МИНИСТЕРСТВО ОБРАЗОВАНИЯ И НАУКИ ХАБАРОВСКОГО КРАЯ
Краевое государственное бюджетное профессиональное образовательное учреждение
«Николаевский – на Амуре промышленно-гуманитарный техникум»

Предметно-цикловая комиссия
экономики и информационных технологий

Допустить к защите
Зам. директора по УПР
_____
«____»_____2018 г.

ДИПЛОМНЫЙ ПРОЕКТ

Разработка персонального помощника для игры в шашки
(тема)

НПГТ 09.02.03.01 ПКСз-41-С ПЗ

Работу выполнил	_____	Шахов В. Ю.
	подпись	Ф.И.О.
Руководитель работы	_____	Южаков П. Н.
	подпись	Ф.И.О.
Рецензент	_____	Кайдалов Р. Ю.
	подпись	Ф.И.О.
Нормоконтроль	_____	Боровик С. В.
	подпись	Ф.И.О.

г. Николаевск-на-Амуре
2018 г.

Содержание

Введение

В настоящее время информационные технологии развились настолько, что подобно ластику стёрли границы и расстояния мешавшие найти партнёра и единомышленника в каком либо занятии, позволяя вести общение и обмениваться информацией в реальном времени практически без временных задержек. В то же время это, без сомнения крайне важное свойство технологий, роль которого в современном мире крайне сложно переоценить, несёт и ряд отрицательных факторов, среди которых один из главных — это «оторванность» от реального мира. Зачастую, найти собеседника и партнёра по игре гораздо проще в виртуальной среде Сети, нежели в лице коллеги, одноклассника, соседа по лестничной площадке. Да что там площадка! Даже сосед по комнате порой не рассматривается в качестве кандидата, настолько огромный и разнообразный выбор предлагает та другая, альтернативная реальность.

Подобные тенденции влекут за собой постепенную атомизацию общества, которая уже давно вплотную приблизилась к самой базовой ячейке нашего общества – семье. Нарушается контакт между детьми и родителями, причём происходит это уже с самого раннего возраста. Того самого, когда ребёнок лишь начинает понимать – что есть окружающий его социум, как с ним взаимодействовать. Он подобно зеркалу повторяет те поведенческие модели, которые непосредственно наблюдает. Хорошо если образцом будет человек, способный рассуждать логически, признающий свои ошибки, способный осмысливать свои ошибки и делать из них выводы, но в то же время упорный и настойчивый в достижении своей

цели. Огромное число детей, начиная уже с дошкольного возраста, многие часы проводит в виртуальных пространствах Сети практически без контроля со стороны взрослых. Чему в таком возрасте они могут там научиться? Какие примеры для подражания получат? Большой вопрос. В лучшем случае это будет бесконечное потребление современных мультфильмов или игра с бездушной машиной или другим удалённым игроком. В таких играх сложно испытывать совместные эмоции, они замыкаются и усиливаются во внутреннем пространстве игрока, а затем могут выплёскиваться в искажённой, гипертрофированной форме.

Потому очень важно, чтобы было как можно больше объединяющих занятий между родителем и ребёнком. Одним из таких дел может стать обучающая логическая игра призванная не только развлечь и развить ребёнка, но и помочь ему сформировать в себе правильные стереотипы поведения и черты характера.

Данная работа имеет своей целью предложить один из вариантов решения в виде приложения для персонального компьютера позволяющего вести совместную игры в шашки между родителем и ребёнком. В ней будут пошагово рассмотрены основные этапы создания виртуальной доски для игры

В главе 1 кратко рассмотрена история возникновения и развития игры, рассказано как изменило взгляд на неё возникновение вычислительной техники, которая впоследствии стала доступна массовому пользователю. Рассказывается о текущем положении дел в индустрии казуальных игр и шашечных программ в частности. Рассмотрены предпосылки мотивировавшие данную работу.

Глава 2 описывает правила и необходимый реквизит для игры в одну

из разновидностей шашек – «русские шашки». Выполняется разработка архитектуры будущего приложения, которое разделяется на отдельные модули, каждый из которых отвечает за свою часть игрового процесса, будь то взаимодействие игрока с реквизитом игры, управление ходом игры на основании описанных правил, отображение реквизита, записи хронологии ходов текущей партии, поиск в архиве уже завершённых партий и их просмотра.

Далее в главе детально рассматривается каждый из модулей, проговариваются соглашения по форматам данных необходимых для хранения информации и обмена между модулями. Разрабатываются ключевые алгоритмы и данные необходимые для реализации заданной функциональности модулей.

Глава 3 описывает процесс переноса ранее разработанной архитектуры и моделей данных на язык инструментальных средств разработки. Показаны метод переноса разработанных абстрактных алгоритмов в пространство ограничений, способы организовать взаимодействие различных технологий в рамках одного проекта.

Приложения в конце документа содержат рисунки и фрагменты псевдокода призванные наглядно отобразить процесс исполнения проекта.

1 Анализ предметной области

1.1 Краткая история развития

Игра «Шашки» известна человечеству с давних времён. Существуют две исторических версии её происхождения. Общепринятой считается та, по которой шашки пришли в мир из Древнего Египта, где служили одним из главных развлечений фараонов и знати (после охоты и рыбалки). Вторая версия, приписывает авторство Паламеду, греческому воину участвовавшему в осаде Трои. Ожидание было столь долгим (осада длилась 10 лет) и скучным, что он выдумал эту игру, используя вместо доски участок земли, расчерченный кончиком копья.

Игра оказалась столь увлекательна, что распространилась далеко за пределы царства фараонов. Археологические находки говорят что в шашки играли на Руси, и территории нынешних Швеции, Дании и Норвегии ещё в III-IV веках.

Правила древних игроков мало отличались от тех что используются в современности. Даже доска размечалась на 64 клетки. Шашки были двух цветов – чёрные и белые. Они символизировали две сражающиеся армии перед боем. После начала сражения, ходить разрешалось только вперёд, отступать, как и в бою, было непозволительно. Как победитель переступает побеждённого, так и взятие шашки противника происходило при перескакивании одной шашки через другую, принадлежащую противнику. Достигая противоположной стороны доски, словно заходя в тыл противнику, шашка становилась дамкой. При этом повышалась её

мобильность, она приобретала новые возможности и становилась грозным фактором на поле боя. Сражение продолжалось до полного уничтожения всех боевых единиц противника.

За прошедшие тысячелетия набор возможных правил значительно расширился и некогда одна игра разбилась на множество подвидов, которых, в настоящее время, исчисляется десятками.

Они отличаются размером игровой доски (до 10х10 клеток в международных шашках), способом взятия фигур («турецкий удар» предписывает оставлять условно срубленные шашки на поле до завершения хода, с запретом повторного их взятия), ограничением времени на одну партию и даже конечной целью игры, которой надо достигнуть (в «Русских поддавках», например, надо отдать все свои фигуры противнику. Выигрывает тот, кто остался без армии).

В XIX веке начали проводиться чемпионаты по шашкам, так-же они были причислены к видам спорта.

Еще на заре развития компьютерной техники появились первые программы позволяющие играть в шашки с машиной. Первые версии использовали ввод-вывод на базе перфокарт (обмен информацией о ходах в заданной нотацией), и играли на уровне любителя. С возрастанием мощностей компьютеров (увеличением объёма оперативной памяти, растущим числом выполняемых процессором операций, в единицу времени, появлением новых устройств ввода-вывода информации) стало возможным просчитывать большее число возможных ходов (за обозримый отрезок времени) программы стали играть много сильнее, выиграть удавалось далеко не каждому. В обиход вошло визуальное отображение шахматной доски, игра стала идти в реальном времени.

На сегодняшний день, машины, в игре в шашки, превзошли человека. Дерево ходов, во многих вариантах игры, просчитано полностью и шансов выиграть у программы практически нет. Если только не понизить «интеллект» искусственного противника внеся элемент случайности в виде преднамеренных ошибочных ходов. Гроссмейстеры часто используют их в качестве тренажеров, и для удобства проведения анализа игровых ситуаций.

Стоит упомянуть и о таком явлении как шашечные задачи, которые представляют собой определённое расположение фигур на доске, и содержащее условие решения (постановка цели). Решением может являться взятие всех фигур за строго отведённое число ходов, выигрыш одной дамкой из заданной расстановки и многое другое. В этом случае виртуальная шашечная доска незаменимый помощник в проверке верности решения или соответствия его граничным условиям задачи.

1.2 Текущее положение дел

Мир современных информационных технологий предлагает нам массу различных приложений предназначенных для развлечения. Как правило, это различные социальные сети, а также, так называемые «казуальные игры» – компьютерные игры, не требующие от игрока сложных действий и особой усидчивости. Очень много разнообразных приложений, позволяющих сыграть, в том числе, и в шашки. Взяв для примера несколько различных приложений можно увидеть как общие их черты так и существенные различия в реализации.

Подавляющее число приложений настолько просто, что предлагает

для игры лишь необходимый минимум – доску и фигуры на ней. Минимальные настройки, отсутствие выбора набора правил игры и примитивная графика, тем не менее позволяют неплохо отдохнуть во время игры, отмобилизовав свои умственные способности.

Приложение Checkers Elite более сложное и интересное в исполнении. Очень красивая графика и масса вариантов игры. Различные режимы и уровни сложности, широкий диапазон настроек, возможность игры в сети, плюс выполнение различных заданий для повышения игрового рейтинга.

Шашки от English Checkers ей ничем не уступают, а графически даже более привлекательны. Приложение поддерживает массу разновидностей игры (русские шашки, турецкие, бразильские, испанские, поддавки и много других). Также присутствует огромный выбор настроек оформления, анимации, позволяющих сделать игру более интересной.

И подобных приложений очень много. Они разные по объёму, качеству исполнения. Но всех объединяет возможность играть независимо от наличия партнёра. Партнёром становится искусственный интеллект.

Из многообразия шашечных программ вытекает сложность (практически невозможность) привнести в игровой процесс или оформление что-то новое, ранее не встречавшееся. Искусственный интеллект уже давно способен просчитать действия на много ходов вперёд и смоделировать ситуацию из которой не сможет выбраться даже опытный игрок.

Популярность шашечных приложений подтверждается большим спросом в сети Интернет. На примере PlayMarket (платформа дистрибуции ПО для ОС Android) можно увидеть, что число скачиваний игр в шашки переваливает порой за отметку в 10 миллионов скачиваний для некоторых

их представителей. Небольшой объём, лёгкость установки на различные типы устройств, мобильность позволяют играть как дома, так и в дороге используя телефон или планшет.

В современном мире высоких технологий подобные устройства есть практически у каждого, даже у детей, что позволяет человеку постоянно присутствовать в виртуальном пространстве сети Интернет и пользоваться доступными в нём развлечениями. В наше время, как уже говорилось, это в основном социальные сети предлагающие несметное число «казуальных» игр, среди которых присутствуют и игры в шашки. Но нести они могут не только комфорт и доступность.

В самой по себе игре в шашки никакой опасности нет. Наоборот присутствует масса положительных последствий. Опасность виртуальной игры заключается в отрыве от реальности. Не только психологи и педагоги, но и сами разработчики игр признают это. На детей это влияет гораздо сильнее, нежели на взрослого человека.

Соперник по игре, находящийся по ту сторону сети как-бы обезличивается либо вовсе подменяется машиной, что не позволяет испытать весь спектр эмоций от игры, которая может принести как радость победы, так и горечь поражения. А ведь это один из способов социализации, выработки поведенческих навыков и реакций, искажение которых может стоить дорого в будущем. Кроме того машина не способна обсудить сложившуюся расстановку, необходимость смены стратегии, последствия сделанного хода.

Польза от игры в шашки имеет образовательный и развивающий характер. Поскольку развитие ребёнка лучше всего происходит в игровой форме, игра в шашки хорошо способствует гармоничному развитию

интеллекта ребёнка. Процесс обучения игре стимулирует развитие способности ориентироваться на плоскости, мышления, способности делать умозаключения и выносить суждения. Учит запоминать, обобщать, предвидеть результаты своей деятельности, формирует волю к победе. В игре в шашки необходимы такие качества, как дальновидность, смелость, хладнокровие, настойчивость и изобретательность. Усидчивость, терпение и дисциплинированность способны определить исход партии в не меньшей степени.

Игра в шашки осваивается гораздо проще чем её старшая сестра – шахматы, особенно маленькими детьми. Она может быть первым шагом перед погружением в сложный и многовариантный мир шахмат.

Не является секретом существование в наше время проблемы изолированности. Нередко, даже взрослые люди жертвуют остающимися крохами свободного времени, проводя его в виртуальной реальности различных сетей. Десятки и сотни тысяч различных игр способны вызывать самый разнообразный спектр эмоций. Наиболее скрытую угрозу среди них представляют так называемые онлайн-игры предоставляющие возможность общения и взаимодействия со своими удалёнными игроками, что призвано стереть расстояние между участниками. Больше не нужно искать себе соперника или партнёра по игре. Не выходя из дома можно сыграть с кем угодно. Для ребёнка это опасно отсутствием возможности знакомиться, общаться с реальными людьми. Развивается асоциальность. Что дополнительно усиливается невозможностью понести ответственность за те или иные свои действия или поступки.

Конечно, детей сложно отвлечь от общения с компьютером или мобильным устройством которые давно стали частью жизни наряду со

столовыми приборами. Но попытаться частично решить эту проблему в пределах пары родитель-ребёнок всё-же можно. Использовать устройство в совместном виде деятельности, форму проведения досуга и обучения. Обучение игре в шашки, когда наставником и противником является родитель может стать той новой особенностью которая будет отличать разработанное приложение от множества уже существующих.

Ребёнок не просто играет, осваивая правила и техники игры наугад, а обучается анализировать ходы, ведёт запись ходов партии. Может просмотреть запись игры неоднократно, включая любой её произвольный участок и совместно с наставником оценить верность того или иного хода, найти ход который был переломным в ходе партии. Он наблюдает за живой реакцией родителя в той или иной ситуации и непроизвольно научается копировать её в сходных обстоятельствах.

В рамках данной работы будет разработано приложение для персонального компьютера реализующее виртуальную доску для игры в «русские шашки», имеющую возможность записи ходов партии, простой фильтр поиска сыгранных партий между двумя заданными игроками, а также возможность их пошагового просмотра и возможности доиграть неоконченную партию с того места где она была прервана.

1.3 Описание реквизита и правил игры

Игровая доска:

- состоит из клеток двух цветов, чёрного и белого
- число клеток каждого цвета одинаково
- клетки размещены в сетку 8x8 ячеек, общим числом 64

- цвет клеток чередуется как в строках так и столбцах

- угловая клетка внизу слева от игрока должно быть чёрным

Фигуры (далее «шашки»):

- могут быть одного из двух цветов – чёрного и белого

- шашка имеющая особые свойства называется «дамкой»

Начальное размещение:

- каждому игроку принадлежит по 12 шашек

- шашки располагаются строго на полях чёрного цвета

- шашки игрока занимают позиции в трёх первых, ближних к игроку горизонталях

- шашки белого цвета занимают нижние позиции

Правила перемещения обычной шашки:

- перемещается на одну клетку вперёд по диагонали , если она не занята другой шашкой или не горизонталь противника

- ходить назад не разрешается

- при достижении первой горизонтали противника (последней относительно игрока) становится дамкой и к ней применяются модифицированные правила

Отличия в правилах перемещения для дамки:

- перемещается по всей длине диагонали до тех пор пока не окажется на первой или последней горизонтали, или не встретится поле занятое другой шашкой

- разрешается ходить назад

Правила взятия фигур противника для обычной шашки:

- если на соседней (по диагонали) клетке стоит шашка противника и при этом следующая за ней клетка пуста, то возникает возможность «взять» (далее по тексту) такую вражескую шашку

- если на доске возникла ситуация позволяющая произвести взятие то игрок обязан его выполнить

- при взятии фигуры противника шашка перемещается на следующую за срубленной шашкой клетку и ход передаётся другому игроку

- разрешается рубить как вперёд так и назад

- если после перемещения шашки при взятии снова возникает возможность срубить, то необходимо выполнять это действие до тех пор пока возможность не исчезнет

- при наличии двух и более альтернативных возможностей взятия выбирается любой из них на усмотрение игрока

- если шашка становится дамкой и при этом существует возможности взятия то они выполняются уже в качестве дамки

- срубленная шашка удаляется с игровой доски

Отличия в правилах взятия фигур противника для дамки:

- рубит по всей длине диагонали

- при взятии не обязана останавливаться сразу после жертвы а может следовать до следующей преграды

Условия завершения партии:

-исчезновение с доски всех шашек одного цвета, при этом победа отдаётся игроку чьи шашки остались на игровом поле

-невозможность совершить ни одного хода (победа отдаётся игроку чей ход привёл к сложившейся расстановке)

2 Архитектура приложения

2.1 Разделение на модули

Разбиение приложения на отдельные слабо связанные друг с другом модули позволяет упростить разработку за счёт того, что разработчик может сфокусироваться на меньшем участке задачи не пытаясь удержать в поле внимания большое количество зависимостей между объектами и тратить (порою существенное) время на решение проблем возникающих из-за их нарушения. Кроме того подобный подход позволяет экономить время в случае командной разработки, когда каждая группа имеет свой участок ответственности. Её работа не зависит от деталей реализации остальной части проекта и разработчики лишь должны придерживаться определённых соглашений по внешнему интерфейсу модуля (типов и структур данных для обмена, методов и функций которыми предоставляют услуги во внешний мир).

Необходимо выделить основные функциональные части проекта, которые можно обособить относительно друг друга.

Доска.

Ответственна за графическое отображение игрового поля и текущей расстановки фигур на нём. Позволяет посредством манипулятора типа «мышь» переместить фигуру с одной клетки на другую, по желанию игрока. В то же время она ограничивает его свободу блокируя возможность поднять фигуру противника или не предназначенную для хода в соответствии с текущими правилами. Решение о передаче хода или завершении партии принимается так же ей.

Доска способна обмениваться данными с внешним миром. Она может отправлять информацию о совершённом ходе (текущую расстановку, запись хода в шашечной нотации и так далее), о завершении партии (цвет победителя, счёт и причина завершения). Может отобразить по запросу от внешнего объекта конкретную расстановку фигур и дополнительную информацию типа комментария к ней. Может запросить отдельный модуль (Решатель) рассчитать возможные ходы для конкретного игрока в текущей расстановке.

Доска может работать в нескольких режимах.

Игра – фигуры доступны для перемещения игроками, на ход игры влияют правила.

Пауза – игра приостановлена, фигуры недоступны игрокам.

Просмотр – пошаговый просмотр записанных партий, кроме расстановки отображается дополнительная информация типа комментария к ходу. Фигуры недоступны игрокам.

Решатель.

Ответственен за поиск множества возможных ходов на основании текущей расстановки и цвета активного игрока, которые он получает на вход.

На выходе модуль возвращает структуру в которой упорядочена информация о всех фигурах доступных к ходу, фигурах способных совершить взятия. Для каждой из них указаны номера клеток на которые они могут переместиться и фигура противника, что должна быть удалена с доски, а так же возможность повышения в дамки простой фигуры, если ей достигнут конец поля.

На основании этой информации доска управляет ходом игры.

Архив.

Данный модуль реализует накопление информации об участниках и совершённых при розыгрыше партии ходах. Кроме того он ответственен за наполнение собранной информацией банка данных сыгранных партий, предварительно конвертируя её в форматы используемые конкретной СУБД (система управления базами данных).

Для режима пошагового просмотра партии модуль предоставляет возможность выдачи хранимых в базе данных фактов в разрезе выбранной партии.

Список ходов.

Отображает список ходов сделанных в ходе партии, записанных в стандартной шашечной нотации. В режиме просмотра партии из архива, дополнительно позволяет перемещаться по нему пошагово, либо перейти к произвольному ходу, номер которого может вернуть в ответ на запрос.

Фильтр.

Отображается в режиме просмотра архива игр и предоставляет набор фильтров в виде элементов графического интерфейса позволяющих выбрать партию для просмотра. Список имеющихся записей игры содержит дату её начала и окончания (либо кнопку позволяющую её возобновить с места остановки).

Окно приложения.

Предоставляет графический интерфейс приложения содержащий все необходимые элементы отображения информации и управления, опирается в своей работе на все описанные выше модули, увязывая их взаимодействие друг с другом.

Основной цикл работы приложения в режиме игры, после его запуска,

отображения интерфейса и начальной расстановки фигур проходит следующим образом:

- Доска передает текущую расстановку и цвет игрока Решателю, получая в ответ список возможных ходов.

- игрок совершает один или несколько доступных ходов, в соответствии с правилами игры.

- информация о простом ходе передаётся в главное окно посредством сообщения содержащего запись хода в игровой нотации и новая, ставшая текущей, расстановка фигур на поле;

- если в следствии сделанного игроком хода произошло взятие фигуры противника, то интерфейсу отправляется сообщение об этом факте содержащее кроме стандартных атрибутов хода ряд дополнительных (цвет взятой фигуры, её статус);

- если список ходов пуст, то интерфейсу отправляется сообщение о завершении игры и цвет победителя, который, в свою очередь, отображает этот факт в окне графически и ожидает начала новой партии или перехода в режим просмотра архивных игр;

- интерфейс передаёт информацию о совершённом ходе в Список Ходов для отображения и Архив для записи его в базу данных;

- право хода передаётся доской другому игроку и приложение ожидает попытки следующего хода или перехода в режим просмотра;

Основной цикл в режиме просмотра архивных партий происходит следующим образом:

- доска и список переводятся в режим просмотра, отображается набор фильтров для поиска и выбора партии (в режиме игры фильтры подразумеваются скрытыми);

- на основании установленных пользователем значений отбирается список партий из локальной базы данных, а при выборе конкретной партии Архив возвращает структуру содержащую информацию о соответствующих ходах.

- список ходов заполняется записями в игровой нотации.

- доске, для отображения отправляется элемент структуры описывающий начальную расстановку на поле (сюда может входить различная информация, не только расстановка фигур, но и комментарии, время затраченное на принятие решения и прочие, второстепенные атрибуты).

- по факту нажатия управляющих кнопок графического интерфейса «вперёд» и «назад» (условно) или двойному щелчку на произвольном элементе списка ходов происходит передача соответствующих данных доске для смены отображения. Новый элемент списка помечается активным.

- приложение ожидает выбора очередного хода или смены режима на игровой.

2.2 Алгоритмы и структуры данных

2.2.1 Модуль Решатель

Наверное, одна из самых сложных частей проекта. Данный модуль должен:

- принимать на вход состояние доски и цвет игрока имеющего право совершить ход;

- выдавать на выходе следующие результаты:

- набор всех доступных для хода фигур;

- набор фигур игрока способных взять фигуру противника;

- набор фигур противника доступных для взятия;

- набор клеток доступных для хода каждой из фигур игрока;

- определять какая из фигур будут снята с доски по итогам хода;

- клетки при перемещении на которые простая шашка может стать дамкой;

Прежде всего необходимо определиться с моделью представления доски и расстановки фигур на ней.

Существует не одно решение этого вопроса. Например, организовать её в виде двухмерного массива, каждый из элементов которого описывает клетку (цвет, наличие фигуры и её статус и прочие необходимые атрибуты), а поиск клеток для оценки возможности хода или проведения атаки через неё вычислять с помощью арифметических действий. В этом случае должны выполняться проверки граничных условий, чтобы оставаться в рамках модели.

Этот вариант – первый, который приходит в голову, при постановке задачи, ведь он наглядно отображает ту доску, которую мы видим перед собой во время игры. По сути – это единственное его преимущество.

Недостатки же в том, что требуется избыточность данных, потому как в массиве будут существовать не используемые в игре элементы соответствующие белым клеткам, которые будут занимать такой же объём памяти как и чёрные. Довольно громоздким и неуклюжим будет расчёт следующей по диагонали клетки для хода в зависимости от цвета игрока,

проверки выхода за границы доски адаптация доски к размерности отличной от стандартной.

Идея второго варианта заключается в том, что клетки доски (причём не все, а только чёрные игровые) можно представить в виде ориентированного графа. Каждая из клеток которого указывает на следующую доступную для хода в каждом из четырёх направлений. Узлы лежащие на границе доски в заданном направлении вместо указателя на элемент содержат пустое значение. Крайние горизонтали имеют соответствующий признак позволяющий принять решение о повышении статуса фигуры в случае, если такая клетка достигнута. Естественно, каждый узел содержит информацию о наличии на ней фигуры и её статусе.

В данной модели достаточно определить какие направления для цвета являются ходом вперёд, а какие назад. Это существенно упрощает логику и время затрачиваемое на поиск следующей клетки. К тому же не тратится лишняя память на хранение клеток в игре не задействованных.

Модель легко адаптируется к любой размерности доски и не требует каких либо дополнительных изменений. Игра ведётся в понятиях направлений и границ.

Третий вариант в той или иной форме используется в программах реализующих искусственного противника, поскольку позволяет реализовать самый компактный вариант записи расстановки фигур на доске, что важно для построения дерева перебора ходов. Также позволяет более просто (в отличие от последовательного перебора) выделять из всех имеющихся на доске фигур подмножества доступных к ходу, к взятию противником и т.д.

Именно этот способ будет использован в проектируемом приложении.

Его идея заключается в использовании для описания доски так называемых «битбордов», когда доска представлена в виде битовой матрицы, каждый бит которой описывает одну из игровых клеток. При этом договоримся что порядковый номер бита числа соответствует порядковому номеру игровой клетки по образцу представленному на рисунке 1 (приложение 1).

Таким образом для перечисления всех игровых клеток понадобится одно двоичное число разрядностью 32 бита. Одна переменная такого типа сможет описывать состояние некоторого атрибута клетки в терминах да/нет (если *да* — бит установлен, иначе сброшен) для всей доски сразу.

Для описания текущей расстановки требуется описать факт наличия на клетке фигуры некоторого типа. Соответственно назовём атрибут Фигура и будем считать, что его состояние *да* говорит о том что клетка в данный момент не пуста.

Для хранения информации о существующих фигурах каждого из игроков понадобится две такие переменные. Но во время игры шашка может менять свой статус становясь дамкой. Для того чтобы иметь возможность отличить такие фигуры от рядовых введём ещё одну переменную точно такого же типа.

Дополнительно требуется знать кому из игроков принадлежит право хода. Эту информацию можно описать переменной с логическим значением да — белые, нет — чёрные.

Итого имеется четыре переменных. Пусть они условно зовутся *белые*, *чёрные* и *дамки*. Таким образом стандартное расположение фигур на доске в начале партии будет описываться соответствующей структурой показанной на рисунке 2 (приложение 1).

Или другой вариант сложившейся ситуации в конце партии где c3 — белая простая фигура, e7 — чёрная дамка, b6 — чёрная простая фигура опишется следующим набором, когда ход принадлежит чёрным. Его можно посмотреть на рисунке 3 (приложение 1).

Эту же идею можно выразить другими словами. Из множества клеток на доске мы выделили три подмножества B, W и S содержащие на себе соответствующие фигуры (обладающими свойством «содержит на себе ...» чёрную, белую фигуры или дамку). Каждое из них описывается двоичной переменной.

Теперь для того чтобы узнать местонахождение дамок заданного цвета надо найти пересечение двух множеств одно из которых содержит фигуры этого цвета, а второе — все дамки. Такие действия хорошо описываются логическими операциями над двоичными числами.

В терминах булевой алгебры операция будет выглядеть так:

ЧЁРНЫЕ.дамки = ЧЁРНЫЕ ∩ ДАМКИ

А в терминах логических операций над нашими двоичными переменными, так:

ЧЁРНЫЕ.дамки = ЧЁРНЫЕ И ДАМКИ

Соответственно, чтобы найти все простые фигуры заданного цвета необходимо выполнить то же действие, но вместо второго операнда использовать его дополнение:

белые.простые = Б ∩ $\overline{Д}_A$ = белые И (НЕ дамки), где А — это универсальное множество содержащее все 32 клетки доски.

Необходимо определиться с выходными данными. Все они поддаются записи в формате аналогичном входным. Это те же самые подмножества клеток доски, но соответствующие значениям другого атрибута. Первые

три из них относятся к состоянию доски в целом, а три вторых имеют смысл лишь в привязке к одной из доступных к перемещению фигур. Поэтому из модуля будет возвращаться структура содержащая первые три значения из перечисленных в начале раздела и некоторый массив содержащий три последних значения для каждой фигуры из доступных для хода. Примерно так:

– все фигуры доступные к ходу

– все фигуры способные взять фигуру противника

– все фигуры противника доступные для взятия

– массив ходов для каждой из доступных фигур:

– свободные клетки доступные для хода

– клетки при ходе на которые будет взята чужая фигура

– клетки при ходе на которые фигура будет повышена

– массив клеток с которых будет снята фигура противника (для каждой из клеток на которых произойдёт взятие)

На этом этапе уже возможно приступить к разработке алгоритмов поиска необходимых выходных данных. Будет рассматриваться лишь один вариант, когда право хода принадлежит игроку белыми и его фигуры расположены снизу доски. Все остальные варианты можно получить путём обмена местами Ч и Б и (если необходимо) разворотом доски на 180 градусов. Порядок действий при этом останется прежним.

Итак, вот что мы имется на старте вычислений:

A — универсальное множество содержащее номера всех чёрных клеток игровой доски.

$|A|=32$.

$ЧЁРНЫЕ \subset A$ – подмножество со свойством «содержат чёрные фигуры».

$БЕЛЫЕ \subset A$ – подмножество со свойством «содержат белые фигуры».

$ЧЁРНЫЕ \cap БЕЛЫЕ = \emptyset$ – на одной клетке не может стоять две фигуры одновременно.

$ДАМКИ \subset A$ – подмножество со свойством «содержат дамки».

Фигура в шашках может ходить в четырёх различных направления северо-восток (СВ), северо-запад (СЗ), юго-восток (ЮВ) и юго-запад (ЮЗ). По этой причине фигура может быть доступной для хода в одном из них, но в то же время не доступна для хода в другом. Аналогично и с потенциальными жертвами. Каждый из этих вариантов следует рассматривать отдельно и в качестве результирующего брать сумму четырёх множеств. Для упрощения представления этого будет введено понятие ориентированного множества клеток — это множество, у свойства которого уточнено направление. Например:

$ЖЕРТВЫ.СВ \subset ЖЕРТВЫ \subset ЧЁРНЫЕ \subset A$ — подмножество клеток со свойством «пустая клетка за фигурой» в направлении на северо-восток.

Какие критерии могут выделить потенциальную жертву среди остальных фигур одного цвета? Их три: она не находится на краю доски, не прикрыта в направлении хода фигурой своего цвета, не прикрыта фигурой противника. Случаи с ходами дамкой через несколько клеток пока не рассматриваются.

Исключить фигуры на краю доски легко. Понадобится определить вспомогательное подмножество $ПЕРИМЕТР \subset A$ со свойством «клетки на периметре» и вычесть его из ЧЁРНЫЕ:

$ЖЕРТВЫ = ЧЁРНЫЕ \setminus ПЕРИМЕТР$

Фигура прикрыта своими, если она присутствует в составе непрерывной цепочки фигур своего цвета находящихся на одной диагонали. Цепочка должна состоять не менее чем из двух фигур. Определим функции над множеством, которые возвращают следующие подмножества:

i.ВЗ(a) – возвращает подмножество клеток фигуры на которых входят в цепочки на диагоналях параллельных направлению с СЗ на ЮВ для соответствующих ходов.

i.ЗВ(a) – возвращает подмножество клеток фигуры на которых входят в цепочки на диагоналях параллельных направлению с СВ на ЮЗ

Круг потенциальных жертв сужается:

$$ЖЕРТВЫ.СЗ = ЖЕРТВЫ.ЮВ = ЖЕРТВЫ \setminus i.ЗВ(ЧЁРНЫЕ)$$

$$ЖЕРТВЫ.СВ = ЖЕРТВЫ.ЮЗ = ЖЕРТВЫ \setminus i.ВЗ(ЧЁРНЫЕ)$$

Фигуры прикрытые цветом противника найти несколько сложнее потому как два этих подмножества не пересекаются. Чтобы получить такую возможность будет определена дополнительная функция s(a), которая возвращает подмножество отражающее состояние клеток после переноса всех фигур в одном из четырёх возможных направлений хода на одну клетку. При этом информация о фигурах ушедших за периметр доски безвозвратно теряется.

Таким образом что:

БЕЛЫЕ = {0, 6, 31}; s.СЗ(БЕЛЫЕ) = {4, 11}

Теперь если сдвинуть белые фигуры в «противоход» рассчитываемому направлению то их пересечение с чёрными даст искомые фигуры прикрытые противником. С учётом полученного ранее:

$$ЖЕРТВЫ.СЗ = ЖЕРТВЫ.СЗ \cap s.ЮВ(БЕЛЫЕ)$$

$$ЖЕРТВЫ.СВ = ЖЕРТВЫ.СВ \cap s.ЮЗ(БЕЛЫЕ)$$

$$ЖЕРТВЫ.ЮЗ = ЖЕРТВЫ.ЮЗ \cap s.СВ(БЕЛЫЕ)$$

$$ЖЕРТВЫ.СВ = ЖЕРТВЫ.СВ \cap s.ЮЗ(БЕЛЫЕ)$$

Теперь имеется четыре подмножества клеток с фигурами противника которые потенциально подвержены атаке.

Выделим $ДОСТУПНЫЕ \subset БЕЛЫЕ$ – ориентированные подмножества клеток имеющих свойство «содержат фигуры доступные для хода». Критериев отбора два, неважно ведётся поиск дамки или простой фигуры, – это отсутствие на следующей клетке фигуры или наличие Жертвы.

Определим $ПУСТЫЕ = A \setminus (БЕЛЫЕ \cup ЧЁРНЫЕ)$ – подмножество имеющее свойство «не содержит фигуру». Тогда:

$$ДОСТУПНЫЕ.СЗ = (БЕЛЫЕ \cap s.ЮВ(ЖЕРТВЫ)) \cup (БЕЛЫЕ \cap s.ЮВ(ПУСТЫЕ))$$

$$ДОСТУПНЫЕ.СВ = (БЕЛЫЕ \cap s.ЮЗ(ЖЕРТВЫ)) \cup (БЕЛЫЕ \cap s.ЮЗ(ПУСТЫЕ))$$

$$ДОСТУПНЫЕ.ЮЗ = (БЕЛЫЕ \cap s.СВ(ЖЕРТВЫ)) \cup (БЕЛЫЕ \cap s.СВ(ПУСТЫЕ))$$

$$ДОСТУПНЫЕ.ЮВ = (БЕЛЫЕ \cap s.СЗ(ЖЕРТВЫ)) \cup (БЕЛЫЕ \cap s.СЗ(ПУСТЫЕ))$$

Дамка и простая фигура хоть и имеют схожую механику хода, но отличаются числом возможных его итераций. Тот факт что простая фигура может ходить и рубить лишь на одну клетку вперёд (назад только рубить) позволяет обработать их поведение в рамках операций с одним подмножеством. Дамки же в силу своих свойств придётся обрабатывать индивидуально. К счастью число дамок возникающих на игровом поле, как правило, невелико, а образуются они ближе к концу игры, когда ряды обоих противников достаточно разряжены, чтобы фигура могла добраться до противоположного края доски избежав встречи с противником.

Таким образом имеются веские причины, чтобы на этом этапе разделить обработку простых фигур и дамок.

Сначала ищутся Жертвы простых фигур:

$$ЖЕРТВЫ.О.СЗ = ЖЕРТВЫ.СЗ \cap s.СЗ(ДОСТУПНЫЕ.СЗ \setminus ДАМКИ)$$

$$ЖЕРТВЫ.О.СВ = ЖЕРТВЫ.СВ \cap s.СВ(ДОСТУПНЫЕ.СВ \setminus ДАМКИ)$$

$$ЖЕРТВЫ.О.ЮЗ = ЖЕРТВЫ.ЮЗ \cap s.ЮЗ(ДОСТУПНЫЕ.ЮЗ \setminus ДАМКИ)$$

$$ЖЕРТВЫ.О.ЮВ = ЖЕРТВЫ.ЮВ \cap s.ЮВ(ДОСТУПНЫЕ.ЮВ \setminus ДАМКИ)$$

Затем Охотники — фигуры которые могут совершить взятие противника. Критерием поиска будет тот факт, что на следующей клетке в направлении хода должна находиться потенциальная Жертва. Из полученных данных можно сразу найти все простые фигуры Охотников. Достаточно сдвинуть найденные Жертвы в обратном направлении:

$$ОХОТНИКИ.О.СЗ = s.ЮВ(ЖЕРТВЫ.СЗ)$$

$$ОХОТНИКИ.О.СВ = s.ЮЗ(ЖЕРТВЫ.СВ)$$

$$ОХОТНИКИ.О.ЮВ = s.СЗ(ЖЕРТВЫ.ЮВ)$$

$$ОХОТНИКИ.О.ЮЗ = s.СВ(ЖЕРТВЫ.ЮЗ)$$

Также необходимо исключить из числа Доступных простые фигуры не имеющие Жертв позади себя:

$$ДОСТУПНЫЕ.ЮВ = (ДОСТУПНЫЕ.ЮВ \cap ОХОТНИКИ.О.ЮВ) \cup (ДОСТУПНЫЕ.ЮВ \cap ДАМКИ)$$

$$ДОСТУПНЫЕ.ЮЗ = (ДОСТУПНЫЕ.ЮЗ \cap ОХОТНИКИ.О.ЮЗ) \cup (ДОСТУПНЫЕ.ЮЗ \cap ДАМКИ)$$

Можно начинать суммировать данные о Доступных, Охотниках и Жертвах:

$$ЖЕРТВЫ = ЖЕРТВЫ.О.СВ \cup ЖЕРТВЫ.О.СЗ \cup ЖЕРТВЫ.О.ЮВ \cup ЖЕРТВЫ.О.ЮЗ$$

$$ОХОТНИКИ = ОХОТНИКИ.О.СВ \cup ОХОТНИКИ.О.СЗ \cup ОХОТНИКИ.О.ЮВ \cup ОХОТНИКИ.О.ЮЗ$$

$$ДОСТУПНЫЕ = ДОСТУПНЫЕ.СВ \cup ДОСТУПНЫЕ.СЗ \cup ДОСТУПНЫЕ.ЮВ \cup ДОСТУПНЫЕ.ЮЗ$$

На этом этапе становится возможным заполнить массивы ходов для простых фигур.

Для каждой из списка доступных простых надо будет найти следующие подмножества клеток:

– на которых она окажется после хода

– с фигурой которая будет взята по результату хода

– на которых она будет повышена до дамки

Опишем алгоритм на примере направления СВ, для остальных он останется тем же. Для этого придётся определить очередные подмножества:

ГРАНИЦА – включающее в себя клетки лежащие на последнем верхнем ряду доски. Это те клетки, попав на одну из которых белая шашка становится дамкой.

$ФИНИШ = \emptyset$ – множество в котором содержится клетка в которую может перейти текущая фигура. Перед поиском в каждом из направлений содержащийся в нём элемент отбрасывается.

$ЖЕРТВА = \emptyset$ – множество из одно элемента в котором содержится клетка с жертвой, если таковая существует в рамках данного хода. Обнуляется аналогично предыдущему.

$ХОД.ФИНИШ = \emptyset$ – множество в котором будут накапливаться клетки на которые можно переставить текущую фигуру. Сохраняет свое состояние между поиском в разных направлениях.

$ХОД.ВЗЯТИЕ = \emptyset$ – аналогично предыдущему, с тем отличием, что накапливаются клетки на которые перейдёт фигура совершив взятие противника ($ХОД.ВЗЯТИЕ \in ХОД.ФИНИШ$).

$ХОД.ЖЕРТВА = \varnothing$ – множество в котором будут накапливаться клетки с жертвами текущей фигуры. Хранит состояние аналогично предыдущему.

$ХОД.ДАМКА = \varnothing$ – множество в котором будут накапливаться клетки ход на которые повысит шашку. Сохраняет состояние аналогично предыдущему.

$ВЗЯТИЯ = \varnothing$ – множество хранящее кортежи $\{ ЖЕРТВА, ФИНИШ \}$. Так же сохраняет своё состояние.

Порядок действий описанный псевдокодом можно посмотреть в листинге 1 (приложение 1).

Осталось получить сходный набор данных по доступным к ходу дамкам. Здесь ситуация сложнее. Дамка ходит на всю длину диагонали как вперёд так и назад. Она не обязана останавливаться на следующей за клеткой Жертвой, путь к ней может оказаться перекрыт дружественной фигурой или прикрытой фигурой противника. Цикл поиска усложнится за счёт дополнительных проверок и итераций. Будет определено дополнительное множество:

$ВСЕ = БЕЛЫЕ \cup ЧЁРНЫЕ$ – все находящиеся на доске фигуры.

Множества содержащие информацию о ходах, которые были описаны ранее будут использованы и здесь (кроме $ГРАНИЦА$), их поведение будет аналогичным.

Псевдокод поиска возможных ходов для Дамок (направление СВ) можно найти в листинге 2 (приложение 1).

Основные алгоритмы и структуры данных Решателя разработаны. В дальнейшем потребуется изложить их на выбранном языке программирования, модифицировав с учётом предъявляемых требований и ограничений.

2.2.2 Модуль Доска

Модуль реализует игровой реквизит и процесс принятия решений по перемещению и снятию фигур. В своей работе тесно взаимодействует с Решателем, обмениваясь с ним общими типами данных.

Доска включает в себя множество автономных экземпляров объектов определённых типов способных обмениваться между собой сообщениями. Часть их статические – те которые не принимают непосредственного участия в процессе игры – это клетки координатные метки отображаемые по периметру доски и белые клетки по которым фигуры перемещаться не могут. Ещё один тип — чёрные клетки на которых и происходит всё действие. Решатель частью Доски не является.

Доска хранит своё состояние в структуре вида:

ПАРТИЯ – сквозной идентификатор текущей партии.

РАССТАНОВКА $\{БЕЛЫЕ, ЧЁРНЫЕ, ДАМКИ\}$ структура описывающая фигуры на доске .

ИГРОК – информация об игроке имеющем право хода.

НОМЕР – порядковый номер хода в партии.

НОТАЦИЯ – запись хода в игровой нотации.

ВЗЯТИЕ – флаг подтверждающий факт взятие фигуры противника в текущем ходе.

Доска оперирует множествами и структурами описанными ранее в 2.3.1. Экземпляры Клеток её составляющие способны принять или отвергнуть попытку поставить на себя фигуру. Сообщить Доске с какой

клетки игрок пытается поднять фигуру, по запросу отобразить или скрыть фигуру указанного цвета и статуса. Доска может пометить любую из Клеток как доступную или недоступную для поднятия/опускания фигуры.

После того как все фигуры в начале партии расставлены и определён игрок который будет ходить первым, эта информация передаётся Решателю, который возвращает в ответ информацию о возможных ходах.

В зависимости от того существует ли в текущей расстановке возможность срубить фигуру противника, Доска помечает необходимые Клетки как разрешённые к поднятию с них фигур и ожидает действий игрока. Если Клетка согласна поднять стоящую на ней фигуру то она сообщает об этом Доске передавая присвоенный ей номер.

На основе полученного номера и ситуации на поле Доска отбирает доступные для хода клетки, помечает их таковыми и снова ожидает действий игрока. Если Клетка на которую Игрок пытается поставить фигуру не согласна её принять она сигнализирует об этом Доске, которая очищает пометки доступности и ожидает попытку поднятия другой фигуры.

Если же Клетка на которую Игрок пытается поставить фигуру согласна её принять, то она также передаёт свой номер Доске в соответствующем сообщении.

На основе полученного номера Доска выясняет было ли совершено взятие фигуры противника и какой именно (в состав какого из кортежей $ЖЕРТВА = \{ЖЕРТВА, ФИНИШ\}$ он входит как $ФИНИШ$). Выполняется коррекция РАССТАНОВКА (клетка исключается из соответствующих подмножеств).

Далее, фигура переносится с начальной клетки на конечную, для чего

выполняется коррекция РАССТАНОВКА (клетка исключается из одних и включается в другие подмножества).

Если в результате хода фигура должна стать дамкой, то выполняется коррекция РАССТАНОВКА (клетка включается в *ДАМКИ*).

Любые метки доступности у Клеток снимаются и игровое поле перерисовывается с учётом новых значений РАССТАНОВКА. Эти новые значения передаются Решателю, при этом цвет игрока не меняется. Если в возвращённых Решателем данных $ЖЕРТВЫ \neq \varnothing$ весь цикл повторяется.

Если в возвращённых Решателем данных $ЖЕРТВЫ = \varnothing$ – ему снова передаётся РАССТАНОВКА, но право хода передаётся второму игроку..

Если в возвращённых Решателем данных $ДОСТУПНЫЕ = \varnothing$ – сообщается об окончании партии в которой текущий игрок считается проигравшим, так как не имеет открытых к ходу фигур, либо все его фигуры сняты с доски противником. В противном случае происходит возврат к началу цикла.

2.3.3 Модуль Архив

Модуль реализует интерфейс к внешней реляционной базе данных хранящей записи ходов всех начатых когда либо партий и информацию о всех их участниках. Её инфологическая модель показана на Рисунке 4 Приложения 1.

Модуль позволяет обновить атрибуты экземпляра сущности Партия или добавить её новый экземпляр, добавлять экземпляры сущностей Игрок и Ход. Он не производит удаление экземпляров каких либо сущностей.

По запросу модуль должен возвращать:

- массив имён игроков

-массив объектов содержащий всех противников с которыми когда либо сыграл указанный игрок

- массив объектов содержащий все партии между указанными игроками

- массив объектов содержащий все ходы в пределах указанной партии

2.2.4 Модули Список и Фильтр

Список представляет собой простой элемент интерфейса типа список. По команде добавляет в последнюю позицию содержимое полей номер_хода и Нотация из структуры описывающей состояние Доски. В режиме просмотра записи игры возвращает номер_хода соответствующий выбранному элементу списка

Фильтр состоит из двух выпадающих списков позволяющих выбрать пару игрок/оппонент для которых будет отобран список хранящихся записей игр.

В начальном состоянии первый список заполнен именами игроков, второй пуст. При выборе элемента у архива запрашивается перечень игроков, которые являлись противниками этого игрока хотя бы в одной из партий. Этими значениями заполняется список номер два. В качестве первого элемента добавляется символ «*» означающий «любой из» и позволяющий показать все игры в которых принимал участие игрок выбранный в первом списке. Выбор же имени противника из второго списка позволяет ограничить список игр двумя участниками.

Когда участники выбраны делается ещё один запрос к архиву на

получение перечня игр между указанными участниками. Значениями его результата заполняется таблица состоящая из колонок содержащих дату начала и завершения партии (пустое значение в случае незавершённой), имена игроков за белые и чёрные фигуры.

По двойному клику на заполненной строке таблицы происходит запрос к Архиву на получение перечня записанных ходов в рамках выбранной партии, после чего Список заполняется номерами ходов и их нотацией упорядоченно в последовательности их совершения. Доске передаётся для отображения записанное состояние перед первым совершенным ходом.

При выборе следующего/предыдущего или произвольного хода соответствующая ему расстановка так же отправляется доске для отображения.

В случае, если партия не завершена, вместо даты окончания в таблице будет отображена кнопка Доиграть по нажатию на которую в Списке будет выбран текущим последний из ходов, восстановлено соответствующее после его совершения состояние доски, а сама доска и интерфейс приложения переведены в режим игры.

3 Реализация проекта в программной среде

3.1 База данных и класс Xeno

Для реализации БД в которой будут храниться данные об участниках и сыгранных ими партиях выбрана встраиваемая СУБД SQite. Выбор обусловлен простотой её использования (она выполнена в виде драйвера и имеет версии под все основные операционные системы), отсутствием необходимости в установке и настройке отдельного сервера, низким потреблением ресурсов.

Сначала потребуется преобразовать ранее описанную модель Сущность-Связь (см. рисунок 4) в даталогическую, на основе которой будут разрабатываться реляционные таблицы под выбранную СУБД. Такая модель показана на рисунке 5 (приложение 1).

Даталогическая модель перенесётся в среду SQLite практически без изменений, надо будет лишь уточнить типы и размеры полей таблиц:

ключ	**имя**	**тип**	**размер**	**авто-т**	**не null**	**уник-ть**
PK	playerID	integer		да		
	name	string	21		да	да

Таблица 1. Отношение Игроки

Описание таблицы на языке DDL (его диалекте используемом в SQLite):

```
CREATE TABLE 'Players' (
'_playerID' 'INTEGER' PRIMARY KEY AUTOINCREMENT,
'name' 'STRING' NOT NULL ON CONFLICT FAIL UNIQUE ON
```

CONFLICT FAIL

);

ключ	имя	тип	размер	авто-т	не null	уник-ть
PK	partyID	integer		да		
FK	playerIDw	integer			да	
FK	playerIDb	integer			да	
	dateStart	datetime			да	
	dateFinish	datetime				

Таблица 2. Отношение Партии

Дополнительно в этой таблице необходимо реализовать проверку на совпадение полей playerIDw и playerIDb потому как игрок не может находиться по две стороны доски одновременно:

CREATE TABLE 'Parties' (

'_partyID' 'INTEGER' PRIMARY KEY AUTOINCREMENT,

'playerIDb' 'INTEGER' NOT NULL REFERENCES 'Players' ('_playerID'),

'playerIDw' 'INTEGER' NOT NULL REFERENCES 'Players' ('_playerID'),

'dateStart' 'DATETIME' NOT NULL,

'dateFinish' 'DATETIME',

CHECK ('playerIDw' <> 'playerIDb')

);

ключ		имя	тип	размер	авто-т	не null	уник-ть
PK	FK	partyID	integer			да	
		ordinal	integer			да	
		notation	string			да	
		player	bool			да	
		snap	blob			да	

Таблица 3. Отношение Ходы

В этой таблице будет составной первичный ключ, так как не может быть партии с двумя одинаковыми порядковыми номерами хода. В то же время часть его будет являться внешним ключом указывающим на элемент таблицы Партии.

Поле BLOB хранит произвольное значение в двоичном виде, в него будет записываться набор из трёх чисел длиной в 32 бита каждое. Та самая РАССТАНОВКА из 2.3.2.

```
CREATE TABLE 'Moves' (
'partyID' 'INT' REFERENCES 'Parties' ('_partyID')
NOT NULL,
'ordinal' 'INT' NOT NULL,
'snap' 'BLOB' NOT NULL,
'notation' 'STRING' NOT NULL,
'player' 'BOOLEAN' NOT NULL,
'strike' 'BOOLEAN' NOT NULL,
'delay' 'INT',
'comment' 'STRING',
PRIMARY KEY (
'partyID', 'ordinal'
```

) ON CONFLICT FAIL

);

Для того чтобы обеспечить ранее описанное взаимодействие с БД потребуется несколько запросов на чтение и запись данных.

– запрос на добавление нового игрока:

INSERT INTO `Players` (`name`) *param1*;

– запрос на добавление новой партии:

INSERT INTO `Parties` (`playerIDb`, `playerIDw`, `dateStart`)

param1, *param2*, *param3*;

– запрос на обновление информации о партии (в случае если её бросили и доиграли позже):

UPDATE `Parties` SET `dateFinish` = value1 "WHERE `_partyID` = value2;

– запрос на добавление нового хода:

INSERT INTO `Moves`

(`partyID`, `ordinal`, `snap`, `notation`, `player`)

param1, *param2*, *param3*, *param4*, *param5*;

– запрос на получение списка всех зарегистрированных игроков:

SELECT _playerID, name FROM Players

– запрос на получение всех оппонентов указанного игрока:

SELECT `_playerID`, `name` FROM (

SELECT `playerIDw` AS _id

FROM `Parties` WHERE `playerIDb` = *param1*

UNION

SELECT `playerIDb` AS _id

FROM `Parties` WHERE `playerIDw` = *param1*

GROUP BY `_id`

)

INNER JOIN `Players` ON `Players`.`_playerID` = `_id`;

– запрос на получение списка партий сыгранных между двумя указанными игроками:

SELECT _partyID, playerIDb, playerIDw, dateStart, dateFinish

FROM `Parties`

WHERE

(playerIDb = :id1 OR playerIDw = *param1*)

AND

(playerIDb = :id2 OR playerIDw = *param2*)

и его модификация для получения списка всех партий сыгранных одним указанным игроком:

SELECT _partyID, playerIDb, playerIDw, dateStart, dateFinish

FROM Parties

WHERE

(playerIDb = :id1 OR `playerIDw` = *param1*)"

– запрос на получение всех ходов совершённых в рамках указанной партии:

SELECT partyID, ordinal, snap, notation, player

FROM Moves

WHERE partyID = *param1*

ORDER BY ordinal;

Потребуется узнавать значение первичного ключа только что вставленной в таблицу записи. Его можно получить используя встроенную функцию SQLite:

SELECT last_insert_rowid();

Все запросы к БД будут вызываться из программного кода класса реализующего поведение модуля Архив.

Эта реализация расположена в классе Xeno, который хранит объект подключения к БД, методы его открытия и закрытия, а так же методы для извлечения и записи в него данных. Содержит описание структур данных в которые отображаются атрибуты таблиц SQLite необходимые для его работы или обмена с остальными классами приложения.

В качестве языка программирования использован C++ – разработанный ещё в 1969-1973 гг. Деннисом Ритчи и ьёрном Страуструпом, сотрудниками Bell Labs, но до сих пор держащем первые позиции за счёт своей мощности и гибкости. В качестве сторонних библиотек используемых для отображения элементов интерфейса выбран QT – кроссплатформенный фреймворк для разработки на C++, разработанный в 1996 г. в компании Trolltech.

Исходный код класса можно найти в приложении 2. Это относится и ко всем остальным рассматриваемым компонентам приложения.

Есть момент в Xeno, который заслуживает краткого пояснения – то как происходит отображение структуры описывающей расстановку фигур на доске используемой в приложении в тип BLOB поля данных СУБД SQLite. Потребуется способы сериализации экземпляра структуры Xeno::Move::Snap и её последующего восстановления из поля БД.

```
struct Snap {
uint32_t B = 0b0;
uint32_t W = 0b0;
uint32_t S = 0b0;
} snap;
```

Для этого перегружаем этому типу данных операторы ввода и вывода:

```
// сериализация Xeno::Move::Snap ( перегружаем оператор <<)
QDataStream &operator <<(QDataStream &in, Xeno::Move::Snap const &x)
{return in << x.B << x.W << x.S;}
// десериализация Xeno::Move::Snap ( перегружаем оператор >>)
QDataStream &operator >>(QDataStream &out, Xeno::Move::Snap &x)
{return out >> x.B >> x.W >> x.S;}
```

Оператор << упаковывает участки памяти где хранятся переменные в один непрерывный массив байтов и помещают его в переменную типа QDataStream. Переменные располагаются в указанном порядке. Оператор » выполняет обратную операцию разворачивая содержимое пакета в области памяти хранящие структуру, восстанавливая переменные. Разумеется типы упаковываемого и восстанавливаемого объектов обязаны совпадать.

В примере ниже объект snap типа Xeno::Move::Snap будет упакована и помещена в массив arr типа QByteArray, который можно записать на носитель или в поле БД:

```
QByteArray arr;
QDataStream stream(&arr, QIODevice::WriteOnly);
stream << snap;
```

Пример обратной операции извлечения объекта snap из хранилища:

```
QByteArray arr;
arr = query.value(2).toByteArray();
QDataStream stream(&arr, QIODevice::ReadOnly);
stream >> move.snap;
```

Здесь пакет извлекается из поля запроса к БД и восстанавливается в snap, объекте типа Xeno::Move::Snap.

3.2 Класс Solver (реализация модуля Решатель)

Рассматриваемый класс реализует алгоритмы и наборы данных ранее разработанные для модуля Решатель. В своей работе частично опирается на типы данных предоставляемые Xeno. Например Xeno::Move::Snap используется как аргумент метода Solver::SetMoves(const Xeno::Move::Snap & snap, Color color).

Все подмножества клеток доски описанные в 2.3.1 представлены в виде 32 разрядных беззнаковых целых в которых порядковый номер установленного бита указывает на номер клетки входящей в подмножество с заданными свойствами. Примеры:

– основные переменные

```
uint32_t player;
uint32_t enemy;
uint32_t super;
uint32_t enemyPerimeter;
uint32_t enemyHoles;
```

uint32_t playerHoles;

uint32_t all;

uint32_t empty;

– структуры для поиска в разных направлениях

struct Sieve::Route {

uint32_t interval;

uint32_t unlocked;

uint32_t hunterOrdinary;

uint32_t prey;

uint32_t preyOrdinary;

} routeFL, routeFR, routeBL, routeBR;

– структура накапливающая информацию о возможных ходах фигур

struct Sieve::Moves {

struct Drops {

uint32_t available = 0b0;

uint32_t strike = 0b0;

uint32_t super = 0b0;

QMap <uint32_t, uint32_t> strikes;

};

uint32_t unlocked;

uint32_t hunter;

uint32_t prey;

QMap <uint32_t, Drops> drops;

} moves;

Moves содержит вложенные контейнеры (Drops) типа словарь хранящие <ключ, значение>, в котором ключом является битовая маска для каждой единственной клетки из Moves::unlocked. Значение же содержит общую информацию о клетках которые могут её принять, с которых может быть снята фигура противника и клетки ход на которые может повысить её до дамки. Кроме того значение содержит вложенный аналогичный контейнер strikes, где ключом является номер клетки приземление на которой вызовет взятие противника, а значением клетка с которой эта самая фигура будет снята.

Операции пересечения, объединения, разности и другие используемые в булевой алгебре заменяются наборами логических операций над двоичными числами. Например:

```
uint32_t all = player | enemy;
uint32_t empty = ~all;
```

Определённое ранее действие над множествами s.(A) преобразится в функции использующие в работе кроме вышеперечисленного ещё и операции двоичного сдвига. Рассмотрим на примере операции сдвига в направлении на северо-восток.

Строки и столбцы на доске пронумерованы справа-налево и снизу-вверх, как показано на рисунке 1 (приложение 1). Теперь, если внимательно изучить доску с нанесёнными на неё номерами клеток становятся видны некоторые закономерности в механике их сдвигов. Например, сдвиг в указанном направлении клеток находящихся на чётных строках приводит к изменению их номера на +3, а находящихся на нечётных – на +4. Или, если отталкиваться от столбцов, наоборот – нечётный столбец +3, а чётный +4. Так, клетка с позицией 1 перейдёт в

позицию 1+4=5, а клетка из позиции 6 перейдёт в 6+3=9. Этого результата можно легко достичь используя операции сдвига в которых двоичное число сдвигается в нужную сторону на необходимое число битов. При одиночном сдвиге влево бит в позиции 0 перемещается в позицию 1, из 1 в 2 и так далее. При этом в младший бит записывается новое значение 0, а старое значение старшего бита теряется.

Разница в количестве сдвигов для чётных и нечётных столбцов не позволяет выполнить операцию за один шаг. Поэтому придётся сдвигать их поочерёдно, затем объединяя полученные результаты. Но сначала их надо как-то разделить для чего будут использоваться заранее рассчитанные битовые маски, где к примеру биты в позициях соответствующих чётным столбцам будут установлены в 1, в то время как остальные будут равны 0. В результате сложения позиций snap.W с такой маской мы сможем получить те позиции белых фигур, которые расположены на чётных столбцах игрового поля.

```
uint32_t maskOdd   = 0b11110000111100001111000011110000;
uint32_t maskEven  = ~maskOdd;
```

Кроме раздельного сдвига столбцов присутствует ещё один неприятный момент в виде фигур, которые расположены по периметру доски. Не всегда при сдвиге они перейдут в нужные позиции. Если сдвинуть те фигуры, которые расположены на крайнем столбце в направлении движения, то они то они словно телепортируются на другую сторону доски, чего в нормальной ситуации происходить не должно. Потому предварительно их надо отсечь с помощью соответствующей маски:

```
uint32_t trimBorderRight = 0b11101111111011111110111111101111;
```

uint32_t maskEvenNE = maskEven & trimBorderRight;

В результате рассматриваемая операция сдвига будет выглядеть так:

uint32_t shiftedSnapW = ((snap.W & maskEvenNE)<<4) |

((snap.W & maskOdd)<<3);

Необходимо реализовать второе действие i(A), которое возвращает клетки с непрерывными цепочками фигур в заданном подмножестве.

Рассмотрим вариант с нахождением установленных битов двоичного числа расположенных подряд и количеством более одного. Пусть:

uint32_t arr = 1b00001000010111100100110000111001;

Можно заметить, некоторое правило которому подчиняются искомые позиции. При сдвиге числа в любую из сторон бит находящийся в составе цепочки перейдёт в такую позицию которая содержит установленный бит в исходном числе.

uint32_t arr_r = 0b000001000010111100100110000111001; //вправо

uint32_t arr = 0b000010000101111001001100001110001; //исходное

uint32_t arr_l = 0b000100001011110010011000011100010; //влево

uint32_t arr_r = 0b00000000000011110000011000011100000; //результат

Это можно представить так:

$$|A|=32 \quad , \quad arr \subset A \quad , \quad сдвиг.ПРАВО(arr) \subset arr \quad , \quad сдвиг.ВЛЕВО(arr) \subset arr$$

$$\forall a : a \in ЦЕПОЧКА, a \neq \varnothing \rightarrow a \in (сдвиг.ПРАВО(arr) \cap arr)$$

Таким образом:

$$ЦЕПОЧКА = сдвиг.ВПРАВО(arr) \cup сдвиг.ВЛЕВО(arr)$$

uint32_t chain = (arr << 1) | (arr >> 1)

Этот принцип применим и к расстановкам, разве что сдвигать числа

надо будет с помощью функций типа той что описана выше. Дополнительно понадобится добавить к результату клетки периметра, содержащие вражеские фигуры. Для этого потребуется соответствующая маска.

```
uint32_t perimeter = 0b11111000000110000001100000011111;
perimeterBlack = snap.B & perimeter;
routeNE.chain = routeSW.chain = (snap.B & enemyNE(snap.B)) |
                                (snap.B & enemySW(snap.B)) |
                                perimeterBlack;
```

Поиск Незапертых и потенциальных Жертв происходит по уже рассмотренным принципам, порой для упрощения расчётов вводятся некоторые предварительно вычисленные вспомогательные переменные. Например «дырки». Под ними понимается инвертированное двоичное число расстановки, которое содержит клетки имеющие противоположное состояние по отношению к исходному.

```
uint32_t holesPlayer = ~snap.W; //клетки без белых фигур
uint32_t routeNE.prey = snap.B & routeNE.chain &
                        playerSW(holesPlayer);
```

Результатом будут все потенциальные Жертвы для направления Северо-Восток. Аналогично для Доступных в том же направлении:

```
uint32_t empty        =  ~all; //все клетки без фигур
routeNE.unlocked  =  (snap.W & playerSW(empty)) |
                        (snap.W & enemyNE(routeNE.prey));
```

Методы реализующие поиск возможных ходов и Жертв для Доступных практически полностью повторяют алгоритмы из 2.3.1 и не

нуждаются в детальном рассмотрении.

3.3 Класс Board (реализация модуля Доска)

Класс реализующий модуль Доска. Основные его составляющие:

Board::Style _style; //описывает размеры и оформление элементов доски

Xeno::Move _move;

Solver* _solver;//указатель на связанный экземпляр

Solver::Sieve::Moves* _moves;//указатель на связанный экземпляр

Так же включает в себя два вложенных класса Cell и Label отвечающих за отображение реквизита игры и взаимодействие с ним пользователя. Такое решение выбрано ради упрощения доступа к закрытым членам объемлющего класса - это информация об оформлении элементов доски, информации о расстановке, режиме работы доски. Потому как члены – экземпляры вложенного класса имеют полный доступ к приватным атрибутам и методам объемлющего класса. Дополнительно это позволяет скрыть детали реализации вспомогательных подсистем от внешних по отношению к Board объектов выставляя наружу минимально необходимый интерфейс.

Label отвечает за отображение координатных меток вокруг игрового поля, а Cell – за отображение клеток и их логику поведения. Кроме прямых обращений к методам Board они связаны с ним посредством системы сообщений сигнал-слот. Суть её в том что сигнал вырабатывается в ответ на какое либо событие или он может быт испущен вручную. Слот является обработчиком сигнала (события) и может быть вызван как в ответ на это

самое событие так и вручную словно обычный метод объекта. Слоты и сигналы связаны между собой – это называется подпиской. Например объект A подписался на один из сигналов объекта B. Это означает что он указал себя одним из получателей информации об этом сигнале и указал свой внутренний метод, который будет вызван при его получении. Сигналы могут нести с собой некоторую информацию, которую слот может обработать или отбросить, на своё усмотрение. При этом сигнатуры сигнала и слота должны совпадать.

В качестве примера можно привести такой пример. Окно приложения имеет кнопку закрытия, которая содержит сигнал испускаемый при клике на неё. Приложение зная о существовании объекта-кнопки подписывается на этот сигнал и при нажатии на кнопку вызывается метод указанный в качестве слота, который отвечает за корректное завершение приложения. Но источниками и получателями вовсе не обязательно должны быть элементы графического интерфейса они доступны в любом классе-наследнике QObject. Получателями сигнала могут быть более одного объекта.

Данный механизм лежит в самой основе Qt и практически все его подсистемы, не только VCL, но и сетевая библиотека и даже STL, используют его возможности.

Доска использует механизм сигналов в трёх случаях – если необходимо передать набор данных объекту вне собственной области видимости об интерфейс которого ему не известно или в случае необходимости передать данные набору объектов одного типа с целью вызвать у них схожую реакцию.

Класс Board содержит сигналы:

– о том что необходимо перерисовать элементы доски при смене оформления или размера:

void Restyle();

- о том что необходимо перерисовать расстановку фигур на доске:

void ShowSnap();

- отправка информации о клетках способных принять фигуру:

void SetDrops(uint32_t const &drops);

- отправка информации о текущем ходе:

void SendMove(Xeno::Move _move);

- сообщение о завершении партии:

void SendGO();

Подписанты этих сигналов:

– Restyle()

Cell::Restyle и Label::Restyle – перерисовывают себя и фигуры в соответствии со значениями экземпляра _style.

– ShowSnap()

Cell::ShowCheck – на основании информации в _style и _move отображает на себе соответствующую фигуру.

– SetDrops(uint32_t const &drops)

Cell::SetDrop – если бит с номером клетки в drops установлен в 1 она помечается как разрешённая к принятию фигуры.

– SetPicks(uint32_t const &drops)

Cell::SetPick – если бит с номером клетки в drops установлен в 1 она помечается как разрешённая к поднятию с неё фигуры.

– SendMove(Xeno::Move _move)

UI::CatchMove(Xeno::Move move) – слот принадлежащий классу главного окна приложения, отправляет данные о совершённом ходе Xeno для записи его в базу данных и экземпляру QListWidget для отображения записи хода в игровой нотации.

Класс Board содержит слот Board::CatchDrop() связанный с сигналом Cell::SendDrop(), который служит для оповещения о том, что поднятая фигура была перемещена в новую позицию. Метод слота осуществляет внесение изменений в _move и принимает решение о передаче хода.

Для генерации записи хода в игровой нотации используется вспомогательная функция:

Qstring _Pos2Note(uint32_t position);

Она выполняет преобразование номера клетки в координаты алфавитно-цифровых меток доски.

Изображения фигур хранятся в виде графических файлов формата .png в файле ресурсов.

3.4 Класс DialogAdd (диалог добавления игрока)

Реализует модальный диалог добавления нового пользователя в базу данных. Контролирует длину вводимого имени в символах. Непосредственно вставка выполняется классом UI.

3.5 Класс Participant (вспомогательный элемент интерфейса)

Реализует выпадающий список игроков содержащий дополнительный элемент – кнопку добавления игрока и обрабатывает все связанные с этим потребности. Например сброс значения, если в двух связанных экземплярах этого класса выбрано одно и то же значение. Пропускание необходимых сигналов от приватных виджетов наружу, заполнение списков, скрытие части элементов, добавление элементов. Организация хранения и выдачи ID текущего игрока по запросу.

3.6 Класс UI (интерфейс приложения)

Реализует модуль Окно приложения и ответственен за координацию работы всех классов описанных ранее. Кроме них содержит экземпляр QListWidget – списка для отображения списка ходов и реализует модуль Фильтра. Управляет скрытием/отображением и доступностью элементов интерфейса в зависимости от текущего режима. О смене режима компоненты оповещаются посредством механизма сигнал-слот.

Отображает всплывающие сообщения в случае попытки прервать неоконченную партию или победы одного из участников. При добавлении нового игрока сигнализирует в случае попытки внести имя уже присутствующее среди участников.

Заключение

Проникновение современных информационных технологий во всё большее число аспектов жизнедеятельности человека оказывает ни с чем не сравнимое влияние на процессы умственного и психического развития человека. Наиболее этому подвержены дети, которые уже с раннего возраста начинают использовать различные мобильные устройства в качестве окна в бесконечно разнообразный мир досуга, предоставляемый глобальной информационной системой по имени Интернет. Его яркость и многообразие может провоцировать замещение собой реальных собеседников, участников по играм. Вырабатывается склонность к частой и резкой смене объекта приложения внимания. В первую очередь перенимаются правила и поведенческие стереотипы главенствующие в сети. Результатом такого длительного, оторванного от живого общения с друзьями и близкими времяпровождения становится личность, с плохо развитой способностью к планомерному и вдумчивому изучению объектов и процессов окружающего мира, к планированию и оценке последствий своих решений.

Очень важную роль в предотвращении подобного сценария развития событий имеет фактор как можно большего присутствия родителей или любого взрослого в повседневном времяпрепровождении ребёнка. Того кто сможет через правильные образцы поведения и мышления привить и развить те положительные качества, которые действительно понадобятся ребёнку, во взрослой жизни, но которых к сожалению не может дать всё многообразие досуга предлагаемого глобальной информационной сетью.

Целью данной работы было поставлено решение обрисованной проблемы путём создания простого приложения для персонального компьютера, которое позволит хотя бы частично привлечь родителя к участию в досуге ребёнка, сделать его действительно полезным и обучающим. Для этого в качестве основы для приложения взята настольная игра в шашки. Игра очень давно известная человечеству, прошедшая бок о бок с ним не одно тысячелетие и при этом, в своей основе, практически не изменившаяся. Правила её просты и ими с лёгкостью может овладеть даже дошкольник в отличие от более сложных шахмат. Игра относится к стратегическим, а подобные игры прекрасно подходят для развития и коррекции качеств, которым по нашему мнению уделяется так мало внимания.

В ходе выполнения работы был проведён поиск и изучение известных фактов о возникновении и развитии игры. Систематизированы её правила и требования к реквизиту.

На основе полученных данных проведено исследование возможных моделей представления игровой механики, рассмотрены их слабые и сильные стороны. Выбором стала модель на основе «битбордов», как наиболее точно её отражающая и легко адаптируемая к возможному дальнейшему развитию функциональности.

Была выполнена разработка и дано математическое обоснование основных алгоритмов, которые будут реализованы непосредственно в приложении. Разработана инфологическая модель данных, которых будет храниться в базе данных приложения. Выделены сущности и связи между ними. Проведено преобразование этой модели в реляционную – связанные отношениями табличные структуры которыми оперирует СУБД SQLite.

Выполнено описание данных структур в виде команд языка DDL. Разработаны запросы на языке SQL необходимые для обмена данными между СУБД и разрабатываем приложением.

Модель и применяемые к ней алгоритмы, адаптированы и изложены в терминах языка программирования C++. В интегрированной среде разработки QT Creator реализован набор основных и вспомогательных классов отвечающих за отображение игрового реквизита а так же справочной и архивной информации, взаимодействие с ними игрока, расчёт возможных ходов с использованием модели, запись и извлечение информации о ходе игры из долговременного хранилища. Для разработки базы данных в качестве инструмента был использован редактор SQLiteStudio.

В ходе выполнения работы был отмечен ряд положительных сторон у инструментов выбранных для реализации проекта.

Так, СУБД SQLite представляется лучшим выбором для хранения локальных данных в компактных и мобильных приложениях. В первую очередь – это возможность отказаться от дополнительного фонового процесса с которым необходимо поддерживать соединение как во многих классических СУБД, построенных на архитектуре клиент-сервер. Такой процесс не только существенно повышает требования к аппаратной конфигурации системы на которой запущено приложение, но и увеличивает время требуемое для написания взаимодействующего кода и его отладки. Избавиться от описанной проблемы позволяет решение разработчиков поставляющих драйвер этой СУБД в виде подключаемой библиотеки имеющей версии для всех основных современных ОС. Кроме прочего подобный подход позволяет снять требование на наличие

установленного в системе драйвера. Во вторых – использование нестрогой типизации данных тоже позволяет ощутимо ускорить разработку за счёт снятия необходимости скрупулёзно отслеживать типы и размеры полей данных. Конечно для сложных многопользовательских решений с удалёнными подключениями такой вариант не подойдёт, но в своей нише он вне конкуренции.

SQLite и запросы к ней представляют собой лишь хранилище и поставщика необходимых данных. Основная же часть приложения написана на языке C++ с использованием фреймворка QT.

Его (QT) неоспоримым плюсом является качественная реализация кроссплатформенности. Разработка приложения велась под ОС семейства Linux, но это не помешало скомпилировать его в ОС Windows без необходимости внесения каких-либо изменений в исходный код.

Крайне эффективен в работе и механизм обмена сообщениями сигнал-слот уже описанный в 3.3.

Собственная объектная иерархия позволяет в большинстве случаев не беспокоиться об удалении динамически созданных объектов, а поручить это дело объекту назначенному им в качестве родителя, который выполнит это действие в момент собственного уничтожения. Объекты в дереве иерархии не обязаны иметь одинаковый тип, достаточно быть наследником QObject. Это ещё один шаг в сторону ускорения процесса разработки.

Скорость разработки занявшей десять дней не в последнюю очередь объясняется выбранной связкой QT+SQLite и отличной встроенной документацией первого из них. Опыт работы с данными технологиями произвёл сильное впечатление и на ближайшее время они точно станут основными

Разработанное приложение имеет и некоторый потенциал развития.

Модульность позволяет реализовать наборы правил для разных диалектов игры с возможностью переключения между ними. Выбранная модель представления в принципе позволяет реализовать быстродействующие алгоритмы построения дерева ходов и принятия на его основе стратегических решений. Другими словами – создать реализацию ИИ противника. В текущей архитектуре не потребует большого труда добавить подсказки для игрока, настраиваемую стилизацию приложения, отмену сделанного хода и даже ветвление хода партии. К сожалению времени отводимого на выполнение работы оказалось недостаточно для воплощения этих идей.

Начиная с последних этапов разработки к тестированию привлекались представители целевой аудитории (дети семи и восьми лет). По результатам наблюдений был заметен живой интерес к игре в таком формате. Велось обсуждение ошибок и выигрышных стратегий. В одном случае отмечено быстрое и ощутимое усиление противника. Розыгрыш совместных партий, пусть и редко, продолжается по сей день.

На основании совокупности всех приведённых выводов и наблюдений цели работы можно считать достигнутыми.

Список использованных источников

1 **ГОСТ 19.105-78** Единая система программной документации. Общие требования к программным документам. – М.: Стандартинформ, 2010 – 4 с.

2 **ГОСТ 19.404-79** Единая система программной документации. Пояснительная записка. Требования к содержанию и оформлению. – М.: Стандартинформ, 2010 – 3 с.

3 **ГОСТ 19.401-78** Единая система программной документации. Текст программы. Требования к содержанию и оформлению. – М.: Стандартинформ, 2010 – 2 с.

4 **ГОСТ 19.402-78** Единая система программной документации. Описание программы. – М.: Стандартинформ, 2010 – 4 с.

5 **Ашарина И. В.** Основы программирования на языках С и С++ / И. В. Ашарина. – Горячая линия-Телеком, 2015. – 207 с.

6 **Баранов А.** Интернет-психология / В. Баранов. – Инфра-М, 2015. – 264 с.

7 **Вайсфельд М.** Объектно-ориентированное мышление / М. Вайсфельд. – Питер, 2018. – 304 с.

8 **Васильев А. Н.** Объектно-ориентированное программирование на С++ / А. Н. Васильев. – Наука и техника, 2016. – 543 с.

9 **Вирный А.** Немного о шашках, но по существу / А. Вирный. – Russian Chess House, 2013. – 320 с.

10 **Гамма Э.** Приёмы объектно-ориентированного проектирования / Э.

Гамма, Р. Хелм, Р. Джонсон, Д. Влисседес. – Питер, 2018. – 366 с.

11 **Гаст Х.** Объектно-ориентированное проектирование: концепции и программный код / Х. Гаст. – Альфа-книга, 2018. – 1040 с.

12 **Головков Ю.** Играем в шашки. Тренеру на заметку / Ю. Головков. – Ростов-на-Дону, 2015. – 160 с.

13 **Грошев, А.С.** Основы работы с базами данных (2-е изд.) / А.С. Грошев. – М. : НОУ «Интуит», 2016. – 256 с.12

14 **Илюшечкин В. М.** Основы использования и проектирования баз данных / В. М. Илюшечкин. – Юрайт, 2018. – 213 с.

15 **Карпова И. П.** Базы данных. Учебное пособие / И. П. Карпова. – Питер, 2018. – 240 с.

16 **Конноли Т.** Базы данных. Проектирование, реализация и сопровождение. Теория и практика / Т. Конноли, К. Бегг. – Вильямс, 2018. – 1439 с.

17 **Лафоре Р.** Объектно-ориентированное программирование в C++. Классика Computer Science / Р. Лафоре. – Питер, 2018. – 928 с.

18 **Мартин Р.** Чистый код: создание, анализ и рефакторинг / Р. Мартин. – Питер, 2018. – 464 с.

19 **Медведев В.** Шашки для начальной школы / В. Медведев. – Феникс, 2016. – 91 с.

20 **Мосин М.** Как обыграть папу в шашки / М. Мосин. – Эксмо, 2015. – 64 с.

21 **Невар Н.** Русские шашки. Комбинации и жертва шашки / Н. Невар. – ПитерСПб, 2016. – 208 с.

22 **Пассиг К.** Программирование без дураков / К. Пассиг, Й. Яндер. –

Питер, 2017. – 416 с.

23 **Пахомова Т. В.** Некоторые психологические проблемы интернет-зависимости / Т. В. Пахомова // Молодой ученый. — 2014. — №15. — С. 236-238.

24 **Погрибной В.** Шашки для детей / В. Погрибной, В. Юзук. – Феникс, 2015. – 137 с.

25 **Стойлова Л.** Теоретические основы начального курса математики. Учебное пособие / Л. Стойлова. – Academia, 2014. – 272 с.

26 **Стружкин Н. П.** Базы данных: проектирование. Практикум / Н. П. Стружкин, В. В. Годин. – Юрайт, 2018. – 291 с.

27 **Ульман Д.** Реляционные базы данных / Д. Ульман, Д. Уидом. – Лори, 2018. – 374 с.

28 **Фёдорова Н. Г.** Основы проектирования баз данных / Н. Г. Фёдорова. – Академия, 2018. – 219 с.

29 **Фуфаев Э. В.** Базы данных / Э. В. Фуфаев, Д. Э. Фуфаев. – Академия, 2017. – 320 с.

30 **Шилдт Г.** Справочник программиста по C/C++ / Г. Шилдт. – Вильямс, 2018. – 429 с.

31 **Шилдт Г.** C++: полное руководство, классическое издание / Г. Шилдт. – Вильямс, 2018. – 796 с.

32 **Шлее, М.** Qt 5.3 Профессиональное программирование / М. Шлее. – БХВ-Петербург, 2015. – 928 с.

33 **Owens M.** The Definitive Guide to SQLite / M. Owens. – Apress, 2014. – 466 с.

Приложение 1

Рисунки и листинги

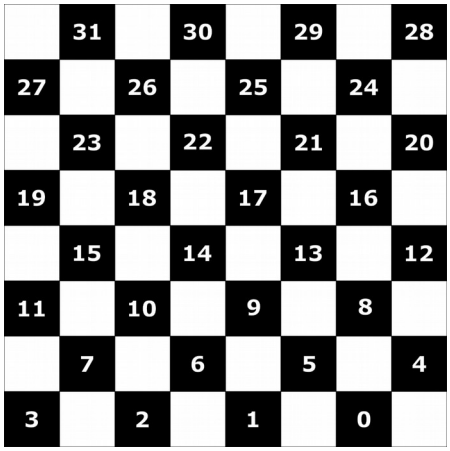

Рисунок 1. Соответствие клеток битам числа

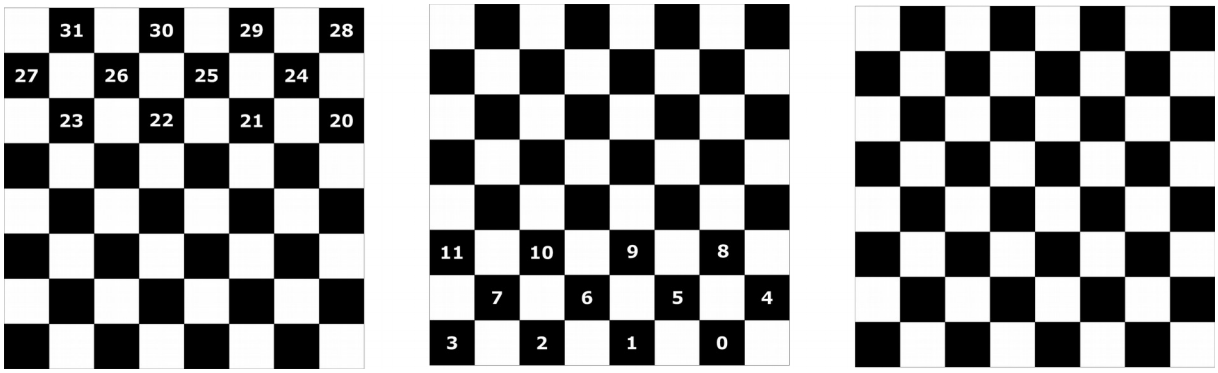

Рисунок 2. Пример начальной расстановки (чёрные, белые, дамки)

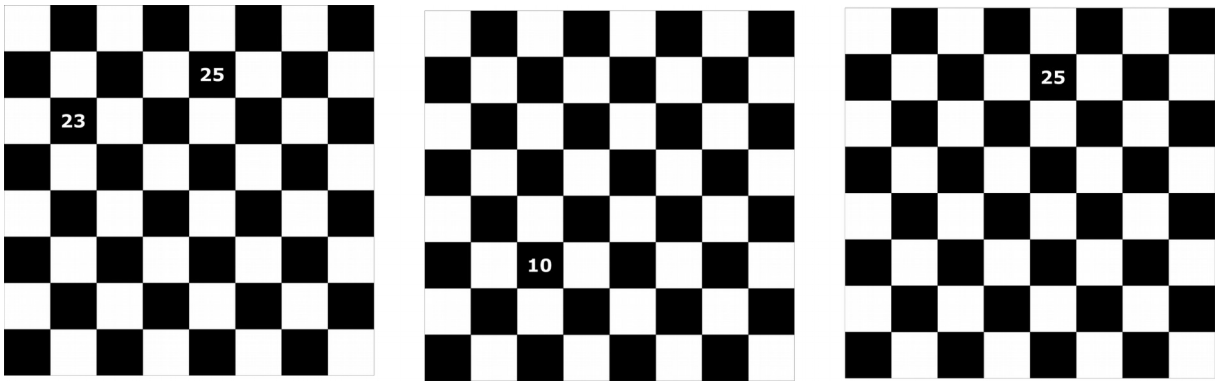

Рисунок 3. Ещё один пример расстановки (чёрные, белые, дамки)

Рисунок 4. Инфологическая модель данных об играх.

Рисунок 5. Даталогическая модель данных об играх

Листинг 1 алгоритм поиска ходов для простой фигуры

```
ФИГУРЫ = ДОСТУПНЫЕ \ ДАМКИ
для каждого ФИГУРА∈ФИГУРЫ
{
    если ( ФИГУРА∈ДОСТУПНЫЕ.СВ )
    {
```

$ФИГУРА = s.СВ(ФИГУРА)$

если ($ФИГУРА \in ЖЕРТВЫ.СВ$)

{

$ЖЕРТВА = ФИГУРА$

$ХОД.ЖЕРТВА = ХОД.ЖЕРТВА \cup ЖЕРТВА$

$ФИГУРА = s.СВ(ФИГУРА)$

$ВЗЯТИЯ = ВЗЯТИЯ \cup \{ФИГУРА, ЖЕРТВА\}$

$ХОД.ВЗЯТИЕ = ХОД.ВЗЯТИЕ \cup ФИГУРА$

}

$ХОД.ФИНИШ = ХОД.ФИНИШ \cup ФИГУРА$

если ($ФИГУРА \in ГРАНИЦА$)

{

$ХОД.ДАМКА = ХОД.ДАМКА \cup ФИГУРА$

}

}

}

Листинг 2 алгоритм поиска ходов для дамки

$ФИГУРЫ = ДОСТУПНЫЕ \cap ДАМКИ$

для каждого $ФИГУРА \in ФИГУРЫ$

{

$ПОДНЯТА = ФИГУРА$

если $ФИГУРА \in ДОСТУПНЫЕ.СВ$

{

$ФИГУРА = s.СВ(ФИГУРА)$

пока $ФИГУРА \neq \varnothing$

{

если $ФИГУРА \in ВСЕ$

{

если $ЖЕРТВА \neq \varnothing$ **прервать цикл**

если $ФИГУРА \in ЖЕРТВЫ.СВ$

{

$ЖЕРТВА = ФИГУРА$

$ЖЕРТВЫ = ЖЕРТВЫ \cup ЖЕРТВА$

$ОХОТНИКИ = ОХОТНИКИ \cup ПОДНЯТА$

$ХОД.ЖЕРТВА = ХОД.ЖЕРТВА \cup ЖЕРТВА$

}

иначе прервать цикл

}

$ХОД.ФИНИШ = ХОД.ФИНИШ \cup ФИГУРА$

если $ЖЕРТВА \neq \varnothing$

{

$ВЗЯТИЯ = ВЗЯТИЯ \cup \{ФИГУРА, ЖЕРТВА\}$

}

$ФИГУРА = s.СВ(ФИГУРА)$

}

}

}

Приложение 2

Компакт-диск

Прилагаемый диск содержит:

– Текст данной пояснительной записки в форматах .odt и .pdf.

– Исходные тексты проекта на языке C++.

– Исполняемый файл приложения скомпилированный для процессорной архитектуры x_86, предназначенный для исполнения в среде ОС Windows.

– Двоичный файл разработанной БД содержащий набор данных и необходимый для работы приложения.

– SQLiteStudio (версия 3.1.1 для Windows) – среда разработки баз данных SQLite.

– QtCreator (версия 4.7.1 для Windows) – интегрированная среда разработки на языке C++ с использованием кросс-платформенных библиотек Qt.

– Набор кросс-платформенных библиотек QT (версии 5.11.2