Министерство образования Республики Беларусь

Учреждение образования
БЕЛОРУССКИЙ ГОСУДАРСТВЕННЫЙ УНИВЕРСИТЕТ
ИНФОРМАТИКИ И РАДИОЭЛЕКТРОНИКИ

Факультет компьютерных систем и сетей
Кафедра информатики
Дисциплина «Операционные среды и системное программирование»

К защите допустить:
И.О. Заведующего кафедрой информатики
_____ С. И. Сиротко

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА
курсового проекта
на тему

ШАШКИ ПО СЕТИ

БГУИР КП 1-40 04 01 013 ПЗ

Студент А.С. Кудош

Руководитель Н.Ю. Гриценко

Нормоконтролёр Н.Ю. Гриценко

Минск 2025

СОДЕРЖАНИЕ

ВВЕДЕНИЕ

Развитие цифровых технологий и рост популярности онлайн-взаимодействия делают актуальными проекты, объединяющие классические игры с современными сетевыми решениями. Шашки, как одна из старейших стратегических игр, остаются востребованными, однако их перевод в цифровое пространство с поддержкой многопользовательского режима требует применения специализированных инструментов. Реализация подобного проекта на платформе *C#* и *ASP.NET* с использованием *SignalR* демонстрирует возможности современных веб-технологий для создания приложений с низкими задержками и синхронизацией данных в реальном времени. Это не только способствует изучению актуальных методов разработки, но и решает практическую задачу организации игрового процесса между удаленными пользователями.

Целью курсовой работы является создание функционального веб-приложения для игры в шашки по сети, обеспечивающего стабильное взаимодействие игроков через интернет. Для достижения этой цели необходимо решить комплекс задач, включая проектирование архитектуры клиент-серверного взаимодействия, реализацию игровой логики, интеграцию механизмов передачи данных через *SignalR*, а также разработку интуитивно понятного пользовательского интерфейса. Особое внимание уделяется обеспечению корректной обработки ходов, проверке правил игры и синхронизации состояния доски между участниками.

Поставленная задача предполагает создание двух ключевых компонентов: серверной части на базе *ASP.NET*, отвечающей за обработку игровых событий и валидацию действий игроков, и клиентской стороны, реализованной с использованием *HTML, CSS* и *JavaScript* для визуализации доски и взаимодействия с пользователем. *SignalR* выступает в роли связующего звена, обеспечивая двустороннюю коммуникацию в реальном времени. Сервер должен гарантировать целостность данных, предотвращать конфликты при одновременных ходах и уведомлять игроков об изменениях в игровом состоянии, что формирует основу для надежного и отзывчивого многопользовательского приложения.

Пояснительная записка оформлена в соответствии с СТП 01-2024 [1].

1 АРХИТЕКТУРА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

1.1 Структура и архитектура вычислительной системы

Клиент-серверная архитектура, выбранная для реализации многопользовательской игры в шашки, обеспечивает централизованное управление игровыми процессами и синхронизацию данных между участниками. Основой системы выступает серверная платформа *ASP.NET Core*, которая обрабатывает логику игры, а клиентская часть, реализованная через веб-интерфейс, предоставляет пользователям интерактивное взаимодействие с доской. Такое разделение позволяет минимизировать риски несанкционированного изменения правил игры, так как все критически важные операции выполняются на сервере.

Серверная часть построена с использованием *ASP.NET Core* – кроссплатформенного фреймворка, обеспечивающего высокую производительность и гибкость. Его модульная архитектура позволяет подключать только необходимые компоненты, что снижает нагрузку на систему. Для обработки коммуникаций в реальном времени интегрирована библиотека *SignalR*, которая автоматически выбирает оптимальный протокол передачи данных: *WebSocket* (при поддержке браузером), *Server-Sent Events* (*SSE*) или *Long Polling*. Это гарантирует стабильность соединения даже в условиях ограниченной сети. На сервере размещена игровая логика, включая валидацию ходов по правилам шашек, определение завершения партии (матч, ничья), а также управление игровыми комнатами. Каждая сессия обрабатывается независимо, что предотвращает конфликты при одновременных действиях игроков.

Клиентская часть представляет собой одностраничное веб-приложение (*SPA*), созданное на базе *HTML*, *CSS* и *JavaScript*. Интерфейс динамически отрисовывает доску с использованием *Canvas* или *DOM*-элементов, реагируя на события пользователя (выбор фигуры, перемещение). Для связи с сервером клиент подключается к *SignalR*-хабу, через который получает обновления состояния игры (например, ход противника) и отправляет собственные действия. *JavaScript*-код обрабатывает ввод, валидирует его на стороне клиента (предварительная проверка) и делегирует финальное решение серверу, что сокращает количество ошибочных запросов.

Архитектура системы разделена на три ключевых слоя:

1 Транспортный слой (*SignalR*) – отвечает за передачу данных между клиентом и сервером, абстрагируя технические детали протоколов.

2 Бизнес-логика (*ASP.NET Core*) – инкапсулирует правила игры, управление сессиями и обработку событий.

3 Презентационный слой (веб-интерфейс) – обеспечивает визуализацию и взаимодействие с пользователем.

Для масштабируемости в систему может быть добавлен слой данных с использованием *Entity Framework Core*, который позволит сохранять историю партий, рейтинги игроков или статистику. Это потребует интеграции базы

данных (например, *PostgreSQL* или *SQL Server*), но не нарушит текущую архитектуру благодаря модульности *ASP.NET Core*.

Выбор клиент-серверной модели обоснован необходимостью централизованного контроля над состоянием. Это исключает возможность читерства, так как клиент не имеет доступа к критической логике – все ходы проверяются сервером. Кроме того, *SignalR* обеспечивает мгновенную синхронизацию: при изменении состояния доски (например, перемещении фигуры) сервер рассылает уведомления всем участникам сессии, обновляя их интерфейсы в реальном времени. Подобный подход снижает нагрузку на клиентские устройства, перенося сложные вычисления на сервер, и обеспечивает кросс-платформенность, поскольку веб-интерфейс доступен из любого браузера.

Таким образом, архитектура системы балансирует между производительностью, безопасностью и удобством, предоставляя игрокам плавный опыт онлайн-взаимодействия, а разработчикам – гибкую основу для будущих улучшений.

1.2 История, версии и достоинства

История разработки программного обеспечения для игр, включая шашки, тесно связана с развитием технологий и платформ, на которых осуществляется взаимодействие игроков. Первые компьютерные версии шашек появились еще в 1950-х годах, когда простые алгоритмы начали использоваться для создания игровых программ. С тех пор технологии значительно эволюционировали, что привело к появлению более сложных и оптимизированных решений.

С развитием интернета в 1990-х годах игры по сети стали набирать популярность. Появление многопользовательских режимов открыло новые горизонты для игр, позволяя игрокам из разных уголков мира соревноваться друг с другом. В этом контексте использование таких технологий, как *ASP.NET* и *SignalR*, стало особенно актуальным. *ASP.NET*, введенный в 2002 году, предоставил мощные инструменты для разработки веб-приложений. Он позволяет создавать динамические веб-страницы и управлять взаимодействием с пользователем, что делает его подходящим для реализации игровых проектов.

SignalR, представленный в 2011 году, стал важным дополнением к *ASP.NET*, обеспечивая возможность двусторонней связи между клиентом и сервером в реальном времени. Это особенно полезно для многопользовательских игр, где необходима мгновенная передача данных о ходе игры. *SignalR* автоматически управляет подключениями и поддерживает различные транспортные протоколы, что делает его удобным инструментом для разработчиков.

Достоинства использования *ASP.NET* и *SignalR* в разработке сетевых игр, таких как шашки, включают:

1 Масштабируемость. Возможность легко добавлять новые функции и

поддерживать большое количество пользователей одновременно.

2 Производительность. Высокая скорость обработки запросов и минимальные задержки при передаче данных.

3 Удобство разработки. Наличие мощных инструментов и библиотек упрощает процесс создания и отладки приложений.

4 Кроссплатформенность. Возможность работы на различных устройствах и операционных системах, что расширяет аудиторию игроков.

Таким образом, развитие технологий, таких как *ASP.NET* и *SignalR*, сыграло ключевую роль в создании современных сетевых игр. Эти инструменты предоставляют разработчикам мощные возможности для создания интерактивного и увлекательного игрового опыта, что делает шашки по сети доступными и привлекательными для широкой аудитории.

1.3 Обоснование выбора вычислительной системы

Выбор вычислительной системы для разработки сетевой игры, такой как шашки, является критически важным этапом, который определяет как производительность приложения, так и пользовательский опыт. В данном случае выбор пал на платформу *C#* с использованием *ASP.NET* и *SignalR* по нескольким ключевым причинам.

Во-первых, *C#* является одним из наиболее популярных языков программирования, обладающим богатой экосистемой и поддержкой. Он предлагает высокую читаемость кода и мощные инструменты для разработки, что способствует быстрому созданию и отладке приложений. Благодаря строгой типизации и обширной стандартной библиотеке, разработка становится более предсказуемой и надежной.

Во-вторых, использование *ASP.NET* предоставляет разработчикам возможность создавать мощные веб-приложения с простым управлением состоянием и сессионной информацией. Это особенно важно для многопользовательских игр, где необходимо сохранять данные о ходе игры и состоянии игроков. *ASP.NET* также предлагает интеграцию с различными базами данных, что облегчает хранение и управление игровыми данными.

Технология *SignalR* играет ключевую роль в обеспечении взаимодействия в реальном времени. Она позволяет организовать мгновенную передачу данных между клиентом и сервером, что критически важно для игр, требующих синхронизации действий между игроками. Благодаря автоматическому выбору транспортного протокола, *SignalR* обеспечивает надежное соединение даже в условиях нестабильного интернета, что значительно улучшает пользовательский опыт.

Кроме того, выбор указанных технологий гарантирует хорошую масштабируемость приложения. При увеличении количества пользователей и нагрузки на сервер, возможности *ASP.NET* и *SignalR* позволяют легко адаптироваться и поддерживать оптимальную производительность.

Наконец, использование технологий *.NET* создает возможности для кроссплатформенной разработки. С помощью таких фреймворков, как *Blazor*,

разработка может быть адаптирована для работы на различных устройствах и операционных системах, что расширяет доступность игры для пользователей.

Таким образом, выбор вычислительной системы на базе *C#, ASP.NET* и *SignalR* обоснован высокой производительностью, удобством разработки, возможностями для взаимодействия в реальном времени и масштабируемостью, что делает ее идеальным решением для реализации сетевой игры в шашки.

1.4 Анализ выбранной вычислительной системы

Visual Studio – это мощная интегрированная среда разработки (*IDE*), созданная *Microsoft*, которая идеально подходит для разработки приложений на языке *C#*. Использование этой среды в сочетании с операционной системой *Windows* предоставляет множество преимуществ, способствующих успешному созданию программного обеспечения, включая сетевую игру в шашки.

1.4.1 Удобство разработки. *Visual Studio* предлагает богатый набор инструментов, таких как редактор кода с подсветкой синтаксиса, автозаполнение, отладчик и встроенные средства тестирования. Это значительно упрощает процесс написания и отладки кода. Возможность работы с проектами различного типа, включая веб-приложения, делает *Visual Studio* универсальным инструментом для разработчиков.

1.4.2 Поддержка *.NET* и *ASP.NET*. *Visual Studio* полностью интегрирована с платформой *.NET*, что позволяет разработчикам легко использовать все возможности *C#* и *ASP.NET*. Наличие шаблонов проектов ускоряет старт разработки, а встроенные инструменты управления зависимостями, такие как *NuGet*, упрощают интеграцию сторонних библиотек и компонентов.

1.4.3 Дебаггинг и тестирование. *Visual Studio* предоставляет мощные средства отладки, позволяя выявлять и устранять ошибки в коде на ранних стадиях разработки. Инструменты для юнит-тестирования и интеграционного тестирования помогают обеспечить высокое качество кода и функциональности приложения. Это особенно важно для сетевых игр, где стабильность и производительность критичны.

1.4.4 Интеграция с *Git*. *Visual Studio* включает возможности интеграции с системами контроля версий, такими как *Git*. Это позволяет командам разработчиков эффективно управлять изменениями в коде, вести совместную работу и поддерживать истории версий, что особенно важно для долгосрочных проектов.

1.4.5 Поддержка *SignalR*. Использование *SignalR* для реализации

функционала реального времени в игре легко интегрируется в *Visual Studio*. Среда разработки предоставляет инструменты для управления подключениями и маршрутизацией сообщений, что упрощает реализацию многопользовательского взаимодействия.

1.4.6 Кроссплатформенность. Хотя *Visual Studio* в первую очередь ориентирована на Windows, она поддерживает разработку кроссплатформенных приложений. Используя *.NET Core*, разработчики могут создавать приложения, которые будут работать на различных операционных системах, что расширяет аудиторию пользователей.

Выбор *Visual Studio* на платформе *Windows* для разработки сетевой игры в шашки обоснован высокими возможностями среды, удобством работы, поддержкой современных технологий и инструментов. Это создает оптимальные условия для реализации проекта, обеспечивая как качество кода, так и эффективное взаимодействие в команде разработчиков.

1.5 Вывод

В этой главе была проведена глубокая оценка архитектуры программного обеспечения для разработки сетевой игры в шашки. Был рассмотрен исторический контекст и эволюцию технологий, таких как *C#*, *ASP.NET* и *SignalR*, что позволило понять, как эти инструменты способствуют созданию интерактивных и многопользовательских приложений.

Обоснование выбора вычислительной системы на базе *Visual Studio* и Windows показало, что данная среда разработки предоставляет мощные инструменты для написания, отладки и тестирования кода. Интеграция с платформой *.NET* и поддержка *SignalR* упрощают реализацию функций реального времени, что критично для сетевых игр.

Таким образом, выбранные технологии и инструменты обеспечивают высокую производительность, надежность и масштабируемость приложения, что делает их идеальными для реализации проекта. Эти аспекты создают прочную основу для дальнейшего развития и успешного завершения курсового проекта, позволяя сфокусироваться на создании качественного игрового опыта для пользователей.

2 ПЛАТФОРМА ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Платформа определяет не только технические возможности приложения, но и его производительность, масштабируемость и удобство эксплуатации. В этой главе будет рассмотрена структуру и архитектуру выбранной платформы, а также проанализирована её история, версии и достоинства.

Особое внимание будет уделено обоснованию выбора платформы, что позволит понять, почему именно она была выбрана для реализации проекта. Кроме того, будет проведен анализ операционной системы и других компонентов программного обеспечения, которые обеспечивают эффективную работу приложения. Данная глава создаст основу для глубокого понимания технологической среды, в которой будет развиваться проект, и позволит сформировать представление о её значимости для достижения поставленных целей.

2.1 Архитектура .NET

Структура и архитектура платформы программного обеспечения представляют собой важные аспекты, которые влияют на проектирование и реализацию приложений. В данном контексте мы рассмотрим платформу *.NET*, которая включает в себя язык программирования *C#*, фреймворк *ASP.NET* и библиотеку *SignalR*, используемую для разработки сетевой игры в шашки.

2.1.1 Архитектура .NET. Архитектура .NET основана на многоуровневой модели, которая включает следующие компоненты:

1 *CLR* (*Common Language Runtime*). Это среда выполнения, обеспечивающая выполнение приложений, написанных на языках *.NET*. *CLR* управляет памятью, выполняет код и обеспечивает безопасность приложения.

2 Библиотеки классов (*BCL*). Набор стандартных библиотек, предоставляющих разработчикам готовые решения для выполнения распространенных задач, таких как работа с данными, файловыми системами и сетевыми запросами.

3 *ASP.NET*. Фреймворк для разработки веб-приложений, который упрощает создание динамических веб-сайтов и сервисов. Он включает в себя различные модели разработки, такие как *MVC* (*Model-View-Controller*) и *Web API*, позволяя выбирать подходящий архитектурный стиль.

4 *SignalR*. Библиотека для создания функционала реального времени, обеспечивающая двустороннюю связь между клиентом и сервером. *SignalR* автоматически управляет подключениями и поддерживает различные транспортные протоколы, что делает его идеальным для многопользовательских приложений.

2.1.2 Структура *ASP.NET*. *ASP.NET* имеет модульную структуру, что

позволяет разделять приложение на отдельные компоненты, такие как:

1 Контроллеры. Обрабатывают входящие запросы и управляют логикой приложения.

2 Модели. Определяют бизнес-логику и структуру данных, используемую в приложении.

3 Представления. Отвечают за отображение информации пользователю, формируя пользовательский интерфейс.

Такая структура способствует лучшей организации кода и его поддержанию, а также позволяет легко интегрировать новые функции.

2.1.3 Многоуровневая архитектура. Приложение может быть построено по многоуровневой архитектуре, которая включает:

1 Слой представления. Отвечает за отображение пользовательского интерфейса и взаимодействие с пользователем.

2 Слой бизнес-логики. Содержит всю бизнес-логику и правила, необходимые для обработки данных и выполнения операций.

3 Слой доступа к данным. Управляет взаимодействием с базой данных и другими источниками данных.

Такой подход обеспечивает четкое разделение ответственности между компонентами приложения, что облегчает их тестирование и модификацию.

2.2 История, версии и достоинства

2.2.1 История платформы *.NET*. Платформа *.NET* была разработана компанией *Microsoft* и впервые представлена в 2002 году. Изначально она предназначалась для создания приложений на *Windows* и была частью стратегии *Microsoft* по унификации разработки программного обеспечения. С момента своего появления *.NET* прошла через несколько значительных этапов эволюции:

1 *.NET Framework 1.0* (2002). Первая версия, включавшая основные компоненты, такие как *CLR* и *BCL*, а также поддержку языков, таких как *C#* и *VB.NET*.

2 *.NET Framework 2.0* (2005). Вторая версия добавила новые функции, включая улучшенную поддержку *Generics* и *ASP.NET 2.0*, что значительно упростило разработку веб-приложений.

3 *.NET Framework 4.0* (2010). Эта версия принесла множество улучшений, включая расширенные возможности параллельного программирования и улучшенную работу с данными.

4 *.NET Core (2016)*. Параллельно с *.NET Framework* была разработана кроссплатформенная версия *.NET – .NET Core*. Это позволило разработчикам создавать приложения, работающие не только на *Windows*, но и на *macOS* и *Linux*.

5 *.NET 5* и далее (2020-2023). В 2020 году *Microsoft* объединила *.NET Framework* и *.NET Core* в единую платформу – *.NET 5*, а затем последовали версии *.NET 6* и *.NET 7*, которые продолжают развивать кроссплатформенные

возможности и производительность.

2.2.2 Достоинства платформы *.NET*. Платформа *.NET* обладает множеством достоинств, что делает её популярным выбором для разработки приложений:

1 Кроссплатформенность. С переходом на *.NET Core* и последующие версии разработчики могут создавать приложения, работающие на различных операционных системах, что открывает новые возможности для распространения и использования.

2 Богатая экосистема. Платформа включает в себя обширные библиотеки и фреймворки, позволяющие быстро и эффективно разрабатывать приложения. Наличие *NuGet*-пакетов упрощает интеграцию сторонних библиотек.

3 Поддержка нескольких языков. *.NET* поддерживает различные языки программирования, включая *C#, F#* и *VB.NET*, что позволяет разработчикам выбирать наиболее подходящий инструмент для своей задачи.

4 Высокая производительность. Оптимизированная работа *CLR* и улучшенные возможности компиляции обеспечивают высокую скорость выполнения приложений.

5 Безопасность. Платформа включает встроенные механизмы безопасности, такие как управление доступом и шифрование, что позволяет создавать защищенные приложения.

6 Сообщество и поддержка. Развитое сообщество разработчиков и обширная документация делают процесс обучения и решения проблем более доступным.

2.3 Анализ операционной системы для написания программы

Для разработки сетевой игры в шашки с использованием платформы *.NET* и фреймворков *ASP.NET* и *SignalR* важно не только выбрать подходящие технологии, но и проанализировать операционную систему, на которой будет вестись разработка и эксплуатация приложения. В данном случае основным объектом анализа является *Windows* – операционная система, на которой традиционно разрабатываются приложения на *.NET*.

2.3.1 Особенности операционной системы *Windows*. Операционная система *Windows* обладает рядом характеристик, которые делают её подходящей для разработки программного обеспечения:

1 Широкая поддержка инструментов для разработчиков. *Windows* поддерживает множество интегрированных средств разработки, включая *Visual Studio*, которая предоставляет мощные инструменты для написания, отладки и тестирования кода. Наличие средств управления версиями и интеграции с облачными сервисами также упрощает командную работу.

2 Совместимость с *.NET*. *Windows* является основной платформой для *.NET Framework*, что обеспечивает полную совместимость всех функций и

возможностей платформы. Это позволяет разработчикам использовать все инструменты и библиотеки наилучшим образом.

3 Инструменты для разработки веб-приложений. *Windows* предоставляет доступ к *IIS* (*Internet Information Services*), мощному веб-серверу, который позволяет разрабатывать и тестировать веб-приложения локально, а также размещать их в продакшене.

2.3.2 Преимущества *Windows* для разработки приложений. *Windows* имеет следующие преимущества:

1 Удобство использования. *Windows* имеет интуитивно понятный интерфейс, что облегчает процесс настройки среды разработки и управления проектами.

2 Обширная документация и поддержка сообщества. Пользователи Windows имеют доступ к обширной документации, форумам и сообществам, что позволяет быстрее решать возникающие вопросы и проблемы.

3 Совместимость с оборудованием. Операционная система *Windows* поддерживает широкий спектр аппаратного обеспечения, что позволяет разработчикам использовать различные устройства для тестирования своих приложений.

2.3.3 Ограничения и недостатки. Несмотря на множество преимуществ, *Windows* также имеет некоторые ограничения:

1 Лицензионные расходы. Использование *Windows* может быть связано с дополнительными затратами на лицензии, что может быть важным фактором для независимых разработчиков или небольших команд.

2 Кроссплатформенность. Хотя современные версии *.NET* поддерживают кроссплатформенную разработку, *Windows* по-прежнему является основной средой для разработки, что может ограничить возможности тестирования и отладки на других операционных системах.

2.4 Вывод

Во этой главе была проведена детальная оценка платформы программного обеспечения, используемой для разработки сетевой игры в шашки. Рассмотрены структура и архитектура платформы *.NET*, включая её ключевые компоненты, такие как *CLR, BCL, ASP.NET* и *SignalR*, что позволило понять, как эти элементы взаимодействуют и способствуют созданию высококачественных приложений.

История и версии платформы *.NET* продемонстрировали её эволюцию от *.NET Framework* до современных кроссплатформенных решений, таких как *.NET 5* и далее. Достоинства, включая кроссплатформенность, богатую экосистему и высокую производительность, делают её привлекательным выбором для разработчиков.

3 ТЕОРЕТИЧЕСКОЕ ОБОСНОВАНИЕ РАЗРАБОТКИ ПРОГРАММНОГО ПРОДУКТА

3.1 Обоснование необходимости разработки

В условиях стремительного роста цифровизации и увеличения интереса к онлайн-играм, разработка сетевых игр, таких как шашки, становится особенно актуальной. Несколько факторов подтверждают необходимость создания подобного программного продукта:

1 Рост популярности онлайн-игр. Современные пользователи все чаще предпочитают развлечения, доступные в интернете, что создает высокий спрос на многопользовательские игры. Шашки, как классическая стратегическая игра, могут привлечь как старшее поколение, так и молодежь, заинтересованную в интеллектуальных состязаниях.

2 Удобство и доступность. Онлайн-платформы позволяют игрокам взаимодействовать независимо от географического положения. Это устраняет ограничения, связанные с физическим присутствием, и способствует расширению аудитории.

3 Интерактивность и социальные взаимодействия. Современные технологии позволяют значительно улучшить взаимодействие между игроками. Возможность вести диалоги, соревноваться с друзьями или незнакомыми соперниками создает более увлекательный игровой процесс.

4 Инновационные технологии. Использование таких технологий, как *ASP.NET* и *SignalR*, позволяет создавать высокопроизводительные приложения с низкими задержками, что критично для многопользовательских игр. Эти инструменты обеспечивают надежную синхронизацию данных и мгновенное обновление состояния игры, что значительно улучшает пользовательский опыт.

5 Образовательные аспекты. Разработка игрового приложения вносит вклад в обучение и развитие навыков программирования, проектирования и работы с сетевыми технологиями как для разработчиков, так и для пользователей, которые могут улучшать свои аналитические способности и стратегическое мышление в процессе игры.

Таким образом, разработка веб-приложения для игры в шашки является не только актуальной задачей, но и имеет значительный потенциал для привлечения пользователей и создания сообществ.

3.2 Технологии программирования, используемые для решения поставленных задач

Для разработки сетевой игры в шашки, учитывая требования к производительности, интерактивности и удобству пользователя, выбраны следующие технологии программирования:

1 *C#*. Язык программирования *C#* является основным инструментом для разработки серверной части приложения. Он обладает строгой типизацией,

что позволяет избежать множества ошибок на этапе компиляции, и обеспечивает высокую читаемость кода. *C#* также предлагает обширную стандартную библиотеку, что упрощает выполнение задач, связанных с обработкой данных и логикой игры.

2 *ASP.NET Core*. *ASP.NET Core* – это кроссплатформенный фреймворк для создания веб-приложений. Он предоставляет мощные инструменты для разработки серверной логики и управления состоянием приложения. Благодаря своей модульной архитектуре, *ASP.NET Core* позволяет подключать только необходимые компоненты, что снижает нагрузку на систему и повышает производительность.

3 *SignalR*. *SignalR* – библиотека для создания приложений, работающих в реальном времени. Она обеспечивает двустороннюю связь между клиентом и сервером, что критически важно для многопользовательских игр. *SignalR* автоматически выбирает оптимальный транспортный протокол (*WebSocket*, *Server-Sent Events* или *Long Polling*), что гарантирует надежность соединения даже в условиях нестабильного интернета.

4 *HTML, CSS* и *JavaScript*. Эти технологии используются для создания клиентской части приложения. *HTML* формирует структуру веб-страницы, CSS отвечает за оформление, а *JavaScript* обеспечивает интерактивность. JavaScript позволяет динамически обновлять состояние доски, обрабатывать пользовательские события и взаимодействовать с сервером через *SignalR*.

5 *Visual Studio*. Интегрированная среда разработки (*IDE*) *Visual Studio* обеспечивает мощные инструменты для написания, отладки и тестирования кода. Она поддерживает управление версиями и интеграцию с различными библиотеками, что делает процесс разработки более эффективным.

Эти технологии в совокупности создают надежную основу для реализации функционального веб-приложения, обеспечивая высокую производительность, безопасность и удобство взаимодействия для пользователей.

3.3 Связь архитектуры вычислительной системы с разрабатываемым программным обеспечением

Архитектура вычислительной системы играет ключевую роль в процессе разработки программного обеспечения, особенно в контексте многопользовательских игр, таких как шашки. В данной разработке используется клиент-серверная архитектура, что позволяет эффективно организовать взаимодействие между пользователями и сервером. Рассмотрим, как архитектура системы связана с разрабатываемым программным обеспечением:

1 Централизованное управление. Серверная часть, построенная на *ASP.NET Core*, отвечает за обработку всех игровых событий, валидацию ходов и управление сессиями. Это позволяет централизовать логику игры и минимизировать риски мошенничества, так как важные операции выполняются на стороне сервера.

2 Синхронизация данных. Использование *SignalR* обеспечивает мгновенную синхронизацию состояния игры между всеми участниками. Сервер автоматически рассылает обновления о ходе игры, что позволяет игрокам видеть изменения в реальном времени. Это критически важно для поддержания честной и конкурентной среды.

3 Модульность и масштабируемость. Архитектура *ASP.NET Core* позволяет легко добавлять новые функции и компоненты, что делает систему масштабируемой. При увеличении числа пользователей или внедрении новых игровых режимов, архитектура позволяет адаптироваться к новым требованиям без значительных изменений в коде.

4 Разделение слоев. Архитектура разделена на три основных слоя: транспортный (*SignalR*), бизнес-логика (*ASP.NET Core*) и презентационный (веб-интерфейс). Это разделение упрощает тестирование и модификацию кода, поскольку каждый слой отвечает за свои функции и может быть изменен независимо от других.

5 Обеспечение безопасности. Серверная архитектура гарантирует, что все критически важные проверки и валидации выполняются на стороне сервера, что предотвращает возможность читерства и несанкционированного доступа к логике игры. Это создает доверительную среду для пользователей.

6 Интерактивность и пользовательский опыт. Клиентская часть, реализованная с использованием *HTML, CSS* и *JavaScript*, предоставляет пользователям интуитивно понятный интерфейс для взаимодействия с игрой. *Canvas API* позволяет создать динамическую визуализацию игрового процесса, что улучшает пользовательский опыт.

Таким образом, архитектура вычислительной системы напрямую влияет на функциональность, производительность и безопасность разрабатываемого программного обеспечения. Правильная организация архитектуры обеспечивает надежную и отзывчивую среду для пользователей, что является критически важным для успешного функционирования многопользовательской игры.

3.4 Вывод

В данном разделе было рассмотрено, как архитектура вычислительной системы связана с разработкой программного обеспечения для многопользовательской игры в шашки. Клиент-серверная архитектура, основанная на *ASP.NET Cor*e и *SignalR*, обеспечивает централизованное управление игровыми процессами и синхронизацию данных в реальном времени, что критически важно для создания честного и конкурентного игрового окружения. Модульность системы позволяет легко добавлять новые функции и адаптироваться к растущим требованиям пользователей, что способствует масштабируемости приложения.

Разделение системы на транспортный, бизнес-логический и презентационный слои упрощает тестирование и модификацию кода, обеспечивая высокую степень надежности и безопасности. Серверная часть

отвечает за все критически важные проверки и валидацию действий игроков, что минимизирует риски мошенничества и создает доверительную атмосферу. В то же время клиентская часть, реализованная с использованием современных веб-технологий, предлагает пользователям интуитивно понятный интерфейс и динамическую визуализацию игрового процесса, что значительно улучшает пользовательский опыт.

Таким образом, правильная организация архитектуры вычислительной системы является ключевым фактором, определяющим эффективность, безопасность и качество взаимодействия пользователей с разрабатываемым программным обеспечением.

4 ПРОЕКТИРОВАНИЕ ФУНКЦИОНАЛЬНЫХ ВОЗМОЖНОСТЕЙ ПРОГРАММЫ

Разработка программного обеспечения для сетевой игры в шашки предполагает реализацию ряда функциональных возможностей, которые обеспечат пользователям комфортный и интерактивный игровой процесс. Ниже представлено обоснование и описание ключевых функций, необходимых для успешной работы приложения.

Первая функция – создание и управление игровыми сессиями. Пользователи должны иметь возможность создавать новые игровые комнаты или присоединяться к существующим. Серверная часть приложения управляет сессиями, обеспечивая корректную обработку входящих запросов и распределение игроков по комнатам. Это позволяет организовать игру между друзьями или случайными соперниками, что способствует развитию игрового сообщества.

Вторая функция – обработка ходов и валидация действий. Сервер отвечает за проверку правильности ходов, основанной на правилах игры в шашки. Это включает в себя проверку допустимости перемещения фигур, захвата противника и определение окончания игры (матч или ничья). Данная функция является критически важной для обеспечения честной игры и предотвращения возможных злоупотреблений.

Третья функция – синхронизация состояния игры в реальном времени. Используя *SignalR*, приложение обеспечивает мгновенное обновление состояния игровой доски для всех участников. Это позволяет игрокам видеть ходы противников в реальном времени, что делает игру более динамичной и увлекательной. Синхронизация также включает уведомления о завершении игры и обновления статуса сессии.

Четвертая функция – интерактивный пользовательский интерфейс. Клиентская часть приложения должна предоставлять интуитивно понятный и привлекательный интерфейс, который позволяет легко взаимодействовать с игрой. Использование *HTML, CSS* и *JavaScript* обеспечивает динамическое обновление визуальных элементов, что значительно улучшает пользовательский опыт. Интерфейс должен быть адаптивным, чтобы корректно отображаться на различных устройствах.

Таким образом, перечисленные функциональные возможности формируют основу для полноценного игрового опыта, обеспечивая как технические, так и пользовательские аспекты разработки. Каждая из функций играет важную роль в создании увлекательной и безопасной среды для игры в шашки, что отвечает современным требованиям игроков и способствует популяризации приложения.

4.1 Вывод

В результате анализа функциональных возможностей программного обеспечения для сетевой игры в шашки было определено несколько ключевых

функций, необходимых для обеспечения комфортного и интерактивного игрового процесса. Создание и управление игровыми сессиями, обработка ходов и валидация действий, а также синхронизация состояния игры в реальном времени являются основными компонентами, которые обеспечивают честность и динамичность игры. Интерактивный пользовательский интерфейс, разработанный с использованием современных веб-технологий, способствует улучшению пользовательского опыта и адаптивности приложения. Все эти функции вместе создают безопасную и увлекательную среду для игроков, что соответствует современным требованиям и способствует успешной популяризации игры.

5 АРХИТЕКТУРА РАЗРАБАТЫВАЕМОЙ ПРОГРАММЫ

Архитектура разрабатываемой программы играет ключевую роль в определении структуры, функциональности и взаимодействия различных компонентов приложения. Для сетевой игры в шашки была выбрана клиент-серверная архитектура, которая позволяет эффективно управлять игровыми сессиями и обеспечивать взаимодействие между пользователями в реальном времени. Эта архитектура обеспечивает централизованное управление логикой игры и синхронизацию данных, что критически важно для многопользовательских приложений. В данном разделе рассмотрим основные компоненты архитектуры, их взаимодействие и влияние на производительность и пользовательский опыт.

5.1 Общая структура программы

Общая структура программы для сетевой игры в шашки основана на клиент-серверной архитектуре, которая обеспечивает эффективное взаимодействие между пользователями и сервером. Структура включает в себя несколько ключевых компонентов, каждый из которых выполняет свою уникальную роль:

1 Серверная часть

а Сервер приложений. Реализован на основе *ASP.NET Core*, отвечает за обработку игровых логик, управление сессиями и валидацию действий игроков. Сервер обеспечивает централизованное управление и хранение данных.

б База данных. Используется для хранения информации о пользователях, игровых сессиях и истории игр. Это позволяет сохранять прогресс игроков и статистику.

2 Клиентская часть

а Интерфейс пользователя. Реализован с использованием *HTML, CSS* и *JavaScript*. Обеспечивает интерактивность и визуальное представление игрового процесса. Пользовательский интерфейс адаптивен и удобен для различных устройств.

б Логика клиента. Обрабатывает ввод пользователя, отправляет запросы на сервер и обновляет отображение состояния игры на экране. Использует библиотеку *SignalR* для получения данных в реальном времени.

3 Коммуникационный слой. *WebSocket* и *SignalR* обеспечивают двустороннюю связь между клиентом и сервером, позволяя мгновенно передавать данные о ходе игры и изменениях состояния. Это создает динамичное и отзывчивое взаимодействие.

Эта структура обеспечивает надежное функционирование программы, позволяя пользователям наслаждаться игрой в реальном времени, а также гарантирует высокую производительность и безопасность. Правильная организация компонентов архитектуры способствует эффективному

масштабированию и легкости в поддержке приложения.

5.2 Описание функциональной схемы программы

Функциональная схема программы для сетевой игры в шашки отражает основные процессы и взаимодействия между компонентами системы. Она иллюстрирует, как различные модули взаимодействуют друг с другом, обеспечивая пользователям целостный и интерактивный опыт. Основные компоненты функциональной схемы:

1 Пользовательский интерфейс (*UI*). Обеспечивает взаимодействие игрока с игрой. Пользователь вводит команды (например, ходы фигур) через интерактивные элементы интерфейса, такие как кнопки и игровая доска.

2 Клиентская логика. Обрабатывает пользовательский ввод и отправляет запросы на сервер. После получения ответов от сервера обновляет состояние игры и интерфейс. Включает обработку событий, таких как нажатия клавиш и клики мыши.

3 Серверная часть. Принимает запросы от клиентов и обрабатывает их. Здесь происходит валидация ходов, управление игровыми сессиями и обновление состояния игры. Сервер отвечает за распределение игроков по комнатам и отправку обновлений в реальном времени.

4 База данных. Хранит информацию о пользователях, их профилях, играх. Сервер взаимодействует с базой данных для сохранения и извлечения данных по мере необходимости.

5 Коммуникационный канал (*SignalR*). Обеспечивает двустороннюю связь между клиентом и сервером. Используется для передачи сообщений о ходе игры, обновления состояния и уведомлений о событиях, таких как завершение игры.

Процесс взаимодействия:

1 Пользователь запускает приложение и попадает на главную страницу, где может создать новую игру или присоединиться к существующей сессии.

2 При создании или присоединении к игре сервер обрабатывает запрос и обновляет состояние игрового процесса.

3 Пользователь делает ход, который отправляется на сервер для валидации.

4 Сервер проверяет допустимость хода и обновляет состояние игры.

5 Обновленная информация о ходе игры отправляется всем участникам через *SignalR*, что позволяет игрокам видеть изменения в реальном времени.

6 Вся статистика и история игр сохраняются в базе данных для дальнейшего доступа и анализа.

Эта функциональная схема демонстрирует, как все компоненты системы взаимодействуют между собой, обеспечивая пользователям гладкий и увлекательный игровой процесс. Правильная реализация этих процессов является ключом к успешной работе приложения.

5.3 Описание блок-схемы алгоритма программы

Блок-схема алгоритма программы для сетевой игры в шашки иллюстрирует последовательность действий и принятия решений в процессе игры. Она служит наглядным представлением логики работы приложения и описывает основные этапы взаимодействия между пользователем, клиентом и сервером. Основные элементы блок-схемы:

1 Начало. Инициализация приложения и загрузка пользовательского интерфейса.

2 Авторизация пользователя. Пользователь вводит свои учетные данные, если учетные данные корректны, переход к следующему шагу; если нет – вывод сообщения об ошибке.

3 Выбор действия. Пользователь выбирает: создать новую игру или присоединиться к существующей, если выбрана новая игра, переход к созданию сессии; если присоединение – запрос списка доступных игр.

4 Создание игровой сессии. Сервер создает новую игровую комнату и добавляет пользователя в нее. Отправка уведомления другим игрокам о создании новой игры.

5 Присоединение к игровой сессии. Пользователь выбирает игровую комнату и подключается к ней. Сервер обновляет статус сессии и уведомляет всех участников.

6 Игровой процесс. Игрок делает ход (выбор фигуры и перемещение). Сервер проверяет, допустим ли ход, если ход допустим, обновляется состояние игры и отправляется уведомление всем игрокам; если нет – выводится сообщение об ошибке.

7 Проверка состояния игры. Проверка на наличие победителя или окончания игры, если игра завершена, переход к этапу завершения; если нет – возврат к игровому процессу.

8 Завершение игры. Сервер сохраняет результаты в базе данных, обновляет статистику игроков и уведомляет всех участников об окончании игры.

9 Выход из игры. Пользователь может выйти из текущей сессии или завершить приложение.

10 Конец. Завершение работы приложения.

Блок-схема алгоритма программы демонстрирует четкую последовательность действий и взаимодействий, позволяя разработчикам и пользователям лучше понять логику работы приложения. Каждый этап алгоритма важен для обеспечения корректного функционирования игры и создания увлекательного пользовательского опыта.

5.4 Вывод

В данном разделе была описана блок-схема алгоритма программы для сетевой игры в шашки, которая наглядно иллюстрирует последовательность действий и взаимодействия между пользователями, клиентом и сервером.

Каждый этап алгоритма, начиная от авторизации пользователя и выбора действия, и заканчивая завершением игры, играет ключевую роль в обеспечении корректного и динамичного игрового процесса.

Четкая структура алгоритма позволяет разработчикам легко отслеживать логику приложения и вносить необходимые изменения, а также гарантирует пользователям интуитивно понятный и увлекательный опыт. Эффективное управление игровыми сессиями, валидация действий и синхронизация состояния игры – все это способствует созданию стабильной и безопасной среды для игры. В итоге, хорошо продуманная блок-схема является основой для успешной реализации функциональности и повышения качества приложения.

СПИСОК ЛИТЕРАТУРНЫХ ИСТОЧНИКОВ

[1] Коллектив авторов Стандарт предприятия / Коллектив авторов. – Минск БГУИР, 2024. – 178 с.