працюють через трансляційні програми, які вводять «вихідний код» (гібрид англійських слів і математичних виразів, який зчитує машина), і в кінцевому підсумку змушує комп'ютер виконувати відповідні команди, які даються на машинному мовою.

З появою мов високого рівня програмісти отримали можливість більше часу приділяти вирішенню конкретної проблеми, не відволікаючись особливо на вельми тонкі питання організації самого процесу виконання завдання на машині. Крім того, поява цих мов ознаменувало перший крок на шляху створення програм, які вийшли за межі науково-дослідних лабораторій і фінансових відділів.

Підводячи підсумок даного періоду розвитку мов програмування, можна зробити висновок, що мови програмування високого рівня (FORTRAN, ALGOL, LISP, COBOL і т. Д.) Не схожі на мову асемблера. Мови високого рівня розроблені спеціально для того, щоб можна було мати справу безпосередньо з завданням, розв'язуваної програмою. На цій посаді вони іноді називаються процедурними мовами, оскільки описують процедуру, яка використовується для вирішення завдання. Мови високого рівня машінонезавісіми. Програми ж на мові асемблера безпосередньо відносяться до тієї машини, на якій вони повинні виконуватися.

Переваги мов програмування високого рівня:

- алфавіт мови значно ширше машинного, що робить його набагато більш виразним і істотно підвищує наочність і зрозумілість тексту;
- набір операцій, допустимих для використання, не залежить від набору машинних операцій, а вибирається з міркувань зручності формулювання алгоритмів розв'язання задач певного класу;
- конструкції операторів задаються в зручному для людини вигляді;
- підтримується широкий набір типів даних.

Недоліком мов високого рівня є більший розмір програм в порівнянні з програмами на мові низького рівня. Тому в основному мови високого рівня використовуються для розробок програмного забезпечення комп'ютерів і пристроїв, які мають великий обсяг пам'яті. А різні підвиди асемблера застосовуються для програмування інших пристроїв, де критичним є розмір програми.

2 АНАЛІЗ РОЗВИТКУ ІГРОВОЇ ІНДУСТРІЇ

2.1 **Зародження ігор**

Можна сказати, що комп'ютерні ігри з'явилися випадково. Сталося це в 1958 році в Брукхевенської Національної Лабораторії (США), коли її співробітник Вільям Хігінботам – представив перший в історії електронний пінг-понг відвідувачам цієї відомої наукової організації. Пізніше він зізнався, що і уявити собі не міг, який успіх чекає в майбутньому його нехитре нововведення. Дещо пізніше в 1961 р – програмісти знаменитого Массачусетського Технологічного Інституту (MIT) на своїх суперкомп'ютерах (для того часу, звичайно) створили ще одну гру. Вона називалася "Зоряні війни" і розроблялася як спосіб комп'ютерного моделювання військових дій. У різні версії цієї гри і сьогодні можна пограти в деяких моделях телефонів.

Хоча персональні комп'ютери стали популярними лише з появою доступних мікропроцесорів, мейнфреймів і міні-комп'ютерів, перші ПК-гри з'явилися в 1960-х роках. Однією з перших PC-ігор став космічний симулятор

Spacewar, розроблений в 1961 році студентами MIT для комп'ютера PDP-1, який використовувався в інституті для статистичних обчислень.

Перше покоління PC-ігор було переважно текстовими пригодами або інтерактивними історіями, в яких гравець «спілкувався» з комп'ютером за допомогою введення команд через клавіатуру. Перша текстова пригодницька гра, Colossal Cave Adventure, була розроблена для PDP-10 Уіллом Кроутер (англ.) Рос. в 1976 році і розширена Доном Вудсом в 1977 році. До 1980-х років ПК стали досить потужними, щоб виконувати текстові ігри на зразок Colossal Cave Adventure, але в цей час вже почала зароджуватися інтерактивна комп'ютерна графіка. Пізніші гри поєднували в собі текстові команди і базову графіком, що можна побачити на прикладі серії ігор «Gold Box» від розробника SSI.

До середини 1970-х років PC-ігри розроблялися групами ентузіастів і поширювалися в більшій мірі між цими групами. Однак тоді вже були ігрові журнали, такі як Creative Computing і пізніше Computer Gaming World, через які також почали поширюватися PC-ігри. У цих виданнях гри поширювалися у вигляді вихідного програмного коду, що стимулювало споживачів до розробки своїх власних ігор і модифікацій.

Microchess стала однією з перших ігор, яка публічно надійшла в продаж. З початку старту продажів в 1977 році ця гра в кінцевому рахунку зібрала касу в 50 000 проданих копій. Поширювалася вона на касетної плівці.

У 70-х рр. електронні ігри швидко перетворилися на бізнес. Батьком цього бізнесу вважається Нолан Башнелл (він заснував першу фірму по виробництву нового типу ігор, яку він назвав Атари (Atari). Вже в 1972 році його знаменитий "Понг" увірвався в світ ігрових розваг. Це була перша ігрова приставка (до телевізора) , за допомогою якої два гравці або гравець і ком-

п'ютер могли на екрані перекидати через сітку кульку, прагнучи утримати його в екранному поле.

Комерційний успіх привів в цей бізнес інші компанії (серед них такі виробники ігор як Nintendo, Magnavox, Namco, Sega і ін.). В результаті, ще до появи персональних комп'ютерів (!) Електронні ігри здобули величезну популярність, показавши всім можливості і цінність розвитку інформаційних технологій. Правда, поки тільки в США, Японії і Європі. Так що не дарма цей вид ігор називали і називають "електронні ігри" або "відеоігри", оскільки вони проводилися у вигляді спеціальних приставок до телевізора або ігрових автоматів. Перші ігри для персональних комп'ютерів з'явилися лише на початку 80-х рр., Коли бум ігрових приставок вже пройшов, але починався новий – бум комп'ютерних ігор, який триває досі.

Відеоігри швидко стали популярною розвагою. У них грали вдома або в барах і кафе, де вони встановлювалися у вигляді гральних автоматах в коридорах, які нагадували аркаду. Звідси назва самого першого жанру електронних ігор – "аркади" ("Зоряні війни", "Понг", "Покемон", "Арканоід" – все це аркадні ігри).

У другій половині 70-х рр. були винайдені ще два специфічних жанру відеоігор. Це пригодницькі ігри ("дії") та симулятори (симулятор літака). І якщо "дії" (які спочатку були текстовими!) Відносяться до давно існуючому жанру популярної культури – пригодницьким історіям (в літературі, кіно, театрі вони, здається, були завжди), – то симулятори представляють собою досить оригінальне досягнення, щось абсолютно нове, що привнесли з собою електронні ігри в світ розваг. Графічна демо-версія симулятора літака була показана Брюсом Артвік – батьком цього жанру відеоігор – в 1978 р А в 1979 він випустив першу версію (перше покоління) Flight Simulator для комп'ютерів "Еппл". Дивно, але вже в симуляторі 2-го покоління (1982) гравець, який виявляється на місці пілота літака, міг вибрати один з 20 маршрутів, змінювати погодні умови, орієнтуватися по координатам і рельєфу місцевості і ма-

ти зображення на екрані в 4 кольорах. З тих пір компанія Майкрософт випустила на ринок вже 9 поколінь знаменитих Microsoft Flight Simulator. Інші виробники не відставали і сьогодні можна придбати симулятори танків, вертольотів, автомобілів, кораблів.

Ще один чудовий, але зовсім не оригінальний жанр відеоігор – логічні ігри, або ігри – інтелектуальні вправи. Тут ми, звичайно ж, згадуємо "Тетріс". Це гра-пазл, суть якої в нескладному укладанні геометричних фігур на потрібне місце за ознакою кольору і форми. І якщо більшість перших ігор придумали американці, то створенням "Тетріс" ми можемо пишатися, оскільки це внесок російських програмістів в розвиток комп'ютерних ігор ("Тетріс" створив А. Пажітнов на початку 80-х). В логічні ігри люди грали і раніше: шахи, карти, хрестики-нулики, пазли та ін. Однак за допомогою комп'ютера вони, безумовно, знайшли нові форми і велику захопливість.

Початок 90-х рр. ознаменувався появою нового жанру, якому судилося стати найпопулярнішим. З виходом гри "Дум" (Doom) "стрілялки" (або "шутери") підкорили світ. Паралельно все більший успіх завойовували спортивні ігри (електронні версії футболу, хокею, баскетболу, тенісу і т.д.), а також ще один жанр електронних ігор, що отримав назву "стратегії" або стратегічні ігри (знаменита "Цивілізація"). Розвиток Інтернету додало сюди оригінальний жанр онлайнових мережевих ігор (їх нерідко називаю рольовими, що вказує на "коріння" цього жанру в світі рольових ігор та театрального типу гри в цілому).

2.2 Жанри комп’ютерних ігор

Всього, згідно із загальноприйнятою класифікацією, існує 10 основних жанрів ігор:

- action;
- аркада;

- симулятор;
- стратегії;
- пригоди;
- музичні ігри;
- рольові ігри;
- пазли;
- традиційні (настільні);
- текстові.

Action, Екшен (action в перекладі з англ. – «дія») – жанр комп'ютерних ігор, в якому робиться наголос на експлуатацію фізичних можливостей гравця, в тому числі координації очей і рук і швидкості реакції. Жанр представлений в безлічі різновидів від файтингов, шутерів і платформер, які вважаються найбільш важливими для жанру, до MOBA і деяких стратегій в реальному часі, які можливо віднести до жанру екшен. В екшен-іграх зазвичай гравець управляє протагоністом або аватаром. Цей персонаж повинен знайти вихід з рівня, зібрати предмети, уникнути перешкод і поборотися з ворогами різними способами. Дія таких ігор розвивається дуже динамічно і вимагає високої концентрації уваги і швидкої реакції на події в грі події. В кінці рівня або ряду рівнів гравець зазвичай бореться з босом, битва з якому набагато більш вимоглива до гравця, а сам бос, найчастіше, крупніше звичайних ворогів. Перешкоди і ворожі атаки виснажують здоров'я і запас життів аватара. При відсутності у нього життів, гравець отримує повідомлення «Game over». В іншому випадку, коли серія рівнів успішно пройдена, гравець перемагає. Проте, в ряді ігор, особливо в аркадах, кількість рівнів може бути нескінченно, і перемогти в таких іграх неможливо. У цьому випадку метою гравця стає отримання якомога більшої кількості очок, збираючи предмети і знищуючи ворогів;

Файтинг – жанр комп'ютерних ігор, що імітують рукопашний бій малого числа персонажів в межах обмеженого простору, званого ареною. Файтинг

близькі до ігор жанру «побий їх всіх», проте між ними існують відмінності. Так, в більшості файтингов гравцеві не потрібно переміщатися по довгому рівню і не можна вийти за межі арени, а бій складається з непарного числа окремих раундів і не є безперервним. Менш значними і необов'язково присутніми ознаками жанру є використання численних шкал для зображення життєво важливих показників персонажів і промальовування бійців на арені в профіль. Важливою особливістю файтингов є їх націленість на змагання, а не на співпрацю гравців, що робить гри цього жанру придатними для кіберспортівних чемпіонатів.

Зазвичай файтинг надають гравцеві можливість вести бій в режимі «один на один» проти комп'ютерного супротивника або іншого гравця, рідше – дозволяють боротися одночасно трьом або чотирьом противникам на одній арені. Найпершим файтингом вважається випущена в 1979 році на аркадних автоматах гра Warrior, розроблена Тімом Скеллі (Tim Skelly). Гра, яка представляла собою дуель двох лицарів, використовувала монохромну векторну графіку. Потужність процесора аркадного автомата була недостатньою для одночасного обрахунку рухів персонажів і зображення арени, тому персонажі і рахунок гри проектувалися поверх арени, намальованої на корпусі автомата.

Протягом наступних десяти років файтинг як самостійний жанр не мали великої популярності і вважалися однією з різновидів жанру «побий їх всіх», практично зливаючись з ним. Прикладами таких файтингов можуть служити Yie Ar Kung-Fu від компанії «Konami» і Karateka від Джордана Мехнер (Jordan Mechner), творця гри Prince of Persia. Однак ще в 1984 році компанія «Data East» випустила гру Karate Champ, в якій були закладені всі найбільш загальні основи файтингов: обмежена арена, розподіл бою на раунди і нарахування гравцеві очок в залежності від того, яким з доступних ударів він потрапив по противнику. Розквіт файтингов почався з виходом в 1991 році на аркадних автоматах гри Street Fighter II від компанії «Capcom». В Street

Fighter II були представлені практично всі принципи, техніки і системи, на яких базуються сучасні файтинг.

Серед них – можливість виконання складних прийомів шляхом поєднання рухів джойстика («чверть кола вперед», «півкола назад» і т. П.) З натисканням кнопок; переривання анімації одних прийомів в інші, що дозволило виконувати безперервні ланцюжки прийомів і послужило основою для концепції комбо; відносно велика кількість персонажів, що мають різні стилі бою; і деякі інші ідеї ігровий механіки, майже без зміни використовуються в сучасних двомірних файтинга;

Шутер – жанр комп'ютерних ігор. На момент зародження жанру за кордоном зміцнилося слово «шутер», як варіант опису ігрового процесу і переклад для слова shooter, в Росії і деяких інших країнах пострадянського простору жанр спочатку був названий як «стрілялка». Гравець знаходиться в тривимірному просторі і має деяку свободу пересування. Рівні, як правило, являють собою обмежений лабіринт, в якому розташовані вороги, союзники і нейтрально налаштовані NPC. Дії більшості шутерів розгортається в анізотропному просторі (приміщення мають очевидні підлогу і стелю, в них діє гравітація), хоча існують і виключення, в яких простір изотропно (наприклад, Descent). Геймплей канонічного шутера зводиться до пошуку виходу з рівня, з усуненням всіх перешкоджають супротивників і перешкод (пошук ключів до закритих дверей, дистанційне відкриття проходу за допомогою органів управління, віддалених від самих дверей). Такі Wolfenstein 3D, Doom і багато подібних, однак це не є межею певної епохи – канонічні шутери як і раніше випускаються. У розширеному жанрі ставляться додаткові цілі, такі, як установка бомби на вороже укріплення, рішення пазлів за відведений час (System Shock і т. П).

Іноді навіть саме знищення ворогів в чистому вигляді є безглуздим заняттям – необхідно виконати певне завдання і цим задіяти якийсь ігровий тригер, щоб пройти далі. Наочним прикладом такого підходу є Call of Duty.

Другим фактором є лінійність. Лінійність – властивість конкретного рівня, а не ігри в цілому (наприклад, рівень Fortress of Mystery в Doom абсолютно лине і спирається на один тактичний прийом, а саме властивість різних монстрів сваритися один з одним; рівень Unholy Cathedral, навпаки, зразок нелінійності). Лінійними називаються рівні, які проходяться в одному можливому напрямку, а завдання для гравця представляє тільки сам бій (в канонічному шутере) або «бойове завдання». Нелінійні рівні можуть бути пройдені безліччю різних способів, велика кількість приміщень є для відвідування в довільному порядку, а від гравця потрібно не тільки дослідити рівень з метою в ньому зорієнтуватися, але і визначити найбільш тактично вигідний для себе маршрут.

Предки тривимірних шутерів зустрічалися як мінімум від 1985 року (Eidolon від Lucasfilm Games) до 1991 року (Hovertank 3D і Catacomb 3D від id Software), але широку популярність жанру принесла гра Wolfenstein 3D, створена тією ж компанією. Процес перестрілок в Hovertank 3D здався б сучасному гравцеві нестерпно повільним. Завдання гравця була врятувати всіх заручників за певний час. Графічно гра була на рівні свого часу: двох з половиною вимірний опис об'єкта рівні, спрайтові персонажі, бестекстурние стіни. Зброя не відображалося і з'явилося тільки в грі «Catacomb 3-D», також володіла вельми млявою динамікою. З точки зору свободи огляду гравець завжди дивився паралельно землі. Лише в Wolfenstein 3D ігровий процес був доведений до прийнятного рівня, що породило поширена помилка, що саме ця гра є першим представником жанру взагалі. У 1993 році компанія id Software розвинула свої ідеї, створивши ігрову лінійку Doom. У порівнянні з «Wolfenstein 3D» ускладнився дизайн рівнів, місцями змінювався рівень підлоги, а також було покращено якість текстур і спрайтів. Але істотним для жанру зміною стала модифицируемость гри: Doom є однією з перших ігор з підтримкою модифікацій і першим подібним шутером. Ентузіасти з допомогою консолі управління могли додавати нові ігрові рівні, звуки і текстури, які

самі ж і створювали. Відображення гравця також обмежувалося лише зброєю, але було додано погойдування при ходьбі, що посилювало ефект занурення в гру. Можливість модифікувати гри, яку «Doom» привніс в жанр, привернула розробників-ентузіастів, багато з яких пізніше влилися в індустрію виробництва ігор. Так, наприклад, значний відсоток штату компанії Valve становили колишні незалежні «ігромоддери». Як і у випадку з «Wolfenstein 3D», движок, на якому був побудований «Doom» був узятий за основу багатьох ігор сторонніх компаній.

У короткий термін розробники зняли обмеження з камери в одній площині; дали можливість дивитися не тільки справа наліво, але і зверху вниз (Heretic); зробили елементи декорацій інтерактивними, тобто реагують на дії гравця; додали можливість стрибати (Hexen); додавши рольову складову (Strife). Пізніше фірмою Parallax Software був створений і випущений перший повністю тривимірний шутер з елементами космічного симулятора Descent. Для гри настійно рекомендувався джойстик, через що більшість гравців в ті роки банально не могло проходити її, що і зробило її маловідомої серед шутерів. Всупереч думці багатьох, саме Descent був одним з перших повністю тривимірних шутерів від першої особи, а не Quake.

Платформер – жанр комп'ютерних ігор, в яких основною рисою ігрового процесу є стрибання по платформах, лазіння по сходах, збирання предметів, зазвичай необхідних для завершення рівня. Деякі предмети, звані пауер-апамі (англ. Power-up), наділяють керованого гравцем персонажа особливою силою, яка зазвичай вичерпується згодом (наприклад: силове поле, прискорення, збільшення висоти стрибків). Колекційні предмети, зброю і «пауер-ап» збираються зазвичай простим дотиком персонажа і для застосування не вимагають спеціальних дій з боку гравця. Рідше предмети збираються в «інвентар» героя і застосовуються спеціальної командою (така поведінка більш характерно для аркадних головоломок). Подібний жанр комп'ютерних ігор сайд-скроллер. Противники (звані «ворогами»), завжди численні і різно-

рідні, мають примітивним штучним інтелектом, прагнучи максимально наблизитися до гравця, або не володіють їм зовсім, переміщаючись по круговій дистанції або здійснюючи повторювані дії.

Зіткнення з противником зазвичай забирає життєві сили у героя або зовсім вбиває його. Іноді противник може бути нейтралізований або стрибком йому на голову, або зі зброї, якщо ним володіє герой. Смерть живих істот зазвичай зображується спрощено або символічно (істота зникає або провалюється вниз за межі екрану). Рівні, як правило, рясніють секретами (приховані проходи в стінах, високі або важкодоступні місця), перебування яких істотно полегшує проходження і підігріває інтерес гравця. Ігри подібного жанру характеризуються нереалістичністю, мальованої мультяшной графікою. Героями таких ігор зазвичай бувають міфічні істоти (наприклад: дракони, гобліни) або антропоморфні тварини. Платформери з'явилися на початку 1980-х і стали тривимірними ближче до кінця 1990-х. Через деякий час після утворення жанру у нього з'явилося дане назва, що відбиває той факт, що в платформер геймплей сфокусований на стрибках по платформах в противагу стрільбі. Правда, у багатьох платформер присутній стрілецьку зброю, в таких, наприклад, як Blackthorne або Castlevania.

Основні представники платформер: Commander Keen, Magic Pockets, Baby Jo in "Going Home", Bumpy`s Arcade Fantasy, Rick Dangerous, Adventures of Lomax, Electro Man, серії Prince of Persia, Sonic the Hedgehog, Ratchet and Clank, Rockman, Super Mario, Rayman, Spyro the Dragon, Crash Bandicoot, Sly Cooper, LittleBigPlanet. Багато з них стартували в 1980-х і тривають досі. Історично склалося, що в Україні стали популярні твори середини 1990-х, що виходили на Sega Mega Drive і персональних комп'ютерах, особливо гри по мультфільмах Діснея (Chip 'n Dale: Rescue Rangers і т. П.) І Jazz Jackrabbit.

Аркада (англ. Arcade game, arcade genre) – жанр комп'ютерних ігор, що характеризується коротким за часом, але інтенсивним ігровим процесом[4][4)]. У строгому сенсі, аркадной вважається гра для аркадних ігрових автоматів. Також аркадними називають гри, портований з аркадного автомата. До розквіту аркадних ігор відносять період з кінця 1970-х до середини 1980-х, називаючи його золотим століттям аркадних ігор. Назва «аркада» походить від англ. arcade – пасаж, крита галерея магазинів, де зазвичай розташовувалися аркадні ігрові автомати. Однією з перших аркадних ігор є Computer Space, яка була створена в 1971 році. Вона є першою промислової електронної грою (англ.) Рос. даного жанру. Незважаючи на свою назву, яке почало асоціюватися з іграми даного типу тільки в кінці 1970-х. Перші аркадні ігри, такі як Asteroids, Battlezone, Star Castle використовували векторну графіку для рендеринга, коли зображення будувалося з ліній. Після появи кольорової растрової графіки з'явилися мультяшні відеоігри і такі персонажі, як Pac-Man і Donkey Kong, які стали культовими і проявилися як явище культури.

В середині 1980-х аркадні ігри почали своє широке поширення, стаючи більш різноманітними, як щодо інтерфейсу управління (аркадного автомата), так і в плані тематики, графіки і жанрів. Використовувалися більш реалістичні контролери, такі як двомісні космічні кораблі Tail Gunner або управління Феррарі в Out Run. З плином часу аркадні автомати витіснялися домашніми системами, так як їх графіка початку обходити більшість аркадних ігор, і таким чином до кінця 1990-х аркадні автомати покинули індустрію. У 2000-х аркадний жанр почав використовуватися в соціальних і мережевих іграх, коли ігрові центри надавали погодинні послуги комп'ютерних та консольних ігор. Відмінністю було обстановка інтернет-кафе, і задіяння переваг відеозалу. Деякі компанії розробляли гри, пробуючи поєднати аркадний досвід гравців і переваги атракціонного підходу. Наприклад, Disney і Sally Corporation в

---

[4][4)] Rouse, Richard. Game Design: Theory & Practice 2.Los Rios Boulevard, Plano, Texas, USA : Wordware Publishing, 2004.698 c.ISBN 1-55622-912-7.

2008 році в грі Toy Story Midway Mania! (Англ.) для чотирьох гравців використовували гігантські відеоекрани, карнавальний стиль і спецефекти з використанням спрея для створення візуальних ефектів.

Класичні аркади характеризуються такими властивостями:

- гра на одному екрані (В класичних аркадах весь ігровий процес зосереджений на одному екрані);
- нескінченна гра (Потенційно гравці можуть грати в аркаду нескінченне час, і відповідно, не можуть виграти);
- безліч життів (Зазвичай, класична аркада пропонує гравцеві кілька спроб (життів). Такий підхід дозволяє новачкам отримати більшу можливість вивчити ігрові механіки до того, як гра закінчується);
- ігровий рахунок (Практично всі класичні аркади включають в себе ігровий рахунок, коли гравець отримує очки за виконання різних цілей або завдань);
- швидке навчання, простий ігровий процес. (Для класичних аркад характерно те, що гравцям легко навчитися геймплею, але стає практично неможливим стати майстром в грі через її складності. Разом з тим, якщо гравець гине в аркаді, то це практично завжди відбувається з його вини);
- ні сюжету / історії (Класичні аркади практично завжди уникали спроб розповісти якусь історію, і дана тенденція продовжується для сучасних аркад).

Симулятор – імітатор (зазвичай механічний або комп'ютерний), завдання якого полягає в імітації управління будь-яким процесом, апаратом або транспортним засобом. Найчастіше зараз слово «симулятор» використовується стосовно до комп'ютерних програм (зазвичай ігор). За допомогою комп'ютерно-механічних симуляторів, абсолютно точно відтворюють інтер'єр кабіни апарату, тренуються пілоти, космонавти, машиністи високошвидкісних поїздів. Симулятор – програмні і апаратні засоби, що створюють враження дійс-

ності, відображаючи частина реальних явищ і властивостей у віртуальному середовищі. Існують різні види симуляторів:

- технічні (За допомогою комп'ютера імітується фізична поведінка і управління будь-яким складним об'єктом технічної системи (наприклад: бойовим винищувачем, автомобілем і т. Д.). Розрізняють космічні, залізничні, авто-, авіа- симулятори та інші);
- спортивні симулятори (Спортивний симулятор (англ. Sport game; інша назва – «Спортс», від англ. Sport simulator – симулятор спорту). Як і випливає з назви – імітація будь-якої спортивної гри, найбільшого поширення набули імітації футболу, хокею, баскетболу, тенісу і гольфу, боулінгу та більярду);
- спортивні менеджери (Спортивний менеджер поєднує в собі елементи спортивного та економічного симулятора. Відмінною особливістю є те, що в спортивному симуляторі гравець спостерігає безпосередньо за ігровим процесом і може впливати на хід матчу прямо під час його проведення, а в менеджері налаштування тактики, стратегії, трансферів і фінансових операцій вибираються заздалегідь, і гравець переглядає результати вже після матчу);
- симулятор будівництва та управління (англ. Construction and management simulation, CMS) – жанр комп'ютерних ігор, різновид симуляторів життя. В іграх цього жанру користувач будує, розширює і здійснює фінансове управління вигаданими спільнотами або проектами в умовах обмежених ресурсів. В стратегічних іграх іноді можуть бути присутніми елементи симуляторів будівництва і управління, оскільки гравець повинен управляти ресурсами в міру руху до мети. Однак власне симулятори будівництва та управління відрізняються від стратегічних ігор в тому, що «метою гравця є не розгром ворога, а споруда чогось в безперервному процесі». Ігри цього жанру часто

описуються терміном management games (з англ. – «гри про управління»).

Action-adventure, або пригодницький бойовик – змішаний жанр комп'ютерних ігор, що поєднує елементи квесту і екшену. Подібні ігри пропонують гравцеві долати перешкоди як інтелектуального, так і фізичного роду, наприклад, випробування гравця на витривалість або швидкість реакції; саме визначення того, коли така гра перестає бути квестом і перетворюється в чисту action-гру, є лише питанням інтерпретації. До елементів квесту можуть належати головоломки; сюжет, численні персонажі, діалоги між ними, інвентар для зібраних предметів та інші прикмети жанру квестів. При цьому в порівнянні з квестами action-adventure в великій мірі спираються на переміщення персонажа у віртуальному світі гри – це відноситься і до геймплею, в якому можуть переважати битви, і до сюжету: переміщення змушує запускатися сюжетні сцени, відповідно до яких змінюється темп гри і контекст дій гравця. Вид від третьої особи є вкрай поширеним в іграх жанру action-adventure; в порівнянні з чистими action-іграми, квестовиє елементи в action-adventure вимагають більш складного поведінки камери і більш складного пристрою ігрових рівнів.

Це одна з тих різновидів комп'ютерних ігор, яка обов'язково потребує присутності сюжету, а не тільки геймплея. Для action-adventure стандартним є пряме управління персонажем, а не інтерфейс на зразок point-and-click. Бретт Уейсс в книзі Classic Home Video Games називав гру Superman (англ.) Poc. (1979) грою в жанрі action-adventure; гра використовувала надзвичайно складний для того часу сюжет і набір завдань для гравця. Марк Уолв в книзі Video Game Theory Reader віддає пальму першості Adventure (1979), заснованої на текстової грі Colossal Cave Adventure, але містить і битви в реальному часі, як в action-іграх. Іншою ранньою грою в жанрі action-adventure була Castle Wolfenstein (1981). Один з розробників Wizardry Роу Адамс описував ранні гри жанру просто як «аркадні ігри в фентезійному сеттінгу», приводячи

в приклад гри Castlevania (1986), Trojan (англ.) Рос. (1986) і Wizards & Warriors (англ.) Рос. (1987). Тревіс Фас з сайту IGN стверджував, що гра The Legend of Zelda (1986) при своїй схожості з Adventure заклала основи «нового піджанру – action-adventure», поєднуючи в собі елементи різних жанрів – дослідження світу в дусі пригодницьких ігор, битви в дусі action, елементи рольової гри і головоломок.

Комп'ютерна рольова гра (англ. Computer Role-Playing Game, позначається абревіатурою CRPG або RPG) – жанр комп'ютерних ігор, заснований на елементах ігрового процесу традиційних настільних рольових ігор. У рольовій грі гравець управляє одним або декількома персонажами, кожен з яких описаний набором чисельних характеристик, списком здібностей і вмінь; прикладами таких характеристик можуть бути хіт-пойнти (англ. hit points, HP), показники сили, спритності, інтелекту, захисту, ухилення, рівень розвитку того чи іншого навику і т. п. У жанру CRPG багато спільного з настільними рольовими іграми на кшталт Dungeons and Dragons – жаргон, сеттінг, геймплейная механіка. Зазвичай гравець управляє одним або декількома головними героями («партією»), і домагається перемоги, виконуючи завдання («квести»), беручи участь в тактичних боях і доходячи до самого кінця сюжету. Ключова особливість CRPG в тому, що персонажі ростуть в здібностях, і це зростання часто контролюється гравцем. CRPG, за винятком піджанру «action RPG», рідко покладаються на фізичну координацію і реакцію. CRPG зазвичай покладаються на продуманий сюжет і ігровий світ.

Сюжет зазвичай ділиться на серію завдань («квестів»). Гравець віддає персонажу команди, і він виконує їх відповідно до чисельними показниками, що відповідають за якість виконання команди. Коли персонаж набирає певну кількість очок досвіду, він отримує черговий рівень, і ці показники збільшуються. CRPG часто пропонують більш складне і динамічне взаємодія гравця з комп'ютерними персонажами, ніж інші жанри – це може забезпечуватися як розвиненим штучним інтелектом, так і жорстко запрограмованим поведінкою

персонажів. Елементи ігрового процесу RPG зустрічаються і в інших жанрах комп'ютерних ігор: покрокових стратегіях, стратегіях реального часу, шутерах від першої або третьої особи, і т. д.

За цілям і завданням, поставленим перед гравцем під час просування по грі, виділяють кілька напрямків в жанрі комп'ютерних рольових ігор:

- RPG-розповідь (англ. Narrative RPG або англ. Story-driven RPG) побудована навколо якогось оповідання; очікується, що гравець буде захоплений сюжетом гри, персонажами і декораціями і буде слідувати основної сюжетної лінії (англ. main quest), навколо якої будується весь інший ігровий процес. Як приклади таких ігор називаються Mass Effect, Dragon Age: Origins.
- RPG-пісочниця (англ. Sandbox RPG) поміщає ігрового персонажа у відкритий світ, надаючи гравцеві робити те, що йому заманеться. В таких іграх замість основної сюжетної лінії гравцеві пропонується безліч незалежних завдань, місць для відвідування і тому подібного. Як приклади таких ігор називаються серія The Elder Scrolls і Fallout.
- зачистка підземель (англ. Dungeon crawler) ставить на чільне місце розвиток персонажа, перетворюючи підвищення характеристик і придбання нової, більш цінною екіпіровки в самоціль. Сюжети і декорації таких ігор дуже прості і служать не більше ніж приводом для винищення безлічі супротивників і пошуку скарбів. Прикладами таких ігор можуть служити серія Diablo, Torchlight, Dungeon Siege.

Імовірно, першими комп'ютерними рольовими іграми були Dungeon (1975) і dnd (1974). Текстові CRPG виросли з текстових адвенчур і MUDов. Серед перших з них були dnd і Akalabeth (1980), що дала початок відомої серії Ultima. У 1981 році вийшла Wizardry: Proving Grounds of the Mad Overlord, одна з перших графічних рольових ігор, що зробила великий вплив на майбутній піджанр японських рольових ігор. З кінця 1980-х жанр комп'ютерних рольових ігор міцно зміцнився на PC-платформі. В 1985-87 виходять такі іг-

ри як Nethack, ймовірно, найбільш відома з текстових RPG, Bard's Tale, перша гра Interplay, Might and Magic Book One: The Secret of the Inner Sanctum, перша гра серії, довгий час що була еталоном для жанру, Ultima IV, гра з нетривіальною системою моральних цінностей, Dungeon Master (серія Zork). З цього починається золотий вік рольових ігор, кожен рік якого приносив нові якісні ігри в жанрі. Рольові ігри набувають барвистість і інтерактивність, з'являються спроби відійти від зображення ситуації двовимірними картинками. У той же час на консолях випускаються гри, які внесли величезний вклад в розвиток жанру: Final Fantasy, Dragon Quest і Phantasy Star.

І окремо я б хотіла розглянути жанр, в якому робився наш продукт.

Survival horror (з англ. – «жах виживання», альтернативний переклад «виживання в кошмарі») – жанр комп'ютерних ігор, для якого характерними є упор на виживання ігрового персонажа і нагнітання атмосфери страху і тривоги, подібно до літератури і фільмів жахів.

Хоча геймплей таких ігор може включати в себе битви з будь-якими супротивниками, як і в іграх інших жанрів, гравець в survival horror не відчуває того рівня контролю над тим, що відбувається, яка є типовою для більшості екшн-ігор. Це досягається різними обмеженнями – нестачею боєприпасів, низьким рівнем здоров'я протагоніста, швидкістю пересування, видимості, а також різними перешкодами, які ускладнюють взаємодію з ігровою механікою. Найчастіше гравець змушений шукати в грі предмети, які відкривають доступ в нові області гри, вирішувати різні загадки і головоломки. Дизайн рівнів в survival horror часто також використовується для створення атмосфери жаху або очікування чогось страшного – наприклад, ігровий персонаж може обстежити темні похмурі приміщення, що нагадують лабіринт, і піддаватися несподіваним нападам ворогів.

Сама назва жанру survival horror вперше було використано при оригінальному японському виданні гри Resident Evil в 1996 році. На розробку гри вплинули виходили раніше гри, виконані в дусі жахів, такі як Sweet Home

1989 року і Alone in the Dark 1992 року[5][5)]. З того часу цей термін використовується для позначення всіх подібних ігор, а також, ретроспективно, до створених до його появи. Починаючи з ігри 2005 року Resident Evil 4 жанр помітно зблизився з екшн-іграми і більш традиційними шутерами від першої і третьої особи, що будуються насамперед на боях з численними ворогами; гра Amnesia: The Dark Descent (2010) ознаменувала рух жанру в іншому напрямку – до жанру стелс-екшн, де ігровий персонаж повинен тікати і ховатися від ворогів, а не боротися з ними.

Однією з унікальних рис жанру survival horror є те, що він визначається в більшій мірі загальною атмосферою гри, ніж ігровий механікою. Хоча термін survival horror зазвичай асоціюється з геймплеєм перших частин Resident Evil і Silent Hill, існує досить представників жанру, що нагадують побий їх усіх, квести, рольові ігри та шутери від першої особи. Всі ігри цього жанру об'єднують теми страху, непояснених явищ, жорстокості і тому подібного. Сюжет типовою гри в жанрі survival horror включає в себе розслідування деяких таємниць і зіткнення з страшними силами; таким чином, ігри можуть використовувати теми і образи з фільмів і літератури жахів, перетворюючи їх в перешкоди для гравця.

На думку творця гри The Suffering Річарда Рауса, умовності фільмів жахів є дуже привабливими для розробників комп'ютерних ігор взагалі: жахи зіштовхують ігрового персонажа з ворожими йому чудовиськами, роблячи боєм з ними морально виправданим; залишають можливість створення мінімалістичного сюжету і використання світу, наближеного до реального, але з додаванням фантастичного елемента.

Крім того, Раус відзначав, що аудиторія фільмів жахів – переважно молодіжна і переважно чоловіча – добре збігається з аудиторією геймерів. Тим

---

[5][5)] Travis Fahs. IGN Presents the History of Survival Horror URL: https://www.ign.com/aricles/2009/10/30/ign-presents-the-history-of-survival-horror (дата звернення: 17.11.2019)

не менш, важливо, що комп'ютерні ігри відрізняються від фільмів жахів інтерактивністю: якщо у фільмі дія триває незалежно від глядача, то просування по грі знаходиться в руках самого гравця, і йому нема кого звинувачувати в своїх невдачах, крім самого себе. Томас Грип, креативний директор студії Frictional Games, стверджував, що завдання розробників хоррор-ігор – не стільки безпосередньо лякати гравця, скільки поміщати його в такі умови, де гравець лякав би сам себе. Подібно фільмам жахів, ігри survival horror можуть бути як психологічними тріллерами з нагнітанням саспенсу, так і бойовиками, де зроблений упор на знищення будь-яких живих мерців або жертв невдалих експериментів. У 2014 році на сторінках журналу Game Informer його автори Гаррі Макін і Тім Турі обговорювали значення цього терміна і можливість відмовитися від нього, міркуючи про те, що слова survival horror були придумані розробниками конкретної гри – Resident Evil 1996 року – і прижилися як опис ігор, схожих на цю гру; проте слова horror («жахи») цілком достатньо для опису всіх «страшних» ігор, навіть зовсім не нагадують Resident Evil.

У 2014 році журналістка Бріттані Вінсент в колонці для сайту Shacknews виділила кілька піджанрів жахів в комп'ютерних іграх, крім власне survival horror в її розумінні – класичних ігор в дусі Resident Evil і Silent Hill.

Ігри, що позначаються Вінсент як «екшн-хоррор від третьої особи» (Third-Person Action Horror) на зразок Dead Space, більшою мірою є action-іграми, творці яких пожертвували якимись елементами жахів, щоб створити більш приємний ігровий процес. Такі ігри – до їх числа Вінсент зараховує Blue Stinger, Nightmare Creatures, Dead Rising, Manhunt – вже не ставлять в основу виживання, пропонуючи гравцеві більший вибір зброї і більшу кількість ворогів в порівнянні з власне survival horror; вони також пропонують гравцеві велику ступінь фізичної взаємодії з оточенням, наприклад, у вигляді елемен-

тів платформера або необхідності руйнувати якісь об'єкти, щоб просунутися далі.

Схожим чином, шутери-хоррори (shooter horror) представляють собою «страшні» шутери, в яких основним фактором, наганяють жах на гравця, є просто велика кількість супротивників, з якими він повинен боротися – до цього піджанру Вінсент зараховує ігри на зразок Doom, FEAR, Resistance або «Метро 2033».

До категорії стелс-хорроров (stealth horror) Вінсент відносить стелс-ігри, що використовують рівно протилежний підхід до геймплею: в таких іграх, як Penumbra і Amnesia: The Dark Descent, Slender: The Arrival, Outlast, SCP: Containment Breach або Alien: Isolation гравець мало що може протиставити противникам і повинен тікати або ховатися від них.

Окремо в класифікації Вінсент стоять гри, засновані на японському фольклорі, на зразок Project Zero, Forbidden Siren, Kuon, Rule of Rose, Clock Tower – ці ігри часто містять сюрреалістичні елементи, досліджують теми зв'язків і відносин між людьми, експлуатують фольклорні уявлення про привидів і демонів -вони.

3 ЗАГАЛЬНИЙ ОПИС ТЕСТУВАННЯ ПРОГРАМНОГО ПРОДУКУ

3.1 Тестування Програмного Забезпечення

Згідно (ISO / IEC TR 19759: 2005), тестування програмного забезпечення – процес дослідження, випробування програмного продукту, який має на меті перевірку відповідності між реальною поведінкою програми і її очікуваним поведінкою на кінцевому наборі тестів, обраних певним чином.

У різний час і в різних джерелах тестування давалися різні визначення, в тому числі:

- процес виконання програми з метою знаходження помилок;
- інтелектуальна дисципліна, що має на меті одержання надійного програмного забезпечення без зайвих зусиль на його перевірку;
- технічне дослідження програми для отримання інформації про її якість з точки зору певного кола зацікавлених осіб;
- перевірка відповідності між реальною поведінкою програми і її очікуваним поведінкою на кінцевому наборі тестів, виконаних певним чином;
- процес спостереження за виконанням програми в спеціальних умовах і винесення на цій основі оцінки будь-яких аспектів її роботи;
- процес, який має на меті виявлення ситуацій, в яких поведінка програми є неправильним, небажаним або не відповідає специфікації;
- процес, що містить у собі всі активності життєвого циклу, як динамічні, так і статичні, що стосуються планування, підготовки та оцінки програмного продукту і пов'язаних з цим результатів робіт з метою

визначити, що вони відповідають описаним вимогам, показати, що вони підходять для заявлених цілей і для визначення дефектів.

Перші програмні системи розроблялися в рамках програм наукових досліджень або програм для потреб міністерств оборони. Тестування таких продуктів проводилося строго формалізовано із записом всіх тестових процедур, тестових даних, отриманих результатів. Тестування виділялося в окремий процес, який починався після завершення кодування, але при цьому, як правило, виконувалося тим же персоналом.

Протягом десятиліть розвитку розробки ПЗ до питань тестування і забезпечення якості підходили дуже і дуже по-різному. Можна виділити кілька основних «епох тестування».

У 50-60-х роках минулого століття процес тестування був гранично формалізован, відділений від процесу безпосередньої розробки ПЗ і «математизований ». Фактично тестування представляло собою скоріше налагодження програм (Debugging). Існувала концепція «Вичерпного тестування»

(Exhaustive testing) – перевірки всіх можливих шляхів виконання коду з усіма можливими вхідними даними. Але дуже скоро було з'ясовано, що вичерпне тестування неможливо, тому що кількість можливих шляхів і вхідних даних дуже велике, а також при такому підході складно знайти проблеми в документації.

У 1960-х багато уваги приділялося «вичерпного» тестування, яке повинно проводитися з використанням всіх шляхів в коді або всіх можливих вхідних даних. Було відзначено, що в цих умовах повне тестування програмного забезпечення неможливо, тому що, по-перше, кількість можливих вхідних даних дуже велике, по-друге, існує безліч шляхів, по-третє, складно знайти проблеми в архітектурі і специфікаціях. З цих причин «вичерпне» тестування було відхилено і визнано теоретично неможливим.

На початку 1970-х років тестування програмного забезпечення позначалося як «процес, спрямований на демонстрацію коректності продукту» або

як «діяльність по підтвердженню правильності роботи програмного забезпечення». У зароджувалася програмної інженерії верифікація ПЗ значилася як «доказ правильності». Хоча концепція була теоретично перспективною, на практиці вона вимагала багато часу і була недостатньо всеохоплюючою. Було вирішено, що доказ правильності – неефективний метод тестування програмного забезпечення. Однак, в деяких випадках демонстрація правильної роботи використовується і в наші дні, наприклад, приймально-здавальні випробування.

У другій половині 1970-х тестування уявлялося як виконання програми з наміром знайти помилки, а не довести, що вона працює. Успішний тест – це тест, який виявляє раніше невідомі проблеми. Даний підхід прямо протилежний попередньому. Зазначені два визначення є «парадокс тестування», в основі якого лежать два протилежних твердження: з одного боку, тестування дозволяє переконатися, що продукт працює добре, а з іншого – виявляє помилки в програмах, показуючи, що продукт не працює. Друга мета тестування є більш продуктивною з точки зору поліпшення якості, так як не дозволяє ігнорувати недоліки програмного забезпечення.

У 70-х роках фактично народилися дві фундаментальні ідеї тестування: тестування спочатку розглядалося як процес докази можливості програми в деяких заданих умовах (positive testing), а потім – строго навпаки: як процес докази непрацездатності програми в деяких заданих умовах (negative testing). Ця внутрішня суперечність не тільки не зникла з часом, але і в наші дні багатьма авторами зовсім справедливо відзначається як дві взаємодоповнюючі мети тестування.

У 1980-і роки тестування розширилося таким поняттям, як попередження дефектів. Проектування тестів – найбільш ефективний з відомих методів попередження помилок. В цей же час стали висловлюватися думки, що необхідна методологія тестування, зокрема, що тестування має включати перевірки на всьому протязі циклу розробки, і це повинен бути керований про-

цес. В ході тестування треба перевірити не тільки зібрану програму, а й вимоги, код, архітектуру, самі тести. «Традиційне» тестування, яке існувало до початку 1980-х, відносилося тільки до компільованою, готової системі (зараз це зазвичай називається системне тестування), але в подальшому тестувальники стали залучатися в усі аспекти життєвого циклу розробки.

Це дозволяло раніше знаходити проблеми у вимогах та архітектури і тим самим скорочувати терміни і бюджет розробки. В середині 1980-х з'явилися перші інструменти для автоматизованого тестування. Передбачалося, що комп'ютер зможе виконати більше тестів, ніж людина, і зробить це більш надійно. Спочатку ці інструменти були вкрай простими і не мали можливості написання сценаріїв на скриптових мовах.

У 80-х роках відбулося ключова зміна місця тестування в розробці ПЗ: замість однієї з фінальних стадій створення проекту тестування стало застосовуватися протягом усього циклу розробки (software lifecycle), що дозволило в дуже багатьох випадках не тільки швидко виявляти й усувати проблеми, але навіть передбачати і запобігати їх появі.

В цей же період часу відзначено бурхливий розвиток і формалізація методологій тестування і поява перших елементарних спроб автоматизувати тестування.

На початку 1990-х років в поняття «тестування» стали включати планування, проектування, створення, підтримку і виконання тестів і тестових оточень, і це означало перехід від тестування до забезпечення якості, що охоплює весь цикл розробки програмного забезпечення. У цей час починають з'являтися різні програмні інструменти для підтримки процесу тестування: більш просунуті середовища для автоматизації з можливістю створення скриптів і генерації звітів, системи управління тестами, ПЗ для проведення навантажувального тестування. В середині 1990-х років з розвитком Інтернету і розробкою великої кількості веб-додатків особливу популярність стало

отримувати «гнучке тестування» (за аналогією з гнучкими методологіями програмування).

У 90-х роках відбувся перехід від тестування як такого до більш всеосяжний процесу, який називається «забезпечення якості (quality assurance), охоплює весь цикл розробки ПЗ і зачіпає процеси планування, проектування, створення і виконання тест-кейсів, підтримку наявних тест-кейсів і тестових оточень. Тестування вийшло на якісно новий рівень, який природним чином привів до подальшого розвитку методологій, появи досить потужних інструментів управління процесом тестування і інструментальних засобів автоматизації тестування, вже цілком схожих на своїх нинішніх нащадків.

У нульові роки нинішнього століття розвиток тестування тривало в контексті пошуку все нових і нових шляхів, методологій, технік і підходів до забезпечення якості. Серйозний вплив на розуміння тестування мала поява гнучких методологій розробки та таких підходів, як «розробка під керуванням тестуванням (test-driven development, TDD). Автоматизація тестування вже сприймалася як звичайна невід'ємна частина більшості проектів, а також стали популярні ідеї про те, що на чільне процесу тестування слід ставити не відповідність програми вимогам, а її здатність надати кінцевому користувачеві можливість ефективно вирішувати свої завдання.

Про сучасні етапи розвитку тестування я буду говорити на протязі всього іншого матеріалу.

Якщо ж зазначити коротко основні характеристики, то вийде приблизно такий список:

- гнучкі методології і гнучке тестування;
- глибока інтеграція з процесом розробки;
- широке використання автоматизації;
- колосальний набір технологій та інструментальних засобів;
- кроссфункціональность команди (коли тестувальник і програміст багато в чому можуть виконувати роботу один одного).

3.2 Визначення поняття тестування ПЗ

Тестування програмного забезпечення – перевірка відповідності між реальним і очікуваним поведінкою програми, що здійснюється на кінцевому наборі тестів, обраному певним чином.

Тестування – це одна з технік контролю якості, яка включає в себе такі процеси, як проектування тестів, виконання тестування і аналіз отриманих результатів.

Загальна схема тестування (рис.3.1):

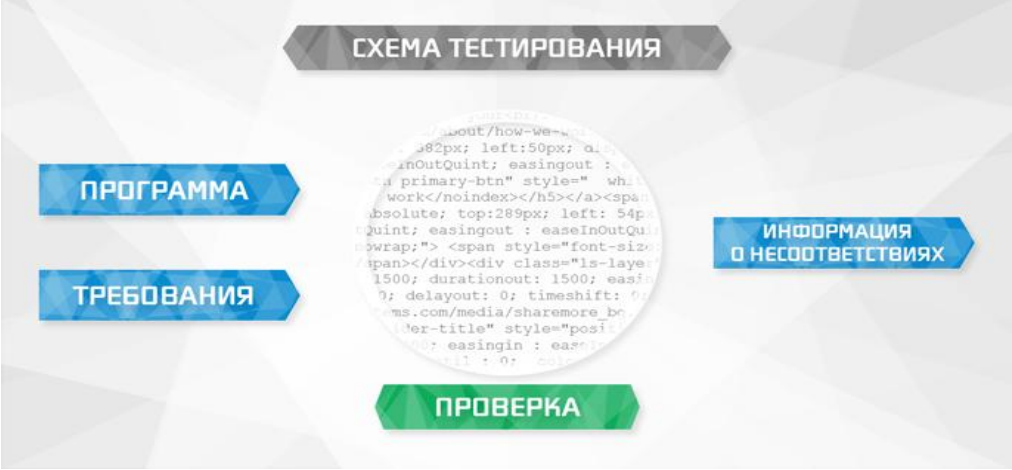

Рисунок 3.1 – Загальна схема тестування

На вході тестувальник отримує програму, яку необхідно тестувати і вимоги. Спостерігаючи за програмою в певних умовах, на виході тестувальник отримує інформацію про відповідність або невідповідність програми вимогам. Дана інформація використовується для виправлення помилок в існуючому продукті, любо для зміни вимог до ще тільки розробляється продукту.

Тест (перевірка) включає в себе обрану певним чином штучно створену ситуацію і опис спостережень, які потрібно здійснити, для перевірки програми на відповідність певним вимогам.

Тест може бути як коротким, так і довгим, наприклад тест продуктивності, перевіряючий працездатність системи при тривалому навантаженні.

Таким чином, в процесі тестування тестувальник виконує дві основні задачі.

Першим завданням є управління виконанням програми, а також створення штучних ситуацій, в яких і відбувається перевірка поведінки програми.

Друге завдання полягає в спостереженні за тим, як програма веде себе в різних створених ситуаціях, і в порівнянні того, що він бачить з тим, що очікується.

Якщо розглядати завдання сучасного тестування, то можна прийти до висновку, що вони полягають не тільки у виявленні помилок в програмах, а й у виявленні причин, за якими вони виникають. Такий підхід до процесу тестування дозволяє розробникам виконувати свою роботу з максимальною ефективністю, усуваючи виявлені помилки швидко і своєчасно.

3.3 Класифікація видів тестування

При тестуванні програмного продукту застосовують велику кількість різних видів тестів. Найбільш широку і детальну класифікацію запропонував автор книги «Тестування Дот Ком» Роман Савін. Він об'єднав види тестування за такими ознаками, як об'єкт, суб'єкт тестування, рівень, позитивність тестування, і ступінь автоматизації тестування. Класифікація була доповнена на підставі таких джерел, як книга Сема Канера, «Тестування програмного забезпечення»[6][6)] та інтернет-ресурс, присвячений тестуванню, «Про Тестінг – Тестування Програмного Забезпечення»[7][7)].

---

[6][6)] Джек Фолк, Сем Канер, Енг. Кек Нгуєн. Тестування програмного забезпечення. Видавництво ДиаСофт, 2001.

[7][7)] Види Тестування URL: http://www.protesting.ru/testing/testtypes.html (дата звернення: 19.11.2019)

По об'єкту тестування діляться на:

Функціональне тестування. Функціональне тестування на сьогоднішній день є одним з найбільш часто вживаних видів тестування. Завдання такого тестування – це встановити на скільки відповідає розроблене програмне забезпечення (ПЗ) вимогам замовника з точки зору функціоналу. Інакше кажучи, проведення функціональних тестів дозволяє перевірити здатність інформаційної системи вирішувати завдання користувачів.

Нефункціональне тестування. Дозволяє перевірити відповідність властивостей програмного забезпечення з поставленими нефункціональними вимогами. Таким чином, нефункціональне тестування – це тестування всіх властивостей програми, що не відносяться до функціональності системи.

Такими властивостями можуть бути пред'явлені характеристики з точки зору таких параметрів як:

- надійність (здатність системи реагувати на непередбачені ситуації);
- продуктивність (здатність системи працювати під великими навантаженнями);
- зручність (дослідження зручності роботи користувача з додатком);
- масштабованість (можливість масштабувати додаток як вертикально, так і горизонтально);
- безпека (дослідження можливості порушення роботи програми та крадіжки призначених для користувача даних зловмисниками);
- портативність (можливість перенести додаток на певний набір платформ) І багато інших якостей.

Тестування для користувача інтерфейсу. Це тестування коректності відображення елементів призначеного для користувача інтерфейсу на різних пристроях, правильності реагування на них на вчинення користувачем різних

дій наскільки і оцінка того, наскільки очікувано поводиться програма в цілому. Таке тестування дає можливість оцінити, наскільки ефективно користувач зможе працювати з додатком і наскільки зовнішній вигляд програми відповідає затвердженим документам, створеними дизайнерами.

При проведенні тестування користувальницького інтерфейсу основним завданням тестувальника є виявлення візуальних і структурних недоліків в графічному інтерфейсі додатка, перевірці можливості і зручності навігації в додатку і коректність обробки додатком введення даних з клавіатури, миші та інших пристроїв введення. Тестування для користувача інтерфейсу необхідно для того, щоб переконатися в тому, що інтерфейс відповідає затвердженим вимогам і стандартам, і гарантувати можливість роботи користувача з графічним інтерфейсом програми.

Тестування зручності використання. Це спосіб тестування, що дозволяє оцінити ступінь зручності використання програми, швидкість навчання користувачів при роботі з програмою, а також наскільки користувачі продукту, що розробляється знаходять її зрозумілою і привабливою в контексті заданих умов. Таке тестування необхідно для забезпечення максимально позитивного користувацького досвіду при роботі з додатком.

Тестування захищеності. Дозволяє виявити головні уразливості програмного забезпечення по відношенню до різних атак з боку зловмисників. Комп'ютерні системи досить часто піддаються кібер атак з метою порушення працездатності інформаційної системи або крадіжки конфіденційних даних. Тестування безпеки дає можливість проаналізувати реальну реакцію і дієвість захисних механізмів, використаних в системі, при спробі проникнення. В процесі тестування безпеки тестувальник намагається виконувати ті ж дії, які виконував би справжній зломщик. При спробі тестувальником зламати систему можуть використовуватися будь-які засоби: атаки системи за допомогою спеціальних утиліт; спроби дізнатися логіни і паролі за допомогою зовнішніх засобів; DDOS атаки; цілеспрямована генерація помилок для вияв-

лення можливості проникнення в систему в процесі її відновлення; використання відомих незакритих уразливостей системи.

Інсталяційне тестування. Під цим терміном мають на увазі тестування коректності установки (інсталяції) певного програмного продукту. Таке тестування зазвичай відбувається в штучно створених середовищах з метою виявити ступінь готовності програмного забезпечення до експлуатації. Основні причини проведення таких тестів пов'язані з необхідністю перевірити коректність поведінки програмного продукту при автоматизованому розгортанні або оновленні. Забезпечення правильної та стабільної установки програмного забезпечення є дуже важливим фактором при створенні програмного продукту, оскільки дозволяє користувачам швидше і з меншими зусиллями почати використовувати продукт, при цьому забезпечуючи однаково коректну поведінку цього продукту у всіх протестованих програмних середовищах.

Конфігураційне тестування. Конфігураційне тестування призначене для оцінки працездатності програмного забезпечення при різноманітних конфігураціях системи. Залежно від типу тестованого програмного продукту, конфигурационное тестування може переслідувати різні цілі. Зазвичай це або визначення оптимальної конфігурації обладнання, що забезпечує достатні для роботи ПО параметри продуктивності, або перевірка певної конфігурації устаткування (або платформи, що включає в себе крім обладнання, стороннє ПО, необхідне для роботи програми) на сумісність з тестованим продуктом. Якщо мова йде про клієнт-серверному програмному забезпеченні, то конфигурационное тестування проводиться окремо для сервера і окремо для клієнта. Зазвичай при тестуванні сумісності сервера з певною конфігурацією стоїть завдання знайти оптимальну конфігурацію, оскільки важлива стабільність роботи і продуктивність сервера. У той час як при тестуванні клієнта, навпа-

ки, намагаються виявити недоліки ПО при будь-яких конфігураціях і по можливості усунути їх[8][8)].

Тестування надійності і відновлення після збоїв (стресове тестування). Такий вид тестування досить часто проводиться для програмного забезпечення, що працює з цінними даними користувачів, безперебійність роботи і швидкість відновлення після збоїв якого критичні для користувача. Тестування на відмову і відновлення здійснює перевірку здатності програми швидко і успішно відновлюватися після відмови обладнання, перебоїв мережі або критичних помилок в самому програмному забезпеченні. Це дає можливість оцінити можливі наслідки відмови і час, необхідний для подальшого відновлення системи. На основі отриманих в ході тестування даних може бути оцінена надійність системи в цілому, і, за умови незадовільних показників, відповідні заходи, спрямовані на поліпшення систем відновлення, можуть бути прийняті

Тестування локалізації. Тестування локалізації дає можливість з'ясувати наскільки добре пристосований продукт для населення певних країн і наскільки він відповідає її культурним особливостям. Зазвичай, розглядаються культурний і мовний нюанси, а саме переклад користувацького інтерфейсу, супутньої документації та файлів на певну мову, також тестується правильність форматів валют, чисел, часу і телефонних номерів.

Тестування навантаження. Тестування навантаження дозволяє виявити максимальну кількість однотипних завдань, які програма може виконувати паралельно. Найпопулярніша мета навантажувального тестування в контексті клієнт-серверних додатків – це оцінити максимальну кількість користувачів, які зможуть одночасно користуватися послугами програми.

---

[8][8)] Что такое Конфигурационное тестирование URL: http://software-testing.org/testing/chto-takoe-konfiguracionnoe-testirovanie-configuration-testing.html (дата звернення: 21.11.2019)

Тестування стабільності. Тестування стабільності перевіряє працездатність додатки при тривалому використанні на середніх навантаженнях. Залежно від типу програми, формуються певні вимоги до тривалості його безперебійної роботи. Тестування стабільності прагне виявити такі недоліки додатки як витоку пам'яті, наявність яскраво виражених стрибків навантаження та інші фактори, здатні перешкодити роботі програми протягом тривалого періоду часу.

Об'ємне тестування. Завданням об'ємного тестування поставлено виявлення реакції додатка і оцінка можливих погіршень в роботі ПО при значному збільшенні кількості даних в базі даних програми. Зазвичай в таке тестування входить:

- замір часу виконання операцій, пов'язаних з отриманням або зміною даних БД при певній інтенсивності запитів;
- виявлення залежності збільшення часу операцій від обсягу даних в БД;
- визначення максимальної кількості користувачів, які мають можливість одночасно працювати з додатком без помітних затримок з боку БД.

Тестування масштабованості. Це вид тестування програмного забезпечення, призначений для перевірки здатності продукту до збільшення (іноді до зменшення) масштабів певних нефункціональних можливостей. Деякі види додатків повинні легко масштабуватися і, при цьому, зрозуміло, залишатися працездатними і витримувати певну призначену для користувача навантаження.

Тестування, пов'язане зі змінами. Саніті є одним з видів тестування, метою якого служить доказ працездатності конкретної функції або модуля відповідно до технічних вимог, заявлених замовником. Санітарне тестування досить часто використовується при перевірці якоїсь частини програми або програми при внесенні в неї певних змін з боку факторів навколишнього се-

редовища. Даний вид тестування зазвичай виконується в ручному режимі. Димові тестування являє собою короткий цикл тестів, метою яких є підтвердження факту запуску і виконання функцій встановлюваного програми після того як новий або редагований код пройшов збірку. По завершенні тестування найбільш важливих сегментів програми надається об'єктивна інформація про присутність або відсутність дефектів в роботі тестованих сегментів. За результатами димового тестування приймається рішення про відправку додатки на доопрацювання або про необхідність його подальшого повного тестування.

Регресійне тестування – тестування, спрямоване на виявлення помилок в уже протестованих ділянках. Регресійне тестування перевіряє продукт на помилки, які могли з'явитися в результаті додавання нової ділянки програми або виправлення інших помилок. Мета даного виду тестування – переконатися, що оновлення збірки або виправлення помилок не спричинило за собою виникнення нових багів.

За рівнем тестування діляться на:

Модульне тестування (Unit тести). Полягає в перевірці кожного окремого модуля (самобутнього елементу системи) шляхом запуску автоматизованих тестів в штучному середовищі. Реалізації таких тестів часто використовують різні заглушки і драйвери для імітації роботи реальної системи. Модульне автоматизоване тестування – це найперша можливість запустити і перевірити вихідний код. Створення Unit тестів для всіх модулів системи дозволяє дуже швидко виявляти помилки в коді, які можуть з'явитися в ході розробки[9][9)].

Інтеграційний тестування. Це тестування окремих модулів системи на предмет коректної взаємодії. Основна мета інтеграційного тестування - знай-

[9][9)] Основные положения тестирования Хабрахабр. URL: https://habrahabr.ru/post/110307/ (дата звернення: 22.11.2019)

ти дефекти і виявити некоректну поведінку, пов'язане з помилками в інтерпретації або реалізації взаємодії між модулями.

Системне тестування. Це тестування програми в цілому, таке тестування перевіряє відповідність програми заявленим вимогам.

Приймальне тестування. Це комплексне тестування, що визначає фактичний рівень готовності системи до експлуатації кінцевими користувачами. Тестування проводиться на підставі набору тестових сценаріїв, що покривають основні бізнес-операції системи.

За виконанням коду діляться на:

Статична тестування. Це виявлення артефактів, що з'являються в процесі розробки програмного продукту шляхом аналізу вихідних файлів, таких як документація або програмний код. Таке тестування проводиться без безпосереднього запуску коду, якість вихідних файлів і їх відповідність вимогам оцінюються вручну, або з використанням допоміжних інструментів. Статична тестування повинне проводитися до динамічного тестування, таким чином, помилки, виявлені на етапі статичного тестування, обійдуться дешевше. З точки зору вихідного коду, статичне тестування виражається в ревізії коду. Зазвичай ревізія коду окремих файлів проводиться після кожної зміни цих файлів програмістом, сама ж ревізія може проводитися як іншим програмістом, так і провідним розробником, або окремим працівником, які займаються ревізією коду. Використання статичного тестування дає можливість підтримувати якість програмного забезпечення на всіх стадіях розробки і зменшує час розробки продукту.

Динамічне тестування. На відміну від статичного тестування, такий вид тестування передбачає запуск вихідного коду програми. Таким чином, динамічне тестування містить в собі безліч інших типів тестування, які вже були описані вище. Динамічне тестування дозволяє виявити помилки в поведінці програми за допомогою аналізу результатів її виконання. Виходить, що майже всі існуючі типи тестування відповідають класу динамічного тестування.

По суб'єкту тестування діляться на:

Альфа-тестування. Це тестування проводиться для найбільш ранніх версій комп'ютерного програмного забезпечення (або апаратного пристрою). Альфа-тестування майже завжди проводиться самими розробниками ПЗ. В процесі альфа-тестування розробники програми знаходять і виправляють помилки і проблеми, наявні в програмі. Зазвичай, під час Альфа-тестування відбувається імітація роботи з програмою штатними розробниками, рідше має місце реальна робота як потенційних користувачів, так і замовників з продуктом. Як правило, альфа-тестування проводиться на самому ранньому етапі розробки ПО, однак в окремих випадках може бути застосоване для закінченого або близького до завершення продукту, наприклад, в якості приймального тестування.

Бета-тестування. Тестування продукції, як і раніше знаходиться в стадії розробки. При бета-тестуванні цей продукт надається для деякої кількості користувачів, для того щоб вивчити і повідомити про виникаючі проблеми, з якими стикаються користувачі. Таке тестування необхідно щоб знайти помилки, які розробники могли пропустити. Зазвичай бета-тестування проводиться в дві фази: закритий бета-тест і відкрите бета-тестування.

Закритий бета-тест – це тестування на строго обмеженому колу обраних користувачів. Такими користувачами можуть виступати знайомі розробників, або їх колеги, які не пов'язані безпосередньо з розробкою тестованого продукту. Відкрите бета-тестування полягає в створенні і розміщенні у відкритому доступі публічної бета-версії. В даному випадку будь-який користувач може виступати бета-тестером. Зворотній зв'язок від таких бета-тестерів здійснюється за допомогою відгуків на сайті і вбудованих в програму систем аналітики і логування призначених для користувача дій, ці системи необхідні для аналізу поведінки користувачів і виявлення труднощів і помилок, з якими вони стикаються.

За позитивності сценарію діляться на:

Позитивне тестування. Тести з позитивним сценарієм перевіряють здатність програми виконувати закладений в неї функціонал. Як правило, для такого тестування розробляються тестові сценарії, при виконанні яких, в нормальних для ПО умовах роботи, не повинно виникати ніяких складнощів.

Негативне тестування. Негативний тестування програмного забезпечення відбувається на сценаріях, відповідних нештатному поведінки програми. Такі тести перевіряють коректність роботи програми в екстрених ситуаціях. Це дозволяє упевнитися в тому, що програма видає правильні повідомлення про помилки, не пошкоджує призначені для користувача дані і поводиться коректно в цілому при ситуаціях, в яких не передбачено штатний поведінка продукту. Основна мета негативного тестування – це перевірити стійкість системи до різних впливів, здатність правильно валідувати вхідні дані і обробляти виняткові ситуації, що виникають як у самих програмних алгоритмах, так і в бізнес-логікою.

За ступенем автоматизації діляться на:

Ручне тестування. Ручне тестування проводиться без використання додаткових програмних засобів, воно дозволяє перевірити програму або сайт за допомогою імітації дій користувача. У цій моделі тестувальник виступає в якості користувача, дотримуючись певних сценаріями, паралельно аналізуючи висновок програми і її поведінка в цілому.

Автоматизоване тестування. Таке тестування дозволяє за рахунок використання додаткового програмного забезпечення для автоматизації тестів значно прискорити процес тестування. Таке додаткове ПО дозволяє контролювати і управляти виконанням тестів і порівнювати очікуваний і фактичний результати роботи програми.

3.4 Процес тестування

Тестування є процес перевірки того, наскільки програмне забезпечення відповідає вимогам, заявленим замовником. Він здійснюється в спеціальних, штучно створюваних ситуаціях за допомогою спостереження за роботою програмного забезпечення. Такого роду штучно побудовані ситуації називають тестовими або просто тестами.

Розробка тестів відбувається на основі перевірених вимог і критерію повноти тестування. Розроблені тести формуються в тест-кейс (набір тестів) і виконуються на ПО, яке потрібно протестувати. Після прогону всіх тестів аналізується результат, в результаті чого можна виявити помилки в програмі.

Процес тестування схематично показаний на (рис 3.3)

Рисунок 3.3 – Процес тестування

3.5 Розробка тест-кейсів

Тест-кейс (тестовий випадок) – це мінімальний елемент тестування (одна перевірка), що містить в собі опис конкретних дій, умов і параметрів, які спрямовані на перевірку будь-якої функціональності. Набір тест-кейсів називається тестовим набором (test suite).

Тест-кейси дозволяють тестувальникам провести перевірку продукту без повного ознайомлення з документацією. За умови, що створений тест-кейс зручний в підтримці, то, написаний один раз, він дозволить заощадити велику кількість часу і зусиль тестувальників. Докладні тест-кейси також здатні істотно знизити варіативність виконання тестів різними тестувальниками, що підвищує якість тестування[10][10)].

Атрибути тест-кейса.

Тест-кейс повинен включати в себе[11][11)]:

- унікальний ідентифікатор тест-кейса. Цей ідентифікатор необхідний для зручності організації та навігації по тестовим набором;
- назва. У назві повинна відображатися основна ідея тест-кейса, мета даної перевірки;
- передумови. Список кроків, які не мають прямого відношення до перевіряється функціоналу, які необхідно виконати до початку тесту. Наприклад, для тест-кейса «Замовлення товару» передумовою може бути крок «авторизуватися на сайті», якщо на даному сайті замовити товар може тільки авторизований користувач;
- кроки. Опис послідовності дій, яка повинна привести до очікуваного результату;
- очікуваний результат. Поведінка системи, передбачене вимогами. Один тест-кейс перевіряє одну конкретну функцію, тому очікуваний результат може бути тільки один;
- статус кейса. Проставляється відповідно до того, чи відповідає фактичний результат очікуваному. Тест-кейс може мати один з трьох статусів:

---

[10][10)] Ron Patton. Software Testing. 2005.

[11][11)] Савин Роман. Тестирование DOT COM. Издательство Дело, 2007.

- позитивний, якщо фактичний результат збігається з очікуваним результатом;
- негативний, якщо фактичний результат не збігається з очікуваним результатом. Якщо статус кейса негативний-знайдена помилка;
- виконання тесту блоковано, якщо після одного з кроків продовження тесту неможливо. У цьому випадку так само, знайдена помилка;
- історія редагування. Дає можливість дізнатися, ким і коли був змінений тест-кейс. Це зручно, оскільки дозволяє більш ефективно редагувати тест-кейси.

Для вимірювання покриття вимог, вимоги до продукту повинні бути проаналізовані і згодом розбиті на пункти. Якщо тест-кейси покривають всі вимоги, то може бути дана позитивна або негативна відповідь про реалізацію даної вимоги в продукті.

Тест є хорошим в разі, коли він може забезпечити високу ймовірність виявлення помилки. Показати, що в програмі повністю відсутні помилки неможливо, тому процес тестування повинен бути спрямований на виявлення раніше не знайдених помилок.

Чіткі, однозначні формулювання кроків. Опис кроків для проходження тест-кейса має містити всю необхідну інформацію, але при цьому тест-кейс не повинен бути надто деталізований. Наприклад, якщо тест-кейс містить такі кроки, як авторизація, в описі необхідно вказувати логін і пароль, але не потрібно вказувати в якому кутку екрану знаходиться вікно авторизації.

Відсутність залежностей тест-кейсів. Якщо тести пов'язані між собою, стає проблематичним зміна, доповнення або видалення конкретного тест-кейса, з'являється необхідність змінювати пов'язані з ним тести. Більш того, взаємопов'язані тести мають конкретний сценарієм від переходу одного тесту до іншого. Це призведе до того що не всі сценарії переходу від одного тесту до іншого будуть протестовані і з'являється ймовірність пропустити баг.

Очікуваний результат необхідно прогнозувати заздалегідь і прописувати його в тест-кейсі. Якщо очікуваний результат не визначити заздалегідь, може виникнути ситуація, коли тестувальник бачить те, що він хоче побачити. При заздалегідь визначеному результаті тестувальника необхідно тільки порівняти очікуваний результат з фактичним[12][12)].

Необхідно приділяти увагу не тільки тестів, які перевіряють правильні дані, але і тим тестам, які перевіряють роботу програми при неправильних, непередбачених даних. Велика кількість помилок пов'язано саме з тими діями користувача, які не передбачені програмою.

Також необхідно перевіряти, чи не робить програма то, чого не повинна. Потрібно проводити перевірку на небажані побічні ефекти.

3.6 Аналіз результатів тестування

Незважаючи на існування різних видів тестування, процеси тестування досить схожі. Розробкою і аналізом тестів може займатися тільки тестувальник. За виконання тест-кейсів так само відповідає тестувальник, однак виконання цих тестів може проводитися як вручну, так і в автоматизованому режимі.

За результатами виконання кожного тесту, йому присвоюється статус (позитивний, негативний, блокований). Якщо тест отримує негативний статус, то в залежності від методології тестування тестувальник може проводити додаткову роботу для виявлення конкретної помилки, яка була причиною некоректної поведінки програми[13][13)].

При використанні методології чорного ящика, аналіз результатів тесту зводиться до виявлення загальних закономірностей, що ведуть до появи по-

---

[12][12)] Рекс Блек. Ключевые процессы тестирования - М.: Издательство Ло-ри, 2014. 544 с.

[13][13)] Тестирование. Фундаментальная теория URL: https://habr.com/ru/post/279535/ (дата звернення: 25.11.2019)

милки. Однак коли використовується білий або сірий ящики, тестувальник може проводити набагато глибший аналіз причин виникнення помилки. Залежно від доступних тестувальника даних (база даних, вихідний код програми, логи і т.п.), він здатний з деякою точністю визначити джерело некоректної поведінки програми.

Після того, як максимально точно виявлена причина небажаної поведінки, тестувальник повинен описати її разом зі способом відтворення помилки для подальшої передачі цієї інформації розробникам. Коли джерело помилки точно визначений і добре описаний, розробникам набагато простіше виправити цю помилку.

4 ПРОВЕДЕННЯ ДОСЛІДЖЕННЯ РІЗНОВИДІВ БАГІВ

4.1 Класифікація помилок

Мета тестування ігор полягає в виявленні багів в грі, щоб зменшити їх кількість до моменту релізу продукту. Баги – це несподіваний результат роботи коду в тестованої версії гри.

Баги можуть варіюватися: починаючи від тих, що заважають гравцеві далі рухатися по сюжету гри і закінчуючи чимось менш значним, наприклад, вихід тексту за кордон кнопки.

Щоб полегшити життя розробникам, баги в іграх класифікуються за їх серйозності («Severity»). Ранжування типів багів в іграх допомагає розробникам зрозуміти, які баги слід пофіксити в першу чергу.

Залежно від ступеня впливу на систему розрізняються такі ки[14][14)]:

- блокуюча (Blocker) – Блокуюча помилка, що приводить гру в неробочий стан, в результаті якого подальша робота з тестованим функціоналом стає неможлива. Рішення проблеми необхідно для подальшого функціонування гри;
- критична (Critical) – Критична помилка, неправильно працює ключова ігрова логіка, діра в безпеці, проблема, яка призвела до тимчасового падіння сервера або приводить в неробочий стан деяку частину функціоналу, без можливості вирішення проблеми, використовуючи інші вхідні точки. Рішення проблеми необхідно для подальшої роботи з ключовими функціями тестируемой гри;
- значна (Major) – Значна помилка, частина основної логіки не функціонує належним чином. Помилка не критична або є можливість працювати з тестованої функцією, використовуючи інші вхідні точки;
- незначна (Minor) – Незначна помилка, що не порушує логіку тестируемой частини гри, очевидна проблема призначеного для користувача інтерфейсу;
- тривіальна (Trivial) – Тривіальна помилка, яка не стосується логіки гри, погано відтворена проблема, малопомітна за допомогою призначено-

[14][14)] КАКИЕ СУЩЕСТВУЮТ САМЫЕ РАСПРОСТРАНЕННЫЕ КАТЕГОРИИ БАГОВ В ИГРАХ? URL: https://training.qatestlab.com/blog/technical-articles/category-bug-game/ (дата звернення: 27.11.2019)

го для користувача інтерфейсу, проблема сторонніх сервісів, проблема, не надає ніякого впливу на загальну якість продукту.

4.2 Категорія багів

Крім серйозності, баги класифікуються за категоріями[15][15)]:

- візуальні (visual) – розрив зображення на екрані, відсутність текстур, клиппинг (обрізання областей зображення) та ін.;
- аудіо (audio) – відсутність озвучення, спотворення звуку, занадто низька / висока гучність;
- баги дизайну рівнів (level design) – невидима стіна, відсутність геометрії (текстура присутній, але 3D моделі немає, що дозволяє пройти крізь стіну);
- штучний інтелект (artificial intelligence) – гравець не в змозі рухатися правильно по ходу гри, не рухається зовсім, занадто часто вмирає, не може відкрити двері;
- баги фізики (physics) – об'єкти ширяють у повітрі, коли не повинні, об'єкт не ламається, об'єкт не зупиняється після того, як його штовхнули, неможливість скласти об'єкти в купу;
- стабільність (stability) – фризи, креш (чорний екран), Crash to Desktop (ПК), неможливо завантажити рівень, гра не відповідає;
- дефекти продуктивності (performance) – найнижчий показник ФПС (проблеми з анімацією), занадто довго вантажаться рівні, мінімальна підтримувана конфігурація ПК не може відтворити гру, гра дуже довго встановлюється, дуже часто гра зупиняється, щоб завантажити дані;

[15][15)] Тестировщик о классификации игровых глюков URL: https://dtf.ru/gamedev/30926-ne-ficha-a-bag-testirovshchik-o-klassifikacii-igrovyh-glyukov (дата звернення: 29.11.2019)

− нетворкінг (networking) – проблеми із з'єднанням, неможливо приєднатися до запрошення, лаги (затримки у відповіді сервера на дії гравця), невидимі гравці, помилки з підрахунком очок.

4.3 Тестування програмних продуктів

Провівши повний цикл і спектр тестування нашого програмного продукту, я не знайшла більш-менш значущих програмних помилок, які могли б бути репрезентативними для типології програмних помилок з різних причин (терміни, досконаліть і простота коду, тому в даній частини ми розглянемо найпомітніші і самі показові програмні помилки на прикладі програмних помилок з інших програмних продуктів різних років, які я піддавала тестування в різний час і під різними умовами.

Тестований продукт: Fallout 4 (RPG, 2015, Bethesda Game Studios)

Тип програмної помилки: Критична

Вид програмної помилки: візуальна (visual) – розрив зображення на екрані, відсутність текстур, клиппинг (обрізання областей зображення) та ін.

Опис бага: Графічні артефакти при некоректній роботі відеокарти (рис.4.1)

Рисунок 4.1 – Приклад візуального багу

Передбачувані умови виникнення: Випадкові моменти часу при високому навантаженні

Передбачувана причина помилки: Це проблема відеокарти. Виникає зазвичай через те, що відеопам'ять розігнана по частоті.

Ймовірне рішення: Беремо Afterburner і починаємо по 5 МГц знижувати частоту відеопам'яті поки не пропадуть полігони. Якщо після зниження на 100+ вони не підуть – один з чіпів пам'яті підгуляв. Визначається "вологим пальцем" (Тестування нагріву чіпа)

Тестований продукт: World of Tanks (MMO, 2010, Wargaming)

Тип програмної помилки: Значна

Вид програмної помилки: баги дизайну рівнів (level design) – невидима стіна, відсутність геометрії (текстура присутній, але 3D моделі немає, що дозволяє пройти крізь стіну);

Рисунок 4.2 – Приклад багу дизайну рівнів

Опис бага: При спробі замаскуватися в кущі гравець натикається на невидиму стіну (рис.4.2)

Передбачувані умови виникнення: При наближенні до об'єкту і при виконанні деяких умов.

Передбачувана причина помилки: Помилка рендеру прозорої текстури (текстура обробляється як об'єкт) або недоліки налагодженості програмного коду.

Ймовірне рішення: Рефакторинг, поглиблене тестування і оптимізація коду, ревізія текстур.

Тестований продукт: S.T.A.L.K.E.R. Call of Pripyat (FPS, GSC Game World, 2009)

Тип програмної помилки: Незначна (Minor)

Вид програмної помилки: баги фізики (physics) – об'єкти ширяються у повітрі, коли не повинні, об'єкт не ламається, об'єкт не зупиняється після того, як його штовхнули, неможливість скласти об'єкти в купу;

Рисунок 4.3 – Приклад багу фізики

Опис бага: Після події (смерть від рук гравця) модель NPC неправильно оброблена за фізичним движком гри (модель застрягла в столі). (рис.4.3)

Передбачувані умови виникнення: У випадкові моменти часу в залежності від досконалості фізичного движка.

Передбачувана причина помилки: Недосконаліть фізичного движка виражене в неповному наборі умов для обробки об'єктів або їх свідоме виключення для збільшення продуктивності.

Ймовірне рішення: Заміна фізичного движка на більш досконалий або додавання додаткових умов разом з рефакторингом коду для збереження продуктивності.

Тестований продукт: Tom Clancy`s The Division (MMO Action/RPG, Ubisoft Massive, 2016)

Тип програмної помилки: Блокуюча (Blocker)

Вид програмної помилки: нетворкінг (networking) – проблеми із з'єднанням, неможливо приєднатися до запрошення, лаги (затримки у відповіді сервера на дії гравця), невидимі гравці, помилки з підрахунком очок.

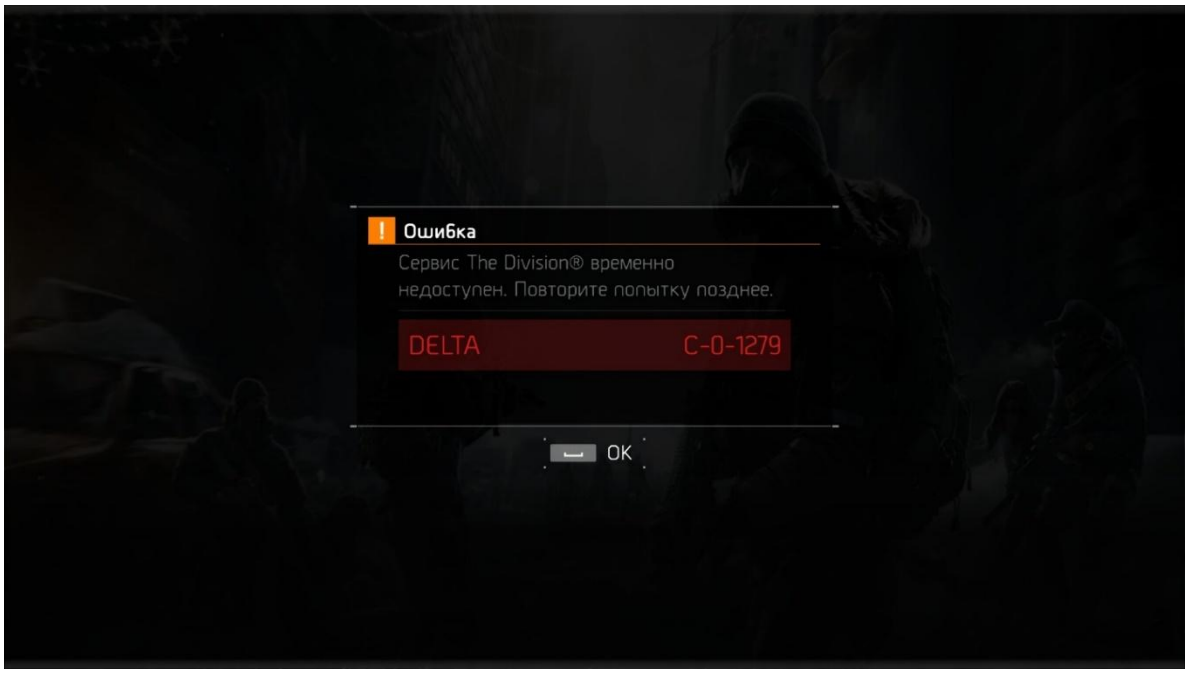

Рисунок 4.4 – Приклад багу нетворкінг

Опис бага: проблеми із з'єднанням невідомої природи. (рис.4.4)

Передбачувані умови виникнення: У випадкові моменти часу в незалежності стабільності з’єднання з інтернетом.

Передбачувана причина помилки: Велика розмаїтність мережевих помилок спільних та власних планів (навантаження на сервера, черга клієнтів, способи і швидкість передачі даних між серверами, якість і кількість мережевих каналів до серверів, вимоги до швидкості з'єднання кінцевого користувача).

Ймовірне рішення: Оптимізація мережевої інфраструктури (збільшення потужності і кількості серверів, поліпшення логістики підключення серверів (виключити бездротове з'єднання з іншими серверами на увазі її ненадійності), розширення інтернет-шлюзу як на вхід так і на вихід. Так як дуже часто багато власників серверів природно або штучно створюють дизбаланс який впливає на підсумкову продуктивність всього кластера в цілому).

Тестований продукт: World of Tanks (MMO, 2010, Wargaming)

Тип програмної помилки: Тривіальна (Trivial)

Вид програмної помилки: штучний інтелект (artificial intelligence) – гравець не в змозі рухатися правильно по ходу гри, не рухається зовсім, занадто часто вмирає, не може відкрити двері;

Опис бага: Дана ситуація пов'язана з тим, що навесні гравець отримав аналогічний танк як орендний. Після закінчення терміну оренди танк не був прибраний з ангара. (рис.4.5 – 4.6)

У підсумку, гра вирішила, що танк у гравця вже є і запропонувала автоматичну компенсацію. Після придбання танка, орендний був замінений на основний зі списанням повної вартості без компенсації.

Передбачувані умови виникнення: Баг внутріігрового магазину при несподіваних умовах.

Передбачувана причина помилки: Недостатня якість QA при релізі на основну версію.

Ймовірне рішення: На етапі розробки і тестування приділяти підвищену увагу на будь-яку додаткову функціональність внутріігрового магазину під час повного життєвого циклу всього продукту.

Рисунок 4.5 – Приклад багу штучного інтелекту до обробки операції з внутрішньою валютою

Рисунок 4.6 – Приклад багу штучного інтелекту після обробки операції з внутрішньою валютою

Тестований продукт: Tom Clancy`s The Division (MMO Action/RPG, Ubisoft Massive, 2016)

Тип програмної помилки: Тривіальна (Trivial)

Вид програмної помилки: візуальні (visual) – розрив зображення на екрані, відсутність текстур, клиппинг (обрізання областей зображення) та ін.;

Опис бага: Моделювання зброї окремо від персонажа не дивлячись на те, що по анімації моделі зброя повинна бути в руках. (рис.4.7)

Передбачувані умови виникнення: Важко відтворена, складно описати точні умови виникнення.

Передбачувана причина помилки: Недостатня якість QA при релізі на основну версію або внутрішня помилка логіки анімації.

Ймовірне рішення: Через складність відтворення даного бага неможливо створити единий концепт вирішення проблеми. Одне з можливих рішень – більшу кількість тестів на логіку анімацій.

Рисунок 4.7 – Візуальний вид програмної помилки

Тестований продукт: Tom Clancy`s The Division (MMO Action/RPG, Ubisoft Massive, 2016)

Тип програмної помилки: Незначна (Minor)

Вид програмної помилки: баги дизайну рівнів (level design) – невидима стіна, відсутність геометрії (текстура присутній, але 3D моделі немає, що дозволяє пройти крізь стіну);

Опис бага: Під час повернення на базу при кооперативної грі один гравець не зміг потрапити всередину, тому що в його клієнті вхід був закритий. (рис.4.8 – 4.9)

Передбачувані умови виникнення: Дуже рідкий випадок, складно описати точні умови виникнення.

Передбачувана причина помилки: Недостатня якість QA на етапі тестування мережевої взаємодії (синхронізація декількох клієнтів не пройшла успішно).

Ймовірне рішення: більша кількість учасників тестування з різними конфігураціями мережевого обладнання.

Рисунок 4.8 – Баг дизайну рівнів

Рисунок 4.9 – Баг дизайну рівнів The Division

Тестований продукт: S.T.A.L.K.E.R. Call of Pripyat (FPS, GSC Game World, 2009)

Тип програмної помилки: Значна (Major)

Вид програмної помилки: баги дизайну рівнів (level design) – невидима стіна, відсутність геометрії (текстура присутній, але 3D моделі немає, що дозволяє пройти крізь стіну);

Опис бага: При підйомі на поверх будівлі раптово пропала частина текстури будівлі, хоча стіна фізично існувала. (рис.4.10)

Передбачувані умови виникнення: Складно повторюваний баг неуточненої форми відтворення.

Передбачувана причина помилки: Недостатня якість QA на етапі тестування в сфері тестування на слабких конфігураціях комплектуючих.

Ймовірне рішення: більша кількість учасників тестування з різними конфігураціями компьютерів.

Рисунок 4.10 – Невидима стіна

Тестований продукт: The Elder Scrolls V: Skyrim (RPG, Bethesda Game Studios, 2011)

Тип програмної помилки: Незначна (Minor)

Вид програмної помилки: баги дизайну рівнів (level design) – невидима стіна, відсутність геометрії (текстура присутній, але 3D моделі немає, що дозволяє пройти крізь стіну);

Опис бага: Кущі висять у повітрі. (рис.4.11)

Передбачувані умови виникнення: Халатна робота команди по дизайну рівнів.

Передбачувана причина помилки: Імовірно, занадто маленький розмір об'єкта в порівнянні із загальним масштабам рендеринга.

Ймовірне рішення: Якщо об'єкт поставлений автоматично – удосконалити логіку процедурного створення об'єктів (більше умов), якщо об'єкт поставлений вручну – змінити параметри об'єкта (розділити об'єкт на частини) або поміняти розташування об'єкта.

Рисунок 4.11 – Незначна помилка

Тестований продукт: The Elder Scrolls V: Skyrim (RPG, Bethesda Game Studios, 2011)

Тип програмної помилки: Значна (Major)

Вид програмної помилки: баги фізики (physics) – об'єкти ширяють у повітрі, коли не повинні, об'єкт не ламається, об'єкт не зупиняється після того, як його штовхнули, неможливість скласти об'єкти в купу;

Опис бага: Після події (таємного вбивства гравцем) об'єкт, зробивши анімацію падіння некоректно провзаємодіяв зі стільцем (впав в нього). (рис.4.12)

Передбачувані умови виникнення: Ймовірно всього, до таких результатів призводить поєднання тригера події і умови відтворення анімації (до уваги не береться довкілля).

Передбачувана причина помилки: Недолік коду виражений в неправильному пріоритеті умов обробки анімацій і подій (анімація програється у відриві від фізичного движка).

Ймовірне рішення: Рефакторинг, оптимізація коду, більшу кількість різних тестів.

Рисунок 4.12 – Баги фізики

Тестований продукт: The Elder Scrolls V: Skyrim (RPG, Bethesda Game Studios, 2011)

Тип програмної помилки: Значна (Major)

Вид програмної помилки: баги фізики (physics) – об'єкти ширяють у повітрі, коли не повинні, об'єкт не ламається, об'єкт не зупиняється після того, як його штовхнули, неможливість скласти об'єкти в купу;

Опис бага: При активації механізму програвся звук падаючих каменів але камені залишилися висіти в повітрі, а герою не було завдано шкоди. (рис.4.13).

Передбачувані умови виникнення: Рідкісний але типовий випадок бага фізичного движка.

Передбачувана причина помилки: Скоріш за все, скрипт анімації блокують логіку роботи інших скриптів (пріоритет анімації найвищий).

Ймовірне рішення: переписати код та провести профільні тести.

Рисунок 4.13 – Камені у повітрі

Тестований продукт: The Elder Scrolls V: Skyrim (RPG, Bethesda Game Studios, 2011)

Тип програмної помилки: Незначна (Minor)

Вид програмної помилки: штучний інтелект (artificial intelligence) – гравець не в змозі рухатися правильно по ходу гри, не рухається зовсім, занадто часто вмирає, не може відкрити двері;

Опис бага: Краб вирішив, що корабель – це земля і сховався в ньому, за логікою написаною розробником. (рис.4.14)

Передбачувані умови виникнення: Плаваюча помилка. Важко відтворювана.

Передбачувана причина помилки: Швидше за все розробники логіки штучного інтелекту в повному обсязі врахували умови використання даного інтелекту (занадто спростили).

Ймовірне рішення: Рефакторинг, оптимізація коду, більшу кількість профільних тестів.

Рисунок 4.14 – Баг штучного інтелекту

Тестований продукт: Tom Clancy`s The Division (MMO Action/RPG, Ubisoft Massive, 2016)

Тип програмної помилки: Незначна (Minor)

Вид програмної помилки: баги фізики (physics) – об'єкти ширяють у повітрі, коли не повинні, об'єкт не ламається, об'єкт не зупиняється після того, як його штовхнули, неможливість скласти об'єкти в купу;

Опис бага: Під час зачіпки троса і на тросі моделі гравців не матеріальні. (рис.4.15)

Передбачувані умови виникнення: Баг просто відтворюється через дію з тросом і його анімації.

Передбачувана причина помилки: Відсутність обробки фізичним движком чого-небудь на тросі з різних причин (оптимізація).

Ймовірне рішення: Ввести обробку моделей на тросі фізичним движком.

Рисунок 4.15 – Фізична помилка

Тестований продукт: The Elder Scrolls V: Skyrim (RPG, Bethesda Game Studios, 2011)

Тип програмної помилки: Критична (Critical)

Вид програмної помилки: баги дизайну рівнів (level design) – невидима стіна, відсутність геометрії (текстура присутній, але 3D моделі немає, що дозволяє пройти крізь стіну);

Опис бага: Під час бою з противником він провалився під землю (під текстури). (рис.4.16)

Передбачувані умови виникнення: Залежить від локації і можливо від певного взаємного розташування героя і противників.

Передбачувана причина помилки: Погана розробка дизайну рівнів, погане тестування кінцевого продукту.

Ймовірне рішення: більш різноманітні профільні тести.

Рисунок 4.16 – Критична помилка

ВИСНОВКИ

Тестувальник – це більше, ніж професія. Це образ проактивного життя і прагнення це життя поліпшити для всіх посильними і ефективними засобами. Цілі тестувальника щодо продукту найбільш подібні до цілей бізнесу та стратегічної мети компанії щодо цього продукту, і в той же час глибокі всередині компанії в ролі дослідника. А раз так, то головні його якості - це енергія, знання і гнучкість. Але в той же час робота тестувальника – це не загальне знання і відповідальність за якість продукту і якість послуг.

У тестування є кордони: з одного боку обмежені проектом і вимогами в ньому (менеджмент проекту та встановлений життєвий цикл програми), і з іншого – процесами, за які відповідає QA. Але про відмінності QA від тестування зовсім інша розмова.

Продовжуючи тему сучасного стану тестування слід зазначити високу актуальність даного диплома, так як з ростом кількості розробників, програм, інтегрованих засобів розробки, мов програмування, різноманітності фізичних движків так само зростає роль і різноманітність засобів, методів і способів тестування, так як велика кількість програм виключає створення єдиного універсального підходу до тестування.

У цій роботі були розглянуті основні види тестування, класифікація видів тестування, історія тестування, деякі інструменти для тестування. Під час написання цього диплома я, нажаль, не виявила багів в нашій грі. Це обумовлено тим, що гра маленька, тому і досконаліше створювалась.
Результатами тестування я підтвердила актуальність теми тестування.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Історія виникнення комп’ютера. URL: https://sites.google.com/site/komputertvijpomicnik/komputer--tvij-pomicnik/istoria-viniknenna-komputera.
2. Як народжувався перший комп'ютер. URL:http://ua.uacomputing.com/stories/mesm/.
3. Роберт У. Себеста. Основные концепции языков программирования. 5-е изд.М.: Вильямс, 2001.672 с.ISBN 5-8459-0192-8.
4. Rouse, Richard. Game Design: Theory & Practice : [англ.].2.Los Rios Boulevard, Plano, Texas, USA : Wordware Publishing, 2004. 698 с.ISBN 1-55622-912-7.
5. Travis Fahs. IGN Presents the History of Survival Horror URL: https://www.ign.com/articles/2009/10/30/ign-presents-the-history-of-survival-horror. (дата звернення: 17.11.2019)
6. Джек Фолк, Сем Канер, Енг. Кек Нгуєн. Тестування програмного забезпечення. Видавництво ДиаСофт, 2001.

7. Види Тестування URL: / Про Тестінг - Тестування платформи Java. URL: http://www.protesting.ru/testing/testtypes.html.(дата звернення: 19.11.2019)
8. Что такое Конфигурационное тестирование. URL: http://software-testing.org/testing/chto-takoe-konfiguracionnoe-testirovanie-configuration-testing.html (дата звернення: 21.11.2019)
9. Основные положения тестирования. Интересные публикации. Хабрахабр. URL: https://habrahabr.ru/post/110307/(дата звернення: 22.11.2019)
10. Ron Patton. Software Testing. 2005.
11. Савин Роман. Тестирование DOT COM. Издательство Дело, 2007
12. Рекс Блек. Ключевые процессы тестирования - М.: Издательство Ло-ри, 2014.544 с.
13. Тестирование. Фундаментальная теория.URL: https://habr.com/ru/post/279535/ (дата звернення: 25.11.2019)
14. КАКИЕ СУЩЕСТВУЮТ САМЫЕ РАСПРОСТРАНЕННЫЕ КАТЕГОРИИ БАГОВ В ИГРАХ? URL:/https://training.qatestlab.com/blog/technical-articles/category-bug-game/ (дата звернення: 27.11.2019)
15. Тестировщик о классификации игровых глюков.URL: https://dtf.ru/gamedev/30926-ne-ficha-a-bag-testirovshchik-o-klassifikacii-igrovyh-glyukov (дата звернення: 29.11.2019)

Д О Д А Т К И

ДОДАТОК А

Баги у програмному продукті «NoSleep»

Рисунок А.1 – Баг з колізією між об’єктами

Рисунок А.2 – Баги текстур