**Технически университет**

**Катедра Софтуерни и интернет технологии**

**Д И П Л О М Н А Р А Б О Т А**

На Мариан Веселинов Карастоянов ф.№61362156

**Специалност:** Софтуерни и интернет технологии

**Образователно-квалификационна степен:** бакалавър

**Тема:** Разработка на помощен инструмент за настолната игра “Only War”

**Научен ръководител:** доц. Владимир Николов

Съдържание

[1. Въведени и цел на проекта 3](#_Toc491538843)

[1.1 Постановка на дипломното задание 3](#_Toc491538844)

[1.2 Цели и задачи на разработката 3](#_Toc491538845)

[1.3 Необходимост от решаването на дипломната задача 4](#_Toc491538846)

[2. Обзор на използваните програмни средства и технологии 6](#_Toc491538847)

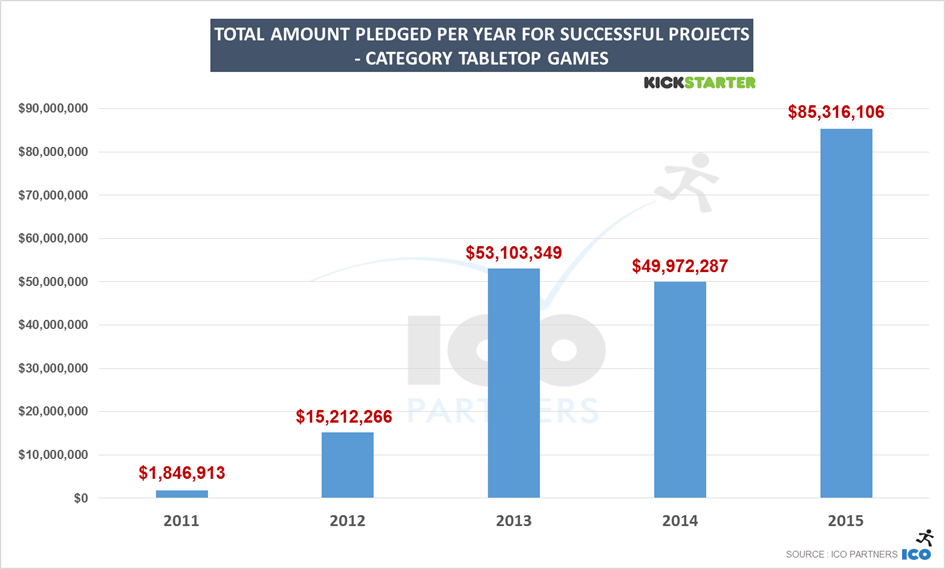
[2.1 Програмен език C sharp 6](#_Toc491538848)

[2.2 Language Integrated Query (LINQ) 7](#_Toc491538849)

[2.3 MongoDB 8](#_Toc491538850)

# Въведени и цел на проекта

* 1. Постановка на дипломното задание

През последните 5 до 10 години жанрът на настолните игри търпи драматично развитие със милиони долари(виж Фиг.1) инвестирани във игри и нужните за тях приложения и инструменти. Основна част от този растеж са така наречените „ролеви игри“ във който играчът поема ролята на персонаж от вселената на дадената игра. Някой от най известните примери за такива игри са “Dungeons and Dragons” (Подземия и Дракони или просто D&D), Зовът на Ктхулу (Call of Chthulu), Нуменера (Numenera) и обектът на тази работа серията Тъмна Ерес и Само Война (Dark Heresy и Only War). Тъй като някой от тези игри са относително стари и са били разработвани преди икономическият взрив на мобилните приложения много от тях изискват значително количество алгоритмизирано смятане и записване което да бъде извършвано от самите играчи.

Фиг.1

В последните години се вижда растеж на помощните приложения и инструменти за някои от тези игри най-известното от който е D&D Legendary което вкарва цялата система в електронният век със значителна софтуерна поддръжка.

За съжаления не съществуват никакви приложения за поддръжка и генериране на герой за системите на Only war и другите игри от тази серия. Тъй като това е фокусът на заданието и решението по-нататък всичко което не е експлицитно упоменато че не се отнася за серията Only War следва да бъде отнесено към нея.

* 1. Цели и задачи на разработката

Целите на разработеното приложение са да покрие нуждите на играчите да създават, запазват, разпространяват и при нужда модифицират създадените герой без да се нуждаят от други спомагателни средства.

По – долу са описани реалните процеси и функционалности който трябва да бъдат имплементирани от приложението.

Основни цели:

* Създаване на герой.
  + Изпиране на „специалност“
  + Избиране на „взвод“
  + Случайно генериране на характеристики
* Преглед във общ и детайлен план създаденият герой.
* Запазване и споделяне на героят.
* Добавяне на ресурсът „Опит на героя“
* Използване на ресурсът „Опит“
  + Закупуване на таланти
  + Закупуване на умения
  + Закупуване на „характеристики“ или „Умения“

1.3 Необходимост от решаването на дипломната задача

Към моментът на решаването на дипломното задание на пазарът няма спомагателни средства който да извършват динамична работа със частите от „Личностния Лист“ (Character Sheet) на героят. Има създадени технологии които записват и запазват подадените параметри, но те не прилагат никаква „бизнес логика“ над тях, а вместо това ги записват като прост текст. Този модел носи със себе си плюсове и минуси, някой много по големи от други в очите на потребителите.

Плюсове:

* Лесна модификация на съществуващите правила.
* По-голямо участие и замесеност във героят.

Минуси:

* Трудоемкост при създаване .
* Трудоемкост при използване и следене на ресурсите.
* Човешка грешка при следенето и калкулацията на дадените точки.
* Нужда от спомагателни средства.

Приложението следва да реши тези проблеми като използва лесен за използване и навигация интерфейс, извършване на всички калкулации без участието на човешки фактор.

# Обзор на използваните програмни средства и технологии

* 1. **Програмен език C sharp**

C# е обектно-ориентиран език за програмиране, разработен от Microsoft, като част от софтуерната платформа .NET. Стремежът още при създаването на C# езика е бил да се създаде един прост, модерен, обектно-ориентиран език с общо предназначение. Основа за C# са C++, Java и донякъде езици като Delphi, VB.NET и C. Той е проектиран да балансира мощност (C++) с възможност за бързо разработване (Visual Basic и Java). Те представляват съвкупност от дефиниции на класове, които съдържат в себе си методи, а в методите е разположена програмната логика – инструкциите, които компютърът изпълнява. Програмите на C# представляват един или няколко файла с разширение .cs., в които се съдържат дефиниции на класове и други типове. Тези файлове се компилират от компилатора на C# (csc) до изпълним код и в резултат се получават асемблита – файлове със същото име, но с различно разширение (.exe или .dll).

Едни от принципи за дизаин за C# са:

* C# е създаден като прост, модерен с общо предназначение и обектно-ориентиран език за програмиране.
* Езикът е предназначен за използване в развиващите се софтуерни компоненти, той е подходящ и за разполагане в разпределена среда.
* На езика C# и върху .NET платформата може да бъде разработван изключително разнообразен софтуер, като офис приложения, уеб приложения, уеб сайтове, настолни приложения, мултимедийни Интернет приложения, приложения за мобилни телефони, различни видове игри и много други.

C# е измислен да бъде лесен за работа, универсален, съвременен обектно-ориентиран език за програмиране от високо ниво.

Обектно-ориентираното програмиране е модел на програмиране, който използва обекти и техните характеристики, и взаимодействия за изграждането на компютърни програми. Софтуерните обекти моделират обекти от реалния свят или абстрактни концепции, които също се разглеждат като обекти.

Класът дефинира абстрактните характеристики на даден обект. Той е план или шаблон, чрез който се описва даден обект. Класовете са градивните елементи на ООП и са неразделно свързани с обектите. Всеки обект е представител на точно един клас.

* 1. **Language Integrated Query (LINQ)**

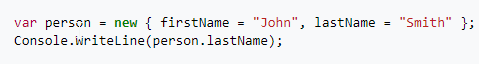
Вградени в езика заявки („Language Integrated Query“) е компонент на .NET Framework които добавя вградена поддръжка за заявки за езиците използващи .NET, въпреки че е екпортиан за PHP (PHPLinq), JavaScript (linq.js), TypeScript (linq.ts), and ActionScript (ActionLinq) - нито един от тези портове не успява да запази пълната функционалност на LINQ в езиците в който е бил интегриране от зачатие като например C#.

LINQ разширява еззикът като добавя като добавя изрази за заявки сходни с тези на SQL и може да бъде използван за лесно извеждане на работната информация от масиви, классове от изброяем тип(Еnumerable classes), XML документи, релационни бази данни и други източници.

LINQ също така дефинира набор от методи(нарачени стандартни оператори за заявки, или стандартни оператори за последователности), заедно със правила за парсиране използвани от компилатора за да парсира свободни изрази от заявки към ези методи, ламбда изрази(lambda expressions) както и анонимни типове.

Въпреки че LINQ е основно имплементиран като библеотека за .NET Framework 3.5, той също така дефинира незадължителни езикови разширения който позволяват заявките да се правят в първокласни езикови конструкций и добавя значитлна "синтактична захар". Тези езикови разширения пъровначално са имплементирани в C# 3.0, VB 9.0, F# и Oxygene.

Езикови разширения биват :

* Lambda expressions: lambda expressions позволяват предикати и други свободни модели да бъдат записани заедно с по конкрен синтаксис.
* Обектов инициализатор (Object Initializer): Object Initializer-ите помагат на обектите да бъдат създавани и инициализани в един и същи scope, както е изисквано от Select и Join операторите
* Анонимни типове (Anonymous types): Анонимните типове позволяват класовете който съдържат само декларации на член-данните да се подразбират от компилаторите(виж Фиг.2) .Това е полезно за Select и Join операторите, чиито резултатни нипове може да се различават от оригиналните типове на обектите. Комплиаторът използва типово подразбиране за да определи полето съдържано в класът и генерира аксесори и мутатори за избраните полета.
* Променливи с имплицитен тип (Implicitly typed variables): Това подобрение позволява променливите да бъдат декларирани без да се опоменава техният тип. Езиците C# 3.0 и Oxygene ги декларират във системната дума var(виж Фиг. 2). В VB9.0 това се извършва със системната дума Dim. Такива обекти все още се изписват често се типизират "силно", за такивиа обекти компилаторът подразбира типът на променливите чрез типово подразбиране, което позволява резултатите от заявките да бъдат специфицирани без декларация на типът на междинните променливи.

Фиг.2

* 1. **NoSQL**

Нерелационната база данни (на английски: Not only Structured Query Language, NoSQL) предоставя механизъм за съхранение и възстановяване на данни, който използва свободен съгласуван модел за разлика от по–често ползваната релационна база данни. Ползите на този подход включват изчистен дизайн, хоризонтално мащабиране и фин контрол върху наличнaта информация. Нерелационната база данни е най-често добре оптимизирано хранилище, съдържащо информация от тип ключ-стойност. Предназначението ѝ е да улесни процесите по възстановяване и добавяне на информация, с цел оптимизиране на производителността в условия на неумишлено забавяне на системата и въвеждане на прекалено големи количества данни. Нерелационната база данни намира значима и нарастваща роля в real-time web и big data приложенията. Нерелационните системи са наричани „Not only SQL“ в превод „Не само база данни“ за да се подчертае, че те в действителност позволяват употребата на езици за търсене различни, но сходни на тези в SQL.

Нерелационната база данни има разпределена и устойчива на повреди архитектура, като обектите с информация се съхраняват на няколко сървъра. По този начин системата може да се предпази като добавя повече сървъри и тогава при повреда на сървър продължава да работи. Този тип база данни се разширява хоризонтално и се използва за управлението на голямо количество от данни, когато производителността (в реално време) е по–важна от последователността (както в индексирането на голям брой документи, обслужващите страници на натоварените сайтове и предоставянето на стриймове).

ACID vs BASE. Нерелационната база данни не могат непременно да дадат пълна гаранция за ACID (Atomicity, Consistency, Isolation, Durability). Обикновено съгласуваността е гарантирана или транзакциите са ограничени до един обект с информация. Това означава, че ако се даде достатъчно дълъг период от време, по време на който не се пращат промени, може да се очаква всички обновления да се разпространят в системата.

Съществуват различни подходи за класифициране на нерелационните база данни с различни категории и подкатегории. Поради различните подходи за определяне на нефункционалните изисквания и съвкупността от характеристики е трудно да се направи конкретен преглед на нерелационните бази данни. Все пак, класификацията, която може би е най-общоприета се базира на модела на данните. Част от нерелационните база данни и техните прототипи, класифицирани на базата на модела на данните са представени в следния списък:

* Колона: Hbase, Cassandra, Accumulo
* Документ : MongoDB, Couch, Raven
* Ключ - стойност: Dynamo, Riak, Azure, Redis, Cache, GT.m
* Граф: Neo4J, Allegro, Virtuoso, Bigdata
  + 1. **MongoDB**

MongoDB е система за обработване на бази данни от документи, разработена от 10gen. Тя е от рода на нерелационните бази данни (NoSQL). Вместо да съхранява информация в таблици, както е при традиционните релационни бази данни, MongoDB съхранява структурираната информация в JSON формат с динамични схеми. Това прави интегрирането на информацията в определени приложения доста по-лесно и по-бързо.

* 1. **JavaScript Object Notation (JSON)**

JSON е текстово базиран отворен стандарт създаден за човешки четим обмен на данни. Произлиза от скриптовия език JavaScript, за да представя прости структури от данни и асоциативни масиви, наречени обекти. Въпреки своята връзка с JavaScript, това е езиково независима спецификация, с анализатори, които могат да преобразуват много други езици в JSON.

Форматът на JSON първоначално е бил създаден от Дъглас Крокфорд (Douglas Crockford) и е описан в RFC 4627. Официалният Интернет медия тип за JSON е application/json. Разширението на файловете написани на JSON е .json.

Форматът на JSON често е използван за сериализация и предаване на структурирани данни през Интернет връзка. Използва се главно, за да предаде данни между сървър и Интернет приложение, изпълнявайки функциите на алтернатива на XML.