МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ
Федеральное государственное автономное образовательное учреждение высшего образования
«КРЫМСКИЙ ФЕДЕРАЛЬНЫЙ УНИВЕРСИТЕТ им. В. И. ВЕРНАДСКОГО»
ФИЗИКО-ТЕХНИЧЕСКИЙ ИНСТИТУТ
Кафедра компьютерной инженерии и моделирования

РАЗРАБОТКА МНОГОПОЛЬЗОВАТЕЛЬСКОГО ПРИЛОЖЕНИЯ "КРЕСТИКИ НОЛИКИ ПО СЕТИ"

Курсовая работа
по дисциплине «Программирование»
студента 1 курса группы ПИ-б-о-201(2)
Никонова Федора Андреевича

направления подготовки 09.03.04 «Программная инженерия»
(код и наименование)

старший преподаватель кафедры компьютерной инженерии и моделирования _____ (оценка) Чабанов В. В.

_____ (подпись, дата)

Симферополь, 2021

Реферат

Никонов Ф. А. Разработка многопользовательского приложения «Крестики нолики по сети» // Курсовая работа (уровень бакалавриата) по специальности 09.03.04 Программная инженерия / Кафедра компьютерной инженерии и моделирования Физико-технического института Крымского федерального университета им. В. И. Вернадского. – Симферополь, 2021. – 31 с., 0 табл., 3 рис., 0 прил., 23 ист.

Игровая индустрия является одной из значимых индустрий, поставляющих прикладное программное обеспечение на компьютеры в настоящее время. Таким образом, умение разрабатывать игры является важным навыком, который может помочь найти вакансию на рынке труда. С другой, стороны современные тенденции рынка показывают, что особую ценность представляют приложения, обладающие клиент-серверной архитектурой.

Исходя из вышесказанного, темой работы была выбрана разработка многопользовательской игры «крестики нолики по сети».

В процессе разработки панируется получить навыки, позволяющие реализовать приложение в полной мере. Также, планируется использовать какие-либо языки разметки для создания интерфейса, поскольку именно они позволяют добиться максимальной гибкости стилевого оформления элементов программного продукта.

ПРОГРАММИРОВАНИЕ, ИГРА, КЛИЕНТ-СЕРВЕР, СИНХРОНИЗАЦИЯ, АВТОРИЗАЦИЯ, КАСТОМИЗАЦИЯ

Оглавление

Список сокращений и условных обозначений

HTML	HyperText Markdown Language
CSS	Cascading Style Sheets

Введение

Игровая индустрия является одной из значимых индустрий, поставляющих прикладное программное обеспечение на компьютеры в настоящее время. Таким образом, умение разрабатывать игры является важным навыком, который может помочь найти вакансию на рынке труда. С другой, стороны современные тенденции рынка показывают, что особую ценность представляют приложения, обладающие клиент-серверной архитектурой.

Исходя из вышесказанного, темой работы была выбрана разработка многопользовательской игры «крестики нолики по сети».

В процессе разработки панируется получить навыки, позволяющие реализовать приложение в полной мере. Также, планируется использовать какие-либо языки разметки для создания интерфейса, поскольку именно они позволяют добиться максимальной гибкости стилевого оформления элементов программного продукта. Это позволит сделать игру более красивой и не фокусировать внимание на реализации сложных элементов интерфейса с помощью базовых алгоритмов «вручную».

Целью курсовой работы является реализация клиент-серверного многопользовательского приложения (игры), которая будет обладать функционалом, свойственным онлайн играм. В этот функционал входят регистрация и авторизация пользователей, и синхронизация игровых действий между клиентскими приложениями.

Стоит отметить ограничения определенные в начале работы – проект должен использовать логику на языках программирования C++ и Python. Допускается использование других языков программирования и разметки, однако суммарная доля C++ и Python должна составлять не менее 60% от всего написанного кода.

Согласно цели работы и заданных ограничений были выдвинуты следующие задачи:

1. определить средства и инструменты разработки, посредством анализа современных методов создания интерфейсов и клиент-серверных приложений;

2. определиться с форматом хранения данных и их архитектурой;
3. реализовать логику регистрации и авторизации;
4. разработать магазин скинов и логику, отвечающую за сохранения данных о скинах в базе данных;
5. написать код, отвечающий за подключение клиентов друг к другу;
6. завершить реализацию приложения, разработкой логики игрового процесса.

Объектом работы в данной курсовой работе является изучение методик разработки многопользовательских клиент-серверных приложений и дальнейшее их применение на практике.

В свою очередь, предметом работы является разработка клиент-серверной игры «крестики нолики по сети», обладающего логикой авторизации и магазином скинов.

В работе используются следующие методы исследования: анализ, синтез, программирование и тестирование.

Анализ выражается в изучении современных методов разработки клиент-серверных приложений и выбор наиболее подходящих под требования проекта. Благодаря синтезу формируется конечная цель и методы её достижения посредством изучения прикладной литературы. Основными методами выступают программирование и тестирование, которые позволяют реализовать игру с выбранным функционалом.

Глава 1

Теоретические основы разработки клиент-серверных приложений

1.1 Базовые понятия клиент-серверной архитектуры

«Клиент — сервер» — вычислительная или сетевая архитектура, в которой задания или сетевая нагрузка распределены между поставщиками услуг, называемыми серверами, и заказчиками услуг, называемыми клиентами [1]. Фактически клиент и сервер — это программное обеспечение.

Обычно эти программы расположены на разных вычислительных машинах и взаимодействуют между собой через вычислительную сеть посредством сетевых протоколов, но они могут быть расположены также и на одной машине. Программы-серверы ожидают от клиентских программ запросы и предоставляют им свои ресурсы в виде данных (например, загрузка файлов посредством HTTP, FTP, BitTorrent, потоковое мультимедиа или работа с базами данных) или в виде сервисных функций (например, работа с электронной почтой, общение посредством систем мгновенного обмена сообщениями или просмотр web-страниц во всемирной паутине).

Поскольку одна программа-сервер может выполнять запросы от множества программ-клиентов, её размещают на специально выделенной вычислительной машине, настроенной особым образом, как правило, совместно с другими программами-серверами, поэтому производительность этой машины должна быть высокой. Из-за особой роли такой машины в сети, специфики её оборудования и программного обеспечения, её также называют сервером, а машины, выполняющие клиентские программы, соответственно, клиентами.

Характеристика клиент-сервер описывает отношения взаимодействующих программ в приложении. Серверный компонент предоставляет функцию или услугу одному или нескольким клиентам, которые инициируют запросы на такие услуги [1]. Серверы классифицируются по предоставляемым ими услугам. Например, веб-сервер обслуживает веб-страницы, а файловый сервер обслуживает компьютерные файлы. Общий ресурс может быть любой из программного

обеспечения и электронных компонентов компьютера - сервера, от программ и данных в процессорах и запоминающих устройств. Совместное использование ресурсов сервера представляет собой услугу.

Является ли компьютер клиентом, сервером или и тем, и другим, определяется характером приложения, которому требуются сервисные функции. Например, на одном компьютере могут одновременно работать веб-серверы и программное обеспечение файлового сервера, чтобы обслуживать разные данные для клиентов, отправляющих различные типы запросов. Клиентское программное обеспечение также может взаимодействовать с серверным программным обеспечением на том же компьютере. Связь между серверами, например, для синхронизации данных, иногда называется межсерверной.

Вообще говоря, служба - это абстракция компьютерных ресурсов, и клиенту не нужно беспокоиться о том, как сервер работает при выполнении запроса и доставке ответа. Клиенту нужно только понять ответ, основанный на известном протоколе приложения, то есть содержание и форматирование данных для запрашиваемой услуги.

Клиенты и серверы обмениваются сообщениями в шаблоне запрос-ответ. Клиент отправляет запрос, а сервер возвращает ответ. Этот обмен сообщениями является примером межпроцессного взаимодействия [1]. Для взаимодействия компьютеры должны иметь общий язык, и они должны следовать правилам, чтобы и клиент, и сервер знали, чего ожидать. Язык и правила общения определены в протоколе связи. Все протоколы клиент-серверной модели работают на уровне приложений. Протокол прикладного уровня определяет основные шаблоны диалога. Чтобы еще больше формализовать обмен данными, сервер может реализовать интерфейс прикладного программирования (API). API — это уровень абстракции для доступа к сервису. Ограничивая связь определенным форматом контента, он облегчает синтаксический анализ. Абстрагируя доступ, он облегчает межплатформенный обмен данными.

Сервер может получать запросы от множества различных клиентов за короткий период времени. Компьютер может выполнять только ограниченное

количество задач в любой момент и полагается на систему планирования для определения приоритетов входящих запросов от клиентов для их удовлетворения. Чтобы предотвратить злоупотребления и максимизировать доступность серверное программное обеспечение может ограничивать доступность для клиентов. Атаки типа «отказ в обслуживании» используют обязанности сервера обрабатывать запросы, такие атаки действуют путем перегрузки сервера чрезмерной частотой запросов. Шифрование следует применять, если между клиентом и сервером должна передаваться конфиденциальная информация.

В дополнение к клиент-серверной модели распределенные вычислительные приложения часто используют peer-to-peer архитектуру.

Клиент-сервер часто проектируется как централизованная система, которая обслуживает множество клиентов. Таким образом, требования к мощности, памяти и объёму хранилища сервера должны масштабироваться с ожидаемой нагрузкой. Системы балансировки нагрузки и отказоустойчивости часто используются для масштабирования сервера за пределами одной физической машины. В одноранговой сети два или более компьютера объединяют свои ресурсы и взаимодействуют в децентрализованной системе.

Одноранговые узлы — это равноправные или эквипотентные узлы в неиерархической сети. В отличие от клиентов в архитектуре клиент-сервер или клиент-очередь-клиент, одноранговые узлы взаимодействуют друг с другом напрямую. В одноранговой сети алгоритм в одноранговом коммуникационном протоколе балансирует нагрузку, и даже одноранговые узлы с небольшим количеством ресурсов могут помочь разделить нагрузку. Если узел становится недоступным, его общие ресурсы остаются доступными до тех пор, пока другие одноранговые узлы предлагают их.

В идеале узлу не нужно достигать высокой доступности, поскольку другие узлы компенсируют любое время простоя ресурсов. По мере изменения доступности и пропускной способности одноранговых узлов протокол перенаправляет запросы. Как клиент-сервер, так и ведущий-ведомый рассматриваются как подкатегории распределенных одноранговых систем [1].

1.2 Задачи и технологий реализации серверной части

Исходя из сказанного в пункте 1.1, серверная часть приложения выполняет ряд ключевых функций, в числе которых:

1) хранения данных пользователей;
2) обработка приходящих запросов;
3) рассылка информации по подключенным клиентам;
4) хранения состояния игры.

Хранения данных является важным аспектом работы программы, таким образом следует определить технологию, которую стоит использовать для передачи и хранения данных.

Существует два кардинально разных подхода: хранения данных с помощью какого-либо формата или же с использованием готовой базы данных.

К наиболее распространенным форматам данных относят JSON и XML. Наиболее популярными базами данных на сегодня являются MySQL и MongoDB.

Рассмотрим эти варианты подробнее.

1.2.1 Форматы данных JSON и XML

JSON — текстовый формат обмена данными, основанный на JavaScript. Как и многие другие текстовые форматы, JSON легко читается людьми. Формат JSON был разработан Дугласом Крокфордом [2].

Несмотря на происхождение от JavaScript (точнее, от подмножества языка стандарта ECMA-262 1999 года), формат считается независимым от языка и может использоваться практически с любым языком программирования. Для многих языков существует готовый код для создания и обработки данных в формате JSON.

За счёт своей лаконичности по сравнению с XML формат JSON может быть более подходящим для сериализации сложных структур. Применяется в веб-приложениях как для обмена данными между браузером и сервером (AJAX), так и между серверами (программные HTTP-сопряжения).

Поскольку формат JSON является подмножеством синтаксиса языка JavaScript, то он может быть быстро десериализован встроенной функцией eval() [3].

JSON-текст представляет собой (в закодированном виде) одну из двух структур:

Набор пар ключ: значение. В различных языках это реализовано как запись, структура, словарь, хеш-таблица, список с ключом или ассоциативный массив. Ключом может быть только строка (регистрозависимость не регулируется стандартом, это остаётся на усмотрение программного обеспечения. Как правило, регистр учитывается программами — имена с буквами в разных регистрах считаются разными, например), значением — любая форма [3]. Повторяющиеся имена ключей допустимы, но не рекомендуются стандартом; обработка таких ситуаций происходит на усмотрение программного обеспечения, возможные варианты — учитывать только первый такой ключ, учитывать только последний такой ключ, генерировать ошибку.

Упорядоченный набор значений. Во многих языках это реализовано как массив, вектор, список или последовательность.

Структуры данных, используемые JSON, поддерживаются любым современным языком программирования, что и позволяет применять JSON для обмена данными между различными языками программирования и программными системами [4].

В качестве значений в JSON могут быть использованы:

- запись — это неупорядоченное множество пар ключ:значение, заключённое в фигурные скобки «{ }». Ключ описывается строкой, между ним и значением стоит символ «:». Пары ключ-значение отделяются друг от друга запятыми.
- массив (одномерный) — это упорядоченное множество значений. Массив заключается в квадратные скобки «[ ]». Значения разделяются запятыми. Массив может быть пустым, то есть не содержать ни одного значения. Значения в пределах одного массива могут иметь разный тип.

- число (целое или вещественное).
- литералы true (логическое значение «истина»), false (логическое значение «ложь») и null.
- строка — это упорядоченное множество из нуля или более символов юникода, заключённое в двойные кавычки. Символы могут быть указаны с использованием escape-последовательностей, начинающихся с обратной косой черты «\» (поддерживаются варианты \", \\, \/, \t, \n, \r, \f и \b), или записаны шестнадцатеричным кодом в кодировке Unicode в виде \uFFFF [2].

Строка очень похожа на литерал одноимённого типа данных в языке Javascript. Число тоже очень похоже на Javascript-число, за исключением того, что используется только десятичный формат (с точкой в качестве разделителя). Пробелы могут быть вставлены между любыми двумя синтаксическими элементами [3].

Следующий пример показывает JSON-представление данных об объекте, описывающем человека. В данных присутствуют строковые поля имени и фамилии, информация об адресе и массив, содержащий список телефонов. Как видно из примера, значение может представлять собой вложенную структуру.

```
{
  "firstName": "Иван",
  "lastName": "Иванов",
  "address": {
    "streetAddress": "Московское ш., 101, кв.101",
    "city": "Ленинград",
    "postalCode": 101101
  },
  "phoneNumbers": [
    "812 123-1234",
    "916 123-4567"
```

```
  ]
}
```

Обратите внимание на пару "postalCode": 101101. В качестве значений в JSON могут быть использованы как числа, так и строки. Поэтому запись "postalCode": "101101" содержит строку, а "postalCode": 101101 — уже числовое значение. Из-за слабой типизации в Javascript и PHP строка может быть приведена к числу и не влиять на логику программы. Тем не менее, рекомендуется аккуратно обращаться с типом значения, так как JSON служит для межсистемного обмена [4].

На языке XML подобная структура выглядела бы примерно так:

```
<person>
  <firstName>Иван</firstName>
  <lastName>Иванов</lastName>
  <address>
    <streetAddress>Московское ш., 101, кв.101</streetAddress>
    <city>Ленинград</city>
    <postalCode>101101</postalCode>
  </address>
  <phoneNumbers>
    <phoneNumber>812 123-1234</phoneNumber>
    <phoneNumber>916 123-4567</phoneNumber>
  </phoneNumbers>
</person>
```

или так:

```
<person firstName="Иван" lastName="Иванов">
  <address streetAddress="Московское ш., 101, кв.101" city="Ленинград" postalCode="101101" />
```

```
  <phoneNumbers>
    <phoneNumber>812 123-1234</phoneNumber>
    <phoneNumber>916 123-4567</phoneNumber>
  </phoneNumbers>
</person>
```

Таким образом, среди текстовых форматов наиболее приемлемым под требования проекта является JSON.

1.2.2 Базы данных MySQL и MongoDB

MySQL — свободная реляционная система управления базами данных. Разработку и поддержку MySQL осуществляет корпорация Oracle, получившая права на торговую марку вместе с поглощённой Sun Microsystems, которая ранее приобрела шведскую компанию MySQL AB [5]. Продукт распространяется как под GNU General Public License, так и под собственной коммерческой лицензией. Помимо этого, разработчики создают функциональность по заказу лицензионных пользователей. Именно благодаря такому заказу почти в самых ранних версиях появился механизм репликации.

MySQL является решением для малых и средних приложений. Входит в состав серверов WAMP, AppServ [6], LAMP и в портативные сборки серверов Денвер, XAMPP, VertrigoServ. Обычно MySQL используется в качестве сервера, к которому обращаются локальные или удалённые клиенты, однако в дистрибутив входит библиотека внутреннего сервера, позволяющая включать MySQL в автономные программы.

Гибкость СУБД MySQL обеспечивается поддержкой большого количества типов таблиц: пользователи могут выбрать как таблицы типа MyISAM, поддерживающие полнотекстовый поиск, так и таблицы InnoDB [7], поддерживающие транзакции на уровне отдельных записей. Более того, СУБД MySQL поставляется со специальным типом таблиц EXAMPLE, демонстрирующим принципы создания новых типов таблиц. Благодаря открытой

архитектуре и GPL-лицензированию, в СУБД MySQL постоянно появляются новые типы таблиц.

26 февраля 2008 года Sun Microsystems приобрела MySQL AB за 1 млрд долларов, 27 января 2010 года Oracle приобрела Sun Microsystems за 7,4 млрд долларов и включила MySQL в свою линейку СУБД [8].

Сообществом разработчиков MySQL созданы различные ответвления кода, такие как Drizzle (англ.), OurDelta, Percona Server и MariaDB [9]. Все эти ответвления уже существовали на момент поглощения компании Sun корпорацией Oracle.

MySQL имеет двойное лицензирование. MySQL может распространяться в соответствии с условиями лицензии GPL. Однако по условиям GPL, если какая-либо программа использует библиотеки (или включает в себя другой GPL-код) MySQL, то она тоже должна распространяться по лицензии GPL. Это может расходиться с планами разработчиков, не желающих открывать исходные тексты своих программ. Для таких случаев предусмотрена коммерческая лицензия, которая также обеспечивает качественную сервисную поддержку. Для свободного программного обеспечения Oracle предоставляет отдельное исключение из правил, явным образом разрешающее использование и распространение MySQL вместе с ПО, распространяемым под лицензией из определённого Oracle списка.

MongoDB — документоориентированная система управления базами данных, не требующая описания схемы таблиц. Считается одним из классических примеров NoSQL-систем, использует JSON-подобные документы и схему базы данных. Написана на языке C++. Применяется в веб-разработке, в частности, в рамках JavaScript-ориентированного стека MEAN [10].

Система поддерживает ad-hoc-запросы: они могут возвращать конкретные поля документов и пользовательские JavaScript-функции. Поддерживается поиск по регулярным выражениям. Также можно настроить запрос на возвращение случайного набора результатов. Имеется поддержка индексов.

Система может работать с набором реплик [11], то есть содержать две или более копии данных на различных узлах. Каждый экземпляр набора реплик может

в любой момент выступать в роли основной или вспомогательной реплики. Все операции записи и чтения по умолчанию осуществляются с основной репликой.

Вспомогательные реплики поддерживают в актуальном состоянии копии данных. В случае, когда основная реплика дает сбой, набор реплик проводит выбор, которая из реплик должна стать основной. Второстепенные реплики могут дополнительно являться источником для операций чтения [12].

Система масштабируется горизонтально, используя технику сегментирования объектов баз данных — распределение их частей по различным узлам кластера [13]. Администратор выбирает ключ сегментирования, который определяет, по какому критерию данные будут разнесены по узлам (в зависимости от значений хэша ключа сегментирования). Благодаря тому, что каждый узел кластера может принимать запросы, обеспечивается балансировка нагрузки.

Система может быть использована в качестве файлового хранилища с балансировкой нагрузки и репликацией данных. Предоставляются программные средства для работы с файлами и их содержимым. GridFS используется в плагинах для Nginx и lighttpd. GridFS разделяет файл на части и хранит каждую часть как отдельный документ [14].

Может работать в соответствии с парадигмой MapReduce. Во фреймворке для агрегации есть аналог SQL-выражения GROUP BY. Операторы агрегации могут быть связаны в конвейер подобно UNIX-конвейрам. Фреймворк так же имеет оператор $lookup [15] для связки документов при выгрузке и статистические операции такие как среднеквадратическое отклонение.

Поддерживается JavaScript в запросах, функциях агрегации (например, в MapReduce). Поддерживает коллекции с фиксированным размером. Такие коллекции сохраняют порядок вставки и по достижении заданного размера ведут себя как кольцевой буфер. В июне 2018 года (в версии 4.0) добавлена поддержка транзакций, удовлетворяющих требованиям ACID. Есть официальные драйверы для основных языков программирования (Си, C++, C#, Go, Java, Node.js, Perl, PHP, Python, Ruby, Rust, Scala, Swift) [15]. Существует также большое количество

неофициальных или поддерживаемых сообществом драйверов для других языков программирования и фреймворков.

Основным интерфейсом к базе данных была командная оболочка «mongo». С версии MongoDB 3.2 в качестве графической оболочки поставляется «MongoDB Compass». Существуют продукты и сторонние проекты, которые предлагают инструменты с GUI для администрирования и просмотра данных.

Среди баз данных наиболее подходит под рамки проекта MongoDB. Однако между JSON и MongoDB [14] преимущество находится на стороне JSON. В первую очередь, это простота работы, поскольку игра относительно не большая и не требует большого количества вычислений и работы с данными. Во-вторых, данные в запросе можно передавать прямо в формате JSON, что позволит декодировать данные прямо из запроса. Таким образом, технологией для хранения данных выбран формат данных JSON.

1.3 Задачи и технологии реализации клиентской части

Для начала выделим главные задачи клиентской части приложения:

1. предоставление интерфейса для запуска игры и или подключения к ней;
2. обработка действий пользователя и отправка данных об этом на север;
3. отрисовка изменений пришедших с сервера.

Таким образом, клиентская сторона должна обладать высокой гибкостью, возможностями и инструментами для создания интерфейсов. На сегодняшний день наиболее гибким инструментом для создания интерфейсов являются языки разметки. Рассмотрим наиболее значимые из них.

1.3.1 Связка HTML+CSS

HTML (от англ. HyperText Markup Language — «язык гипертекстовой разметки») — стандартизированный язык разметки документов во Всемирной паутине. Большинство веб-страниц содержат описание разметки на языке HTML (или XHTML). Язык HTML интерпретируется браузерами; полученный в результате интерпретации форматированный текст отображается на экране монитора компьютера или мобильного устройства [17].

Язык HTML до 5-й версии определялся как приложение SGML (стандартного обобщённого языка разметки по стандарту ISO 8879 [18]). Спецификации HTML5 формулируются в терминах DOM (объектной модели документа).

Язык XHTML является более строгим вариантом HTML, он следует синтаксису XML и является приложением языка XML в области разметки гипертекста [19].

Во всемирной паутине HTML-страницы, как правило, передаются браузерам от сервера по протоколам HTTP или HTTPS, в виде простого текста или с использованием шифрования.

В HTML можно встроить программный код на языке программирования JavaScript, для управления поведением и содержанием веб-страниц. Также включение CSS в HTML описывает внешний вид и макет страницы [20, 21].

1.3.2 XAML

XAML (англ. eXtensible Application Markup Language) — расширяемый язык разметки для приложений (произносится [замл] или [зэмл]) — основанный на XML язык разметки для декларативного программирования приложений, разработанный Microsoft [22].

Модель приложений Vista включает объект Application. Его набор свойств, методов и событий позволяет объединить веб-документы в связанное приложение. Объект Application управляет выполнением программы и генерирует события для пользовательского кода. Документы приложения пишутся на XAML. Впрочем, с помощью XAML описывается, прежде всего, пользовательский интерфейс. Логика приложения по-прежнему управляется процедурным кодом (C#, VB, JavaScript и т. д.). XAML может использоваться как для браузер-базированных приложений, так и для настольных приложений.

XAML включает в себя основные четыре категории элементов: панели, элементы управления, элементы, связанные с документом, и графические фигуры. Заявлено 7 классов панелей, которые задают принципы отображения вложенных в них элементов [23]. Для задания положения элементов относительно границ

родительской панели используются атрибуты на манер свойств в объектно-ориентированных языках. Подобный синтаксис не совсем соответствует рекомендациям CSS, но является привычным для программистов настольных приложений.

Приложения, объявленные в XAML, могут включать множество страниц. Элемент управления PageViewer позволяет разбивать содержание на страницы и обеспечивает навигацию по ним [22]. Элемент ContextMenu помогает в создании навигационных меню приложения. Код процедурного языка может быть размещён непосредственно в файле XAML или же назначен при сборке проекта.

Так как в проекте заданы рамки в виде обязательного использования языка Python, лучше использовать связку HTML+CSS, поскольку на питоне имеются библиотеки для создания интерфейсов с помощью этих языков.

Одной из таких технологий является CEFPython3 – библиотека, имеющая в своем составе ядро Google Chrome и позволяющая манипулировать страницей как с помощью Python, так и с помощью JavaScript. Встроенные средства разработчика позволят значительно ускорить процесс разработки и получить в точности требуемый результат и ровный интерфейс.

Другой технологией, которая потребуется для создания клиента является сервер, который должен быть развернут на стороне клиента и получать данные от удаленного сервера. Такой технологией является Flask. Это библиотека, создающая новый сервер и позволяющая обрабатывать приходящие запросы. Данная библиотека основа на встроенном модуле языка Python – socket.

Глава 2

Разработка игрового приложения

2.1 Дизайн и логика работы приложения

Следующим этапом разработки приложения является определение основных функций, которые должны присутствовать для корректной работы. Такими функциями являются:

1. регистрация/авторизация;
2. магазин скинов;
3. окно ожидания подключения других клиентов;
4. игровая сцена.

Исходя из перечисленного функционала и требований проекта был разработан дизайн приложения, который представлен на рисунках 2.1 и 2.2.

Рисунок 2.1. Экран регистрации и главное меню.

Рисунок 2.2. Магазин скинов и игровое поле.

Требуется реализовать интерфейс приложения максимально приближенно к этим макетам.

2.2 Хранение и формат данных

Все данные в игре хранятся в формате JSON в файлах. Исходя из требуемого функционала, в программе должны быть файлы, которые хранят:

1. логины и пароли пользователей;
2. состояние товаров магазина у каждого игрока;
3. статусы подключений к играм;
4. текущее состояние игры.

Для удобства сгруппируем пункты 1-2 и 3-4. Следовательно будет два основных файла, выполняющие роль базы данных:

1. users.json;
2. games.json.

Отдельный файл, который также не является шаблонным – это цены на товары. Цены на товары имеет смысл выносить из кода в отдельный файл, поскольку в дальнейшем может потребоваться корректировка цен для улучшения игрового процесса, а изменения работающего может приводить к появлению ошибок и повторному тестированию продукта. Таким образом, будет ещё один файл – price.json.

Для упрощения генерации данных пользователя и игр имеет смысл создать вспомогательные данные, которые будут хранить шаблонные данные для новой сущности. Такими файлами станут:

1. default-user.json – шаблонная информация о пользователе;
2. default-game.json – шаблонная информация об игре и подключениях;
3. default-game-user.json – шаблонная информация о клиенте пользователя в рамках одной игры.

Итоговая структура файлов представлена на рисунке 2.3.

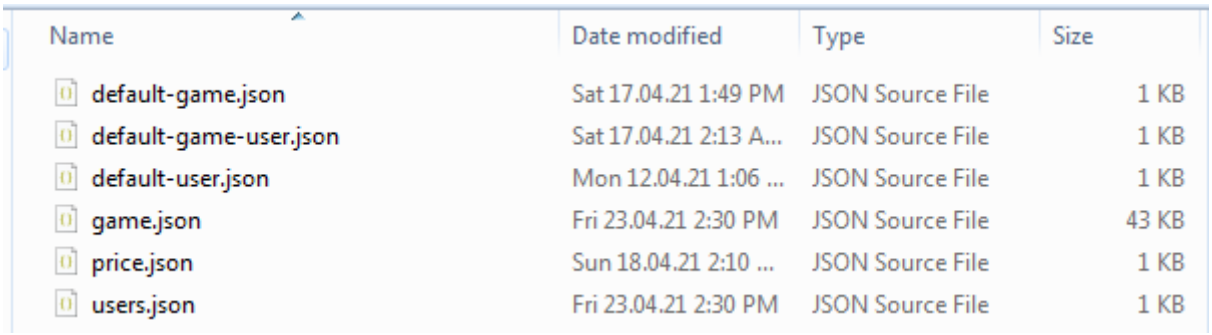

Рисунок 2.3. Структура файлов с данными на сервере.

2.3 Клиент-сервер и логика работы программы

Ранее были рассмотрены дизайн, формат данных, технологии и требования к программе. Теперь приступим к программной реализации приложения. Далее представлен обобщенный алгоритм работы приложения.

После запуска игры, пользователь попадает на главный экран. Это экран входа и регистрации. Пользователь вводил логин и пароль. Может нажать на одну из двух кнопок: «вход» или «регистрация». Библиотека CEFPython3 позволяет передавать данные между JavaScript и Python, таким образом запросы на сервер отправляются питоном. Если клиент нажал на «вход», сервер получит логин и пароль, затем попытается найти такой аккаунт в файле users.json. Если поиск не дал результатов, то будет возвращена информация об ошибке. В противном случае, сервер вернет ID найденного аккаунта. Эта информация потребуется клиенту для дальнейших обращений к серверу. В случае, если пользователь нажал на кнопку «регистрация», происходит проверка наличия такого логина в файле. Если такой

логин не был найдет – регистрация проходит успешно, в противном случае клиент получает сообщение об ошибке от сервера.

После входа в игру, пользователь попадает на главный экран. Этот экран содержит 4 кнопки:

1. создать игру;
2. подключится к игре;
3. магазин;
4. выйти.

Рассмотрим функционал этих кнопок по-отдельности.

При нажатии на кнопку «Создать игру», клиент оправляет запрос на создание игры. Сервер открывает файл games.json и определяет наименьший SID игры на текущий момент. Эта процедура выполняется для получения нового уникального SID игры. Затем, игра с новым SID создаётся с шаблонными данными. В неё сразу добавляется пользователь, создавший игру. Информация об игре содержит текущее положение фишек на поле и информацию о подключенных клиентах, в том числе принадлежность выбранных фигур игрокам, состояние готовности к игре и т.д. Фактически, весь игровой процесс синхронизируется с файловой системой и может быть продолжен в случае перезапуска сервера. После завершения операция на сервере, клиент получает ответ в виде SID созданной игры. SID игры выводится на экран выбора фигуры, для того чтобы игрок мог поделиться им со своим напарником. После этого клиентское приложение переходит на экран выбора фигур.

Если же клиент нажимает на кнопку «Подключится к игре», то игра запрашивает у него SID игры, к которой он хочет подключится. Чтобы узнать SID игры, пользователь должен связаться со своим напарником. После ввода SID игры, сервер выполняет поиск игры в файле games.json. Если игра была найдена, то данные о подключении дописываются в выбранный объект игры. В ответ клиенту приходит количество подключенных игроков, для вывода на экран количества подключений. После этого клиентское приложение переходит на экран выбора фигур.

При подключении или создании игры, клиентское приложение запускает локальный сервер для принятия запросов от сервера. Это требуется для получения обратной связи от удаленного сервера. Рассмотрим на примере создания и подключения игры. Клиент создающий игру запускает локальный сервер. Он отправляет хост и порт на котором этот он открыл сервер на удалённый сервер. Таким образом, сервер записывая информацию о клиенте, записывает и информацию о его локальном сервере, что даёт в дальнейшем ему возможность рассылки информации. Экран выбора фигур имеет информацию о количестве подключений к игре. Таким образом, при подключении нового игрока, это число должно изменится. Теперь, к игре подключается новый игрок. Он также разворачивает локальный сервер и отправляет на удаленный сервер информацию о том на каком хосте и порту он это сделал. Однако, сервер фиксирует подключение к игре и выполняет рассылку информации об этом, содержащую количество подключений на текущий момент. Так как клиентское приложение, создавшее игру, уже имеет локальный сервер, то на него приходит запрос от удаленного сервера с новым числом подключений, который в дальнейшем обрабатывается и число обновляется на экране у первого клиента. По такому принципу работает вся игровая логика приложения, разница лишь в том, что сервер и клиент обрабатывают разные запросы с разной информацией внутри них. Назовем этот процесс «синхронизация».

Таким образом, происходит синхронизация при выборе фигур в игре. Фигур должно быть выбрано от двух до пяти. Когда все участники игры нажимают кнопку «готов», т.е. приобретают состояние готовности – игра начинается.

Во время начала игры сервер определяет очередность ходов случайным образом и рассылает информацию об этом на все клиенты. Клиент разработан таким образом, что он не может ходить не в свой ход.

Игровой процесс подразумевает поочередные ходы между несколькими клиентами. После каждого хода сервер оценивает ситуацию на игровом поле с целью опередить наличие победной комбинации у одного из игроков. Если такая комбинация найдена – игра завершается. Победивший игрок получает выигрыш –

внутри игровую валюту, которую он может потратить в магазине для покупки скинов, а остальные игроки – информацию о поражении. Затем игроки отключаются от игры. Когда в игре не остаётся подключений она автоматически удаляется из файла games.json.

Рассмотрим логику работы магазина. При клике на кнопку «магазин» в главном меню, клиентское приложение загружает экран магазина. При загрузке сервер получает запрос, в ответ на который отправляет текущее состояние скинов у клиента – какие скины выбраны на текущий момент, какие куплены, какие нет. Также, клиентское приложение получает текущее количество валюты у игрока.

При выборе скина информация об этом отправляется на сервер и сохраняется в файле users.json. Если же происходит покупка, то сервер сперва проверяет наличие достаточного объема валюты, и, если проверка пройдена успешно, изменяет состояние скина на «выбрать», что дает возможность выбора этой фигуры. Таким образом, магазин предоставляет функционал по выбору и покупке скинов, за валюту, получаемую после побед в играх. Стоит отметить, что примерка скинов происходит исключительно за счет клиента и нагрузку на сервер не несёт.

Ключевым моментом программной реализации приложения явилась разработка алгоритма «синхронизации», который потребовал слаженной работы как клиента, так и сервера. В дальнейшем этот проект можно использовать как базу для создания подобных клиент-серверных приложений.

Глава 3

Тестирование программы

Тут будет описание тест-кейсов и отчетов тестирования.

Глава 4

Перспективы развития приложения

4.1 Перспективы технического развития

Для технического развития проекта следует решить два основных аспекта: повысить уровень проработанности анимаций и переходов между экранами, а также придумать каким образом защищать комнаты от постороннего входа.

Так как игра не имеет строгого ограничения функционала, можно расширить игровой процесс, добавив другие игры. Основной идеей будет использование тех же фигур и клетчатого поля.

Шагом для привлечения игроков может стать адаптация игры под мобильные устройства. Так как мобильные устройства занимают не малую долю рынка, это повысит доступность игры и позволит играть в перерывах между учебой или работой.

4.2 Перспективы монетизации

Главным источником прибыли в игре имеет смысл выделить рекламу. Рекламные вставки можно добавить на экран ожидания выбора фигур и готовности игроков, а также в магазин скинов.

В игре имеется небольшое число скинов, поэтому стоит увеличить их количество, с целью предоставления пользователям возможности приобретать скины и поля за реальную валюту.

Так как игра имеет строгие правила и во многом схожа с классической игрой крестики-нолики, можно добавить режим «команда на команду». Таким образом победа будет засчитываться не отдельным игрокам, а команде, что позволит проявить коллективное сознание игроков. После этого шага станет возможным проведение турниров и игра выйдет на другой уровень.

Турниры предоставляют дополнительный уровень заработка, поскольку турнир – это своего рода шоу. В случае проведения турниров игра должна обладать анимациями высокого уровня, поскольку игровая логика не предполагает большого количества игровых событий и наблюдатели могут потерять интерес.

Заключение

В результате курсовой работы была реализована игра, позволяющая организовывать коллективные состязания в игре крестики-нолики. Правила игры претерпели небольшие изменения, согласно техническому заданию, составленному до начала разработки. Таким образом, эта игра обладает следующим функционалом:

1. форма авторизации игроков совмещает логин и регистрацию;
2. игра позволяет создавать своего рода игровые комнаты, в которые в дальнейшем можно подключаться с других клиентов;
3. имеется внутри игровой магазин скинов, который влияет на внешний вид фигур и поля.

Игра имеет открытый исходный код и может служить базой для разработки подобных проектов. Отдельно можно выделить использование языков разметки для создания интерфейсов, что позволило достигнуть высокой гибкости и качества разработанного интерфейса. Также, использование библиотеки CEFPython3 позволило выделить часть логики, относящейся исключительно к отрисовке элементов, отдельно, не засоряя основной код клиента, что повышает читаемость и чистоту программного кода.

Поставленная цель была достигнута посредством решения следующих задач:

1. Были определены средства и инструменты разработки. Клиентское приложение использует библиотеку CEFPython3 и языки разметки HTML+CSS. Также, клиентское приложение использует библиотеку Flask для обработки запросов, поступающих от сервера в обратном направлении. Сервер реализует логику обработки запросов посредством библиотеки cpp_httplib.
2. В качестве формата данных выбран JSON.
3. Логика регистрации и авторизации была реализована.
4. Был разработан магазин скинов, который влияет на отображение игровых фигур и плиток полей во время игрового процесса.

5. Был написан код, отвечающий за соединение клиентов друг с другом. Посредником между ними является сервер, который рассылает запросы по клиентским приложениям, подключенным к текущей игре.
6. Реализация приложения была завершена программной логикой игрового процесса. Ключевым моментом на этом этапе стал алгоритм поиска победной комбинации в рамках поля.

В работе используются следующие методы исследования: анализ, синтез, программирование и тестирование.

Анализ выражается в изучении современных методов разработки клиент-серверных приложений и выбор наиболее подходящих под требования проекта. Благодаря синтезу формируется конечная цель и методы её достижения посредством изучения прикладной литературы. Основными методами выступают программирование и тестирование, которые позволяют реализовать игру с выбранным функционалом.

Дальнейшими перспективами развития проекта можно выделить добавление новых скинов и режимов игры.

Список литературы

1. Многоуровневые системы клиент-сервер [электронный ресурс] / Валерий Коржов - 17.06.1997 - режим доступа: https://www.osp.ru/nets/1997/06/142618, дата обращения: 18.05.2021;

2. JSON [электронный ресурс] / JSON Standart - режим доступа: http://json.org/json-ru.html, дата обращения: 18.05.2021;

3. JSON RPC [электронный ресурс] / Wayback Machine - режим доступа: http://web.archive.org/web/20141229142012/http://json-rpc.org/, дата обращения: 18.05.2021;

4. Формат JSON, метод toJSON [электронный ресурс] / Learn Javascript - 30.11.2019 - режим доступа: https://learn.javascript.ru/json, дата обращения: 18.05.2021;

5. Васвани, В. MySQL: использование и администрирование / В. Васвани — М.: «Питер», 2011 — 368 с. — ISBN 978-5-459-00264-5;

6. Суэринг, С. PHP и MySQL. Библия программиста / С. Суэринг, Т. Конверс, Д. Парк — М.: «Диалектика», 2010 — 912 с. — ISBN 978-5-8459-1640-2;

7. Шелдон, Р. MySQL 5: Базовый курс / Р. Шелдон, Д. Мойе — М.: «Диалектика», 2007 — 880 с. — ISBN 978-5-8459-1167-4;

8. Кузнецов, М. MySQL на примерах / М. Кузнецов, И. Симдянов — Спб.: «БХВ-Питербург», 2008 — 952 с. — ISBN 978-5-9775-0066-1;

9. Дюбуа, П. MySQL / П. Дюбуа — М.: «Вильямс», 2006 — 1168 с. — ISBN 5-8459-1119-2;

10. Редмонд, Э. Семь баз данных за семь недель. Введение в современные базы данных и идеологию NoSQL / Э. Редмонд, Д. Р. Уилсон — М.: ДМК Пресс, 2013 — 384 с. — ISBN 978-5-94074-866-3;

11. Бэнкер, К. MongoDB в действии / К. Бэнкер — М.: ДМК Пресс, 2014 — 394 с. — ISBN 978-5-97060-057-3;

12. Chodorow, K. MongoDB: The Definitive Guide, 2nd Edition / K. Chodorow — O'Reilly, 2013 — 432 с. — ISBN 978-1-4493-4468-9;

13. Hows, D. The Definitive Guide to MongoDB: A complete guide to dealing with Big Data using MongoDB, Third Edition / D. Hows, P. Membrey, E. Plugge, T. Hawkins — Apress, 2015 — 376 c. — ISBN 978-1-4842-1183-0;

14. Plugge, E. The Definitive Guide to MongoDB: The NoSQL Database for Cloud and Desktop Computing / E. Plugge, P. Membrey, T. Hawkins — Apress, 2010 — 327 c. — ISBN 1-4302-3051-7;

15. Hoberman, S. Data Modeling for MongoDB / S. Hoberman — Technics Publications, 2014 — 226 c. — ISBN 978-1-935504-70-2;

16. Pirtle, M. MongoDB for Web Development / M. Pirtle — Addison-Wesley Professional, 2011 — 360 c. — ISBN 9780321705334;

17. Фримен, Эр. Изучаем HTML, XHTML и CSS / Эр. Фримен, Эл. Фримен — П.: «Питер», 2010 — 656 с. — ISBN 978-5-49807-113-8;

18. Титтел, Э. HTML, XHTML и CSS для чайников / Э. Титтел, Д. Ноубл — М.: «Диалектика», 2011 — 400 с. — ISBN 978-5-8459-1752-2;

19. Лабберс, П. HTML5 для профессионалов: мощные инструменты для разработки современных веб-приложений / П. Лабберс, Б. Олберс, Ф. Салим — М.: «Вильямс», 2011 — 272 с. — ISBN 978-5-8459-1715-7;

20. Шафер, С. HTML, XHTML и CSS. Библия пользователя / С. Шафер — М.: «Диалектика», 2010 — 656 с. — ISBN 978-5-8459-1676-1;

21. Макфарланд, Д. С. Новая большая книга CSS / Д. С. Макфарланд — Санкт-Петербург: Питер, 2017 — 720 с. — ISBN 978-5-496-02080-0;

22. Общие сведения об XAML [электронный ресурс] / Microsoft Docs - режим доступа: https://docs.microsoft.com/ru-ru/dotnet/desktop/wpf/xaml/?view=netdesktop-5.0&redirectedfrom=MSDN&viewFallbackFrom=netdesktop-5.0, дата обращения: 21.05.2021;

23. Ликнесс, Д. Приложения для Windows 8 на C# и XAML / Д. Ликнесс — Спб.: Питер, 2013 — 368 с. — ISBN 978-5-496-00349-0;