МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение
высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

В.В. Шишкин, Д.С. Афонин

РАЗРАБОТКА ЛОГИЧЕСКИХ КОМПЬЮТЕРНЫХ ИГР С ГРАФИЧЕСКИМ ИНТЕРФЕЙСОМ В СРЕДЕ ПИТОН

Учебное пособие

Ульяновск
УлГТУ
2023

УДК 004 (075)
ББК 32.973-018.1я7
Ш 12

Рецензенты:
Главный конструктор по серии АО «УКБП», канд.техн.наук Комиссаров А.В.
Генеральный директор ООО «ЦТТ», канд.техн.наук Святов К.В.

Утверждено редакционно-издательским советом университета

в качестве учебного пособия

Шишкин, Вадим Викторинович

Ш 12 Разработка логических компьютерных игр с графическим интерфейсом в среде Питон: учебное пособие для студентов направления 09.03.02 «Информационные системы и технологии» / В.В. Шишкин, Д.С. Афонин. – Ульяновск : УлГТУ, 2023. – 88 с.

ISBN 978-5-9795-2339-2

Целью данного учебного пособия является изучение методов разработки компьютерных приложений с графическим интерфейсом на языке Питон. Изложение ведется в применении данных методов в области компьютерных игр. На их основе даются методические материалы по выполнению курсовой работы по дисциплине «Алгоритмы и структуры данных».

Работа подготовлена на кафедре «Измерительно-вычислительные комплексы».

УДК 004 (075)
ББК 32.973-018.1я7

ISBN 978-5-9795-2339-2

ОГЛАВЛЕНИЕ

ВВЕДЕНИЕ

Язык программирования Питон является в настоящее время одним из самых развитых и используемых языков. Его применяют как для разработки систем искусственного интеллекта, так и для широкого класса других приложений, включая графические, интернет, параллельные и др. Он хорошо зарекомендовал себя в сфере обучения программированию.

В настоящее время широкий круг пользователей компьютерных приложений привык к работе на компьютере через графический интерфейс. Качество графического интерфейса часто определяет привлекательность (в значительной степени коммерческий успех) приложения. Однако программирование графических приложений имеет ряд особенностей, реализация которых требует изучения особых методов.

Кроме графического интерфейса в приложении должна быть реализована определенная логика поведения, при необходимости с элементами искусственного интеллекта. Реализация логики поведения базируется на определенной группе моделей, структур данных и алгоритмов.

Хорошим полигоном для изучения и отработки этих двух особенностей графических приложений могут служить логические компьютерные игры. Тем более, что игровой сегмент компьютерных приложений переживает в последнее время взлет. И в процессе обучения студенты получат дополнительные компетенции в разработке приложений для быстро растущего рынка развлекательной индустрии.

На рис. 1 представлены известные логические игры с точки зрения сложности реализации на компьютере [1]. Игры разделены на 4 группы:

– простые (компьютер никогда не проигрывает человеку);

– компьютер может (не всегда) выиграть у профессионалов-людей;

– компьютер пока еще проигрывает профессионалам-людям;

– сложные (компьютер никогда не сможет обыграть человека).

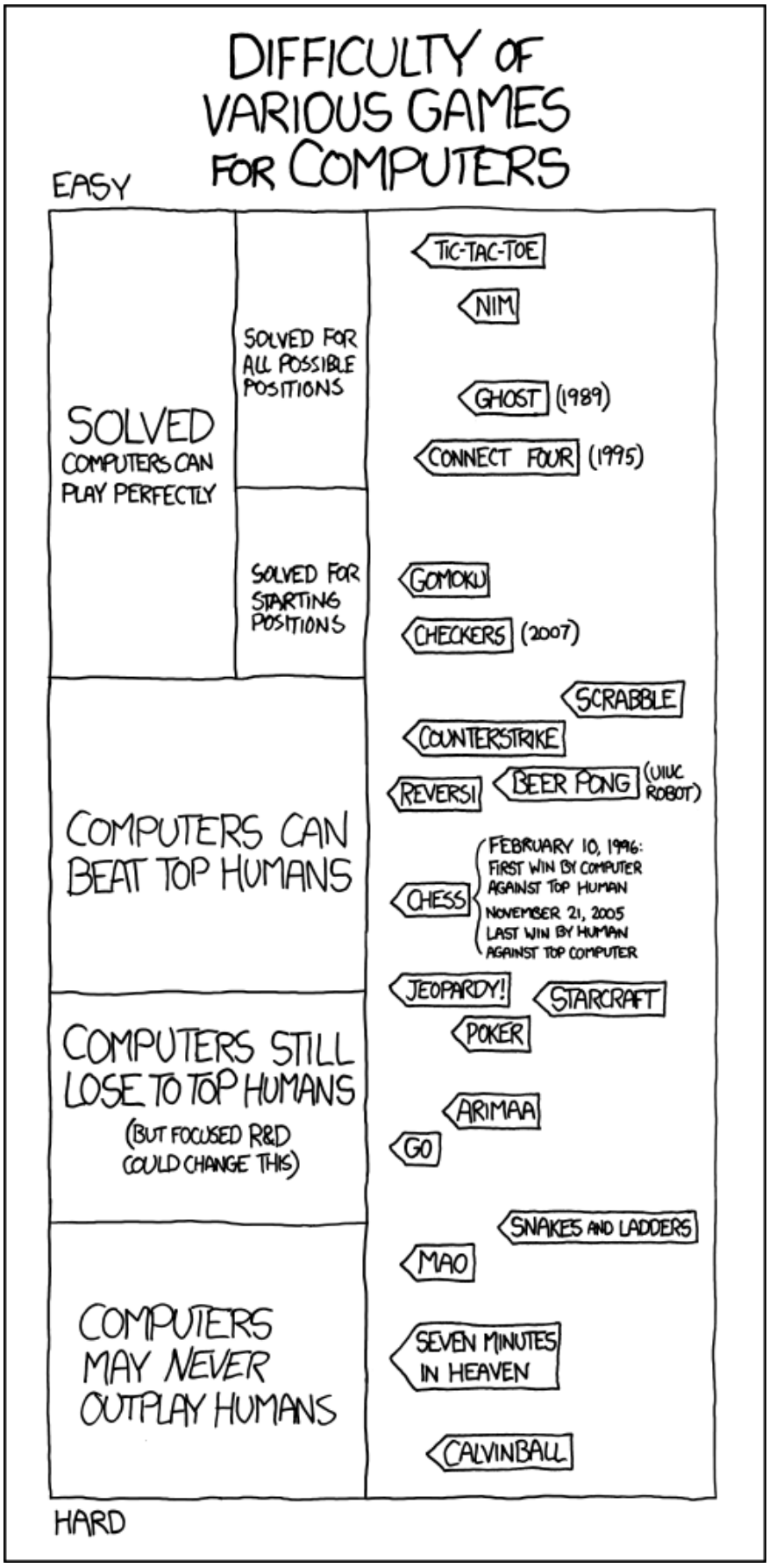

Рисунок 1. Классификация игр по сложности

В каждой группе представлены наиболее известные логические игры, расположенные относительно друг друга по сложности. На рис. 1 первая группа начинается игрой в «крестики-нолики», а заканчивается шашками, реверси и шахматы относятся ко второй группе, покер и го – к третьей, а кальвинбол – к четвертой.

Разработка такого графического приложения по объему и сложности решаемых задач выходит за рамки лабораторного практикума и может быть реализована в форме курсовой работы.

Выполнение курсовой работы студентом – это важный этап теоретического и практического изучения дисциплины, который базируется на обобщении ранее выполненных работ.

Курсовая работа по дисциплине «Алгоритмы и структуры данных» предполагает разработку графического приложения, в рамках которой на языке Питон реализуется логическая настольная игра. Также студент получает компетенции по разработке определенных элементов технической документации в соответствии с 19 группой государственных стандартов (ГОСТ) [2]: техническое задание, пояснительная записка и руководство программиста.

ЭЛЕМЕНТЫ ТЕОРИИ ИГР

Как правило, под теорией игр понимают раздел математики, изучающий математические модели принятия решений в конфликтных ситуациях. Для конфликтной ситуации характерно участие нескольких (минимум двух) сторон, называемых игроками и имеющих несовпадающие (часто противоположенные) интересы. Несовпадающие интересы игроков связаны с некоторой целью, которую игроки стараются достичь. С точки зрения основателя кибернетики Джона фон Неймана, конфликт – это взаимодействие двух

объектов, обладающих несовместимыми целями и способами достижения этих целей.

В экономике, торговле, юриспруденции, политике и других сферах наблюдается достаточно много конфликтных ситуаций различной природы и сложности. Их математические модели, изучаемые в теории игр, весьма разнообразны.

Основы теории игр изучаются в средней школе и проверяются в заданиях ЕГЭ (задания 19-21). Там изучаются основные категории теории игр, навыки построения дерева игры, определение выигрышной стратегии. При этом под деревом игры понимается дерево (граф без циклов), корнем которого является начальная игровая ситуация, а далее последовательно строго по ярусам рассматриваются все возможные ходы (вершины графа) первого и второго игрока до окончания игры.

А стратегия – это набор правил, по которым выбираются ходы, или конкретная последовательность ходов. При этом выигрышная стратегия – это стратегия, приведшая игрока к выигрышу.

Таким образом, игра – это реализация одного из сценариев развития конфликта от начала до конца. Игроки – это конкурирующие стороны, конфликта. Ход – это одно из возможных действий игрока. Стратегия – это правила, по которым игрок выбирает ходы или последовательность ходов. Исход – это итог игры. Смысл теории игр заключается в поиске (динамическом построении) оптимальной (максимально выигрышной) стратегии для конкретного игрока.

Оптимальным решением по определению является решение, обеспечивающее достижение глобального экстремума целевой функции при удовлетворении функций ограничений заданным значениям. Таким образом, для определения оптимальной стратегии необходимо определить

целевую функцию, по которой будет проводиться оптимизация (поиск глобального экстремума).

Мы будем рассматривать логические игры, относящиеся к классу некооперативных игр с полной информацией и нулевой суммой. Игры с нулевой суммой – это игры, в которых выигрыш одного игрока равняется проигрышу другого. Таким образом, должна быть определена функция, оценивающая игровую ситуацию с точки зрения выигрыша/проигрыша.

Возможны несколько вариантов определения оптимальной стратегии.

Вариант первый. Для не слишком сложных игр все дерево игры может быть предварительно просчитано и храниться в виде дерева решений в программе. На каждом ходе из него будет выбран лучший вариант. Такой вариант возможен, например, для стандартной игры «крестики-нолики» на игровом поле 3×3. А для усложненного варианта этой игры, такой как «5 в ряд» на игровом поле 20×20, этот вариант, из-за существенного роста количества разных версий ходов, уже является проблематичным.

Вариант второй. Выработка набора правил, по которым на основе анализа игровой ситуации происходит выбор очередного хода. В качестве примера можно ознакомиться с программой «крестики-нолики» по книге М. Доусона [14].

Вариант третий. Выбор очередного хода происходит на основе расчета по специальным алгоритмам, таким как, например, МиниМакс или др. Алгоритм МиниМакса будет рассмотрен в разделе по разработке игрового интеллекта нашего приложения. Возможна комбинация третьего и второго вариантов. В этом случае в расчетные алгоритмы могут включаться специальные правила в форме эвристик, позволяющих сократить количество переборов.

Вариант четвертый. Определение ходов с помощью обученной нейронной сети. Данный вариант стал возможен в силу существенного

роста вычислительных возможностей современных компьютеров и развития теории и практики искусственных нейронных сетей. В этом варианте разработанная нейронная сеть обучается на большом материале сыгранных партий (обучающая выборка) и после обучения может генерировать ходы на достаточно высоком уровне.

Вариант пятый. Выбор хода случайным образом. Это, с точки зрения реализации, самый простой вариант. И очевидно, что он самый худший.

Вариант шестой. Использование методов псевдослучайной оптимизации, таких как метод генетического алгоритма, метод Монте-Карло и т. п. Данный вариант не обеспечивает выбор оптимального хода, но, не затрачивая много времени, может предложить достаточно приемлемый (псевдооптимальный) вариант.

Методы теории игр возникли и применялись для решения конкретных задач (как правило, экономических) задолго до появления компьютеров. Появление компьютеров повысило их точность и эффективность, расширило применимость, а также открыло новые возможности для их развития, перевело их на другой уровень. Кроме автоматизации расчетных процедур, повышения точности и эффективности методов теории игр появление компьютеров привело к появлению и бурному развитию нового сегмента в сфере IT-бизнеса – компьютерным играм. Так только по играм для персонального компьютера в 2022 году рынок достиг объема 38,2 млрд долларов.

В настоящее время существует огромное количество компьютерных игр разных типов и жанров. Их создание привело к разработке большого количества специальных моделей, методов и алгоритмов, обеспечивающих реализацию данных игр. В данном учебном пособии рассмотрим подходы, модели, методы и алгоритмы, необходимые для разработки компьютерных приложений для игры в настольные логические игры.

Контрольные вопросы

1. Что такое теория игр?
2. Дайте определения игры, игроков, хода, дерева игры, стратегии, выигрышной стратегии.
3. Приведите пример дерева игры.
4. Что такое игра с нулевой суммой?
5. Что такое оптимальное решение?
6. Что такое целевая функция?
7. Какие варианты определения оптимальной стратегии вы знаете?
8. Что такое эвристика?
9. Что такое обучающая выборка?
10. Какой граф является деревом?
11. Какова роль появления компьютеров в теории игр?
12. Всегда ли вариант случайного выбора хода худший?

ОСОБЕННОСТИ ОРГАНИЗАЦИИ ГРАФИЧЕСКИХ ПРИЛОЖЕНИЙ

С точки зрения интерфейса пользователя современные компьютерные приложения разделяют на два типа: с графическим интерфейсом (GUI – graphical users interface), и с алфавитно-цифровым интерфейсом (CLI – command line interface). У них есть несколько отличий:

– CLI приложения работают в алфавитно-цифровом режиме дисплея, а GUI приложения – в графическом. Их совмещение невозможно;

– CLI приложения работают в окне консоли среды программирования, а GUI приложения – только в специально созданном графическом окне (виджете);

– CLI приложения имеют конец программы, а GUI приложения – исполняют бесконечный цикл, который прерывается по определенному событию.

Таким образом, GUI приложения – это событийно-ориентированные приложения, т. е. управляемые событиями, происходящими в процессе работы приложения. В процессе выполнения данного бесконечного цикла программа отслеживает появление того или иного события. В приложении может происходить множество событий. Как правило, события по мере поступления записываются в очередь событий, из которой они выбираются для обработки. С каждым событием связана функция – обработчик события, которая начинает выполняться при появлении данного события. Источниками событий могут быть сигналы от мыши, клавиатуры, таймера или завершение какого-либо процесса. Если у какого-то события нет обработчика, то оно игнорируется. Ряд источников событий связан с взаимодействием пользователя с графическими элементами интерфейса, а ряд нет. Например, событие нажатия на клавиатуре комбинации клавиш Ctrl+C не зависит от графических элементов, изображенных в окне, а нажатие левой кнопки мыши на крестике в правом верхнем углу окна – зависит.

В настоящее время почти все графические интерфейсы строятся в соответствии с моделью WIMP – window, icon, menu, pointer. В ней ключевым понятием является виджет – стандартизованный компонент GUI, видимый на экране, с которым взаимодействует пользователь. По ходу программы виджеты могут создаваться и уничтожаться, их вид может меняться. Виджет может находиться внутри другого виджета. Главное окно программы является корнем дерева виджетов, обход которого и приводит к формированию графического интерфейса.

Таким образом, программирование GUI приложения сводится к созданию иерархии виджетов и программированию обработчиков

событий. При этом можно использовать как процедурное, так объектно-ориентированное программирование.

Контрольные вопросы

1. Как подразделяются компьютерные приложения с точки зрения интерфейса?
2. Чем отличаются графические и командные приложения?
3. Что такое GUI приложение, как оно работает?
4. Что такое модель WIMP?
5. Что такое виджет?
6. Какими функциями обладает виджет?
7. В чем заключается программирование GUI приложении?
8. Может ли быть GUI приложение объектно-ориентированным?
9. Может ли в приложении быть несколько главных графических окон?
10. С помощью каких парадигм программирования можно создавать GUI приложение?

ЭЛЕМЕНТЫ ТЕХНОЛОГИИ РАЗРАБОТКИ ПРОГРАММНОГО ОБЕСПЕЧЕНИЯ

Любая промышленная разработка компьютерного приложения (программного обеспечения (ПО)) опирается на использование определенной технологии разработки. Под технологией подразумевают некоторые модели, инструменты и методы применения, позволяющие эффективно вести разработку ПО. Без опоры на технологические приемы разработка серьезного приложения (более 1000 строк кода) становится в лучшем случае крайне неэффективной, а в худшем – невозможной. В связи

с ростом сложности современных компьютерных приложений вопросам технологии разработки стали уделять все больше внимания.

Любая технология возникает и развивается в рамках определенной модели жизненного цикла ПО. В последнее время появилось достаточно много разнообразных технологий разработки ПО и их модификаций [3], включая классическую каскадную (водопадную) модель, различные модификации гибкой разработки (Agile) и др. Мы будем использовать каскадную модель с некоторыми уточнениями. Эта модель включает следующие основные этапы жизненного цикла ПО (рис. 2).

Рисунок 2. Каскадная модель жизненного цикла ПО

Контрольные вопросы

1. Зачем нужна технология разработки ПО?
2. Чем обусловлена объективная потребность в технологиях разработки ПО?
3. Какие технологии ПО вы знаете?
4. Какие этапы включает каскадная модель жизненного цикла ПО?

РАЗРАБОТКА ТЕХНИЧЕСКОГО ЗАДАНИЯ

Разработка любого компьютерного приложения начинается с разработки технического задания (ТЗ). ТЗ является четким описанием (желательно в формализованном стиле) требований к разрабатываемому приложению: к его функциональности, особенностям, вариантам и режимам использования, группам пользователей и т. д. Разработка требований в форме ТЗ является, как видно из рис. 2, первым этапом жизненного цикла программного обеспечения. Если у разработчика (аналитика) разработка ТЗ вызывает затруднения, связанные со сложностью (неоднозначностью) задачи, отсутствием необходимых компетенций в предметной области и т. п., перед разработкой ТЗ присутствует этап предпроектного исследования, который также можно отнести к разработке требований. Следует очень ответственно отнестись к разработке ТЗ, т. к. ошибки (неточности, неоднозначности), допущенные в ТЗ, породят множество ошибок при разработке, которые впоследствии сложно диагностируются, исправляются и иногда приводят к тому, что приходится переписывать все приложение целиком.

ТЗ является очень важным документом. Он обычно согласуется с заказчиком и именно по нему происходит сдача проекта. Существует насколько вариантов разработки ТЗ. Они базируются на положительном

опыте программистских компаний. С основными направлениями в разработке ТЗ можно ознакомиться в публикации [4]. Мы будем опираться на стандартизованные документы. В информационных технологиях разработчики опираются, как правило, на две группы государственных стандартов (ГОСТов): 19 и 34. Для нашей задачи предпочтительнее ГОСТы 19 группы. Рекомендуем разрабатывать ТЗ в соответствии с документом «ГОСТ 19.201-78 Техническое задание. Требования к содержанию и оформлению» [5]. Шаблон ТЗ для курсовой работы приведен в приложении 3.

ТЗ включает несколько групп требований. С точки зрения разработки основной группой является группа функциональных требований, на которых мы и сосредоточимся. Остальные могут быть разработаны по шаблону с учетом специфики решаемой задачи.

Дальнейшее изложение будем вести на примере приложения по игре в уголки [6]. Поскольку возможны разные варианты правил, то необходимо провести предпроектное исследование и согласовать с заказчиком те правила, по которым приложение будет вести игру. Согласованные правила оформляются отдельным документом, либо сразу явно включаются в функциональные требования ТЗ на разработку. Выберем второй вариант. Далее представим возможный вариант функциональных требований (часть ТЗ) для разработки приложения по игре в уголки.

Функциональные требования для разработки приложения по игре в уголки

Функциональное назначение

Требуется разработать однопользовательское десктопное приложение по игре в уголки с графическим интерфейсом в среде Windows.

Требования к функциональным характеристикам

Приложение должно соответствовать следующим правилам игры.

Игра ведется между двумя соперниками (пользователь-компьютер) шашками разного цвета на квадратном поле размером 8×8 клеток.

В начальной позиции у каждого игрока 9 шашек, построенных квадратом (3×3) в углу игрового поля диагонально напротив шашек противника. Данные 9 клеток называются домом игрока.

Игроки совершают ходы поочередно. Первый ход делает пользователь. В процессе хода игрок может переместить только одну свою шашку. Возможны ходы двух типов. Первый тип – игрок может переместить шашку по горизонтали или вертикали на любую соседнюю пустую клетку. Второй тип – игрок может своей шашкой «перепрыгнуть» через занятую шашкой клетку, если за ней есть пустая клетка. В этом случае ход может быть продолжен, пока есть возможность дальнейшего «прыжка». «Прыжки» могут быть остановлены игроком в любой позиции.

Цель игры – как можно скорее занять дом противника. Игра завершается, когда один из игроков займет дом противника или один из игроков не выведет из своего дома хотя бы одну шашку в течение 40 ходов. Игрок, первым занявший дом противника, считается победителем, второй игрок – проигравшим. Если игроки заняли дома одновременно (по количеству ходов) – объявляется ничья. Игрок, не выведший из своего дома хотя бы одну шашку в течение 40 ходов, считается проигравшим, второй игрок – победителем.

Требования к структуре приложения

Приложение должно быть разработано в виде одного модуля с дополнительными информационными файлами при необходимости.

Требования к составу функций приложения

В приложении должны быть реализованы в графическом режиме следующие основные функции:

– регистрация/авторизация пользователя;

– отрисовка игрового поля;

– взаимодействие с пользователем;

– интерактивные прием, проверка правильности и отрисовка хода пользователя;

– проверка окончания игры;

– вычисление, проверка правильности и отрисовка хода компьютера;

– информирование пользователя об окончании игры и победителе.

Требования к организации информационного обеспечения, входных и выходных данных

В приложении должен быть реализован графический интерфейс взаимодействия с пользователем. Изображения шашек могут храниться в отдельных графических файлах. Логин и пароль пользователя должны вводиться с клавиатуры. Логины и пароли зарегистрированных пользователей должны храниться в отдельном файле или базе данных в зашифрованном виде. Пояснительные информационные сообщения для пользователя должны выводиться внизу игрового поля по ходу игры.

Практическая часть

1. Напишите функциональные требования для игры в крестики-нолики.
2. Напишите функциональные требования для приложения в соответствии с вашим вариантом курсовой работы.

Контрольные вопросы

1. Что такое техническое задание?
2. Какие цели преследует техническое задание?
3. Какие разделы включает техническое задание?
4. Какой раздел технического задания принципиален для разработки?
5. Что такое предпроектное исследование и зачем оно нужно?
6. Какими группами государственного стандарта нужно пользоваться при составлении технического задания?

7. Какую роль играет техническое задание в жизненном цикле приложения?

8. Какие неточности (неоднозначности) присутствуют в приведенном ТЗ для приложения игры в крестики-нолики?

РАЗРАБОТКА МОДЕЛИ

Как следует из каскадной модели (рис.2), следующим шагом после разработки ТЗ является проектирование приложения. В основе любого компьютерного приложения лежит некоторая формальная модель, описывающая интересующую нас систему или процесс с необходимой степенью детализации и в терминах задачи, которую мы решаем относительно данной системы или процесса. Разработка модели ведется аналитиками, хорошо знающими предметную область и владеющими определенным набором математических моделей. Этап разработки модели не всегда явно присутствует при проектировании приложения. Для не очень сложных приложений программист сам проектирует приложение, опуская явную разработку модели и переходя сразу к разработке архитектуры и алгоритмов приложения. В этом случае он, опираясь на свои знания математических моделей и опыт предыдущих разработок, в уме в неявном виде представляет эту модель. Такой подход в серьезной разработке является не эффективным даже для не слишком сложных приложений. Принятые программистом не вербализованные проектные решения не позволяют правильно работать в команде, ведут к множеству ошибок, усложняют разработку программной документации, не позволяют эффективно сопровождать приложение в процессе его эксплуатации.

На ранних этапах развития информационных технологий определилась концепция разделения компьютерной системы на

аппаратную и программную части. В программной части в свою очередь выделяют декларативную (структуры данных) и процедурную (алгоритмы) части. Наиболее четко это представлено в книге Н. Вирта «Алгоритмы + структуры данных = программы» [7].

Таким образом, в качестве модели будем разрабатывать определенный набор структур данных и алгоритмов, позволяющих решить задачу в виде компьютерного приложения (программы). Выбор структур данных и алгоритмов определяется исходя из функций приложения, зафиксированных в ТЗ.

Для нашего примера (игра в уголки) основными функциями из ТЗ, требующими вербализованного представления модели, являются:

– отрисовка игрового поля и ходов игроков;

– проверка правильности хода игрока и окончания игры;

– вычисление хода компьютера;

– информирование игрока.

Остальные функции можно разрабатывать, опираясь на опыт предыдущих разработок без обязательной вербализации. Главное – ничего не забыть, т. к. сдача проекта будет проводиться по требованиям, зафиксированным в ТЗ.

Принципиальным является модель игрового поля. Для функций визуализации, проверки правильности хода игрока и окончания игры в качестве модели (структуры данных и операций над ней) может быть выбрана матрица 8×8, со следующими правилами заполнения:

- «0» – пустая клетка;

- «1» – клетка занята черной шашкой;

- «-1» – клетка занята белой шашкой.

При вычислении хода компьютера приложение должно выбирать лучший ход. Для этого требуется специальная функция оценки качества хода. Для ее реализации модели игрового поля будет недостаточно.

Потребуется дополнительная матрица 8×8 весов клеток игрового поля. Например, следующая:

[[7, 6, 5, 4, 3, 2, 1, 0],
[8, 7, 6, 5, 4, 3, 2, 1],
[9, 8, 7, 6, 5, 4, 3, 2],
[10, 9, 8, 7, 6, 5, 4, 3],
[11, 10, 9, 8, 7, 6, 5, 4],
[12, 11, 10, 9, 8, 7, 6, 5],
[13, 12, 11, 10, 9, 8, 7, 6],
[14, 13, 12, 11, 10, 9, 8, 7]].

Логику данной матрицы поясним позже. Выбор необходимых структур данных для реализации тех или иных функций нужно осуществлять с учетом алгоритмов реализации этих функций. Как любое проектирование, данный процесс является итерационным, т. е. принятые решения могут уточняться или заменяться другими (более эффективными) на последующих итерациях.

Практическая часть

1. Разработайте основные структуры данных для игры в «крестики – нолики».
2. Разработайте основные структуры данных для приложения в соответствии с вашим вариантом курсовой работы.

Контрольные вопросы

1. Что такое модель для компьютерного приложения?
2. Какую форму имеет модель для компьютерного приложения?
3. Как связаны модель и техническое задание компьютерного приложения?

РАЗРАБОТКА АРХИТЕКТУРЫ ПРИЛОЖЕНИЯ

Следующим шагом в проектировании является разработка архитектуры приложения. Архитектура приложения (ПО) является одной из основополагающих концепций современных информационных технологий. Понятие архитектуры ПО стало актуальным в связи с увеличением сложности компьютерных приложений. Очевидно, что для программы в 50 строк не нужно разрабатывать архитектуру. Вся архитектура представляет собой один модуль. Однако современные программные приложения состоят из множества сложно взаимодействующих программных модулей и баз данных, совокупный объем которых может достигать сотен миллионов строк кода. Очевидно, что разрабатывать и сопровождать такие приложения без четкого проекта (архитектуры) не возможно. В настоящее время имеется множество книг, статей и интернет-публикаций по теме разработки архитектуры ПО, например [8,9,10,11]. Каждый из авторов по-своему понимает и описывает архитектуру ПО. Интересным представляется подход определения архитектуры через ее функции [10]. Согласно ему, архитектура ПО решает следующие задачи:

– определяет структуру программы и позволяет понять, как она устроена, на каких уровнях выполняются те или иные задачи и функции;

– определяет поведение и взаимодействие элементов, благодаря чему становится понятно, что происходит, если выполняется определенное действие;

– определяет значимые и второстепенные элементы, что позволяет оценить стоимость разработки, понять, какие элементы обязательно внедрять, а от каких можно отказаться в пользу экономического соображения;

– помогает понять, насколько программа масштабируема, как сложно будет внедрять новые функции и какой стек технологий использовать;

– позволяет удовлетворить потребности клиента и адаптировать программу под взаимоисключающие требования, например, высокий уровень функциональности и определение границ времени, в таком случае становится понятно, как это реализовать;

– позволяет понять логические взаимосвязи в программе;

– дает возможность корректно вести документацию и четко расписывать функционал, что существенно упрощает дальнейшее обслуживание программы, внесение изменений и работу с существующим функционалом.

Данное учебное пособие не преследует цель изложить теорию разработки архитектуры ПО. Как отмечалось выше, эта тема достаточно хорошо раскрыта во множестве книг, статей и интернет-публикаций [8,9,10,11] и др. Поисковый запрос «архитектура ПО» в поисковой системе выдает более 10 страниц ссылок по данной тематике, среди которых много достаточно качественных. Также там есть реклама основных книг, которые также можно найти в библиотеках и на web-ресурсах.

Для решения нашей задачи принципиальной будет первая функция из вышеприведенного списка. Исходя из нее, под архитектурой будем понимать взаимосвязанную совокупность основных программных и информационных компонент нашего приложения. В качестве примера приведем архитектуру приложения игры в уголки в форме графа вызова функций (рис. 3). Архитектура должна учитывать все функциональные требования ТЗ.

Архитектура, кроме общего представления о структуре приложения, позволяет четко определить требования к каждому модулю, интерфейсы их взаимодействия, область «видимости» структур данных и в целом эффективно управлять разработкой.

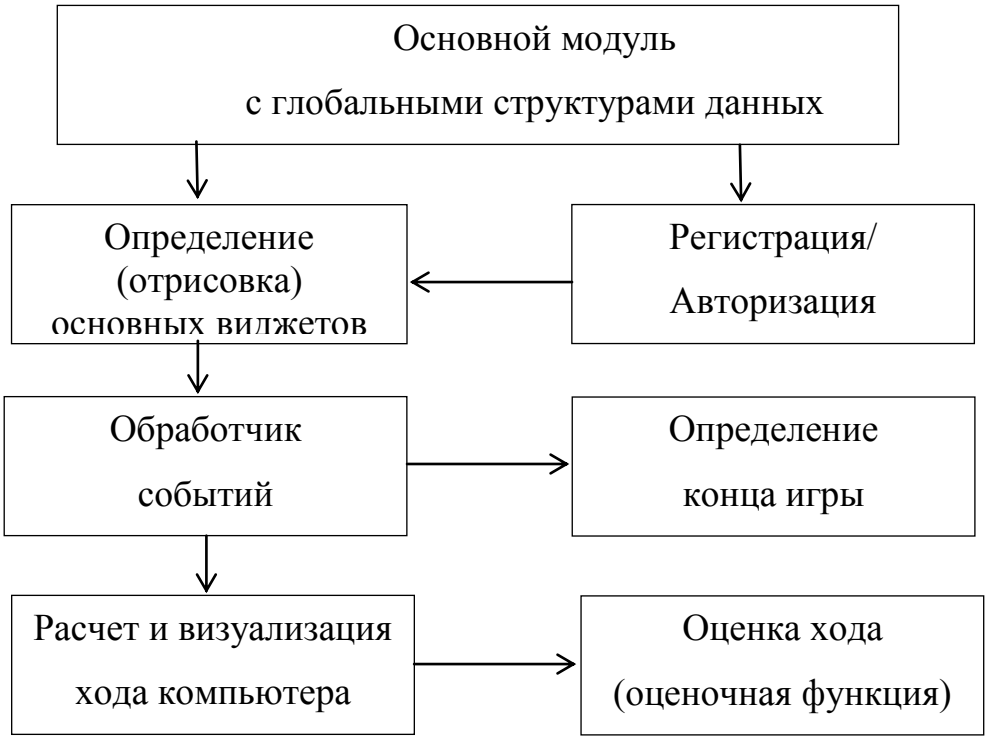

Рисунок 3. Архитектура приложения «игра в уголки»

РАЗРАБОТКА ПРИЛОЖЕНИЯ С ГРАФИЧЕСКИМ ИНТЕРФЕЙСОМ «ИГРА «УГОЛКИ» С ПОМОЩЬЮ БИБЛИОТЕКИ TKINTER

Краткая характеристика графической библиотеки tkinter

Для языка Питон известно более 10 подходов построения графического интерфейса. Наиболее популярными подходами являются:

– на основе библиотеки tkinter;

– на основе пакета Pygame;

– на основе библиотеки Qt.

Есть варианты применения их комбинаций. Любая из графических библиотек решает все основные задачи построения графического

интерфейса, однако что-то она делает лучше, а что-то хуже, чем другие. Так библиотека Pygame наиболее эффективна при разработке сложных графических игр, т. к. обладает мощными средствами управления графикой, анимацией и звуком.

В данном учебном пособии рассмотрим основы работы с объектами и функциями библиотек tkinter, которых будет достаточно для создания логических игр с графическим интерфейсом. Теоретический материал будет подкреплен примерами.

Библиотека tkinter входит в стандартный дистрибутив Питона и является базовым средством создания GUI приложений.

Для использования библиотеки tkinter она должна быть подключена командой из листинга 1 или из листинга 2.

Листинг 1

```
import tkinter
```

Листинг 2

```
from tkinter import *
```

Второй вариант предпочтительней. Для Питона версий ниже 3.0 название библиотеки должно быть с заглавной буквы.

По умолчанию приложение, созданное с помощью tkinter, имеет только одно графическое окно, которое представляет класс tkinter.Tk. Запуск приложения приводит к запуску этого главного графического окна, в рамках которого размещаются все остальные виджеты. Закрытие главного окна приводит к завершению работы приложения. Однако в рамках главного окна также можно запускать вторичные, неглавные окна.

Главное графическое окно задается командой из листинга 3

Листинг 3

```
root = Tk()
```

Список доступных виджетов (классов) библиотеки представлен в таблице 1.

Таблица 1

Классы виджетов

Название	Описание
Canvas	основа для вывода графических примитивов
Button	кнопка (может быть нажата)
Checkbutton	кнопка-флажок (выбор нескольких)
Radiobutton	кнопка-переключатель (выбор одного из нескольких)
Entry	позволяет пользователю ввести одну строку текста. Имеет дополнительное свойство bd (сокращенно от borderwidth), позволяющее регулировать ширину границы
Label	метка – предназначена для отображения какой-либо надписи (строчки) без возможности редактирования пользователем
Text	многостраничный редактор текста – позволяет пользователю ввести любое количество текста
Listbox	список-меню, из которого пользователь может выбирать один или несколько пунктов
Spintbox	список-меню с двумя стрелками прокрутки, из которого пользователь может выбирать один или несколько пунктов
ScrolledText	многостраничный редактор текста с возможностью прокрутки
Frame	рамка, внутри которой можно размещать другие виджеты
Labelframe	рамка с надписью для группировки виджетов
Message	надпись из нескольких текстовых строк
Panedwindow	окно из нескольких зон с подвижными разделителями
Scale	шкала с ползунковым регулятором
Scrollbar	полоса прокрутки
Menu	меню и подменю в главном меню
Menubutton	кнопка, имитирующая выпадающее меню класса Menu
OptionMenu	выпадающее меню

С помощью данных виджетов может быть реализован любой графический интерфейс.

Принципиальным для реализации графического интерфейса элементом библиотеки tkinter являются менеджеры компоновки. С их помощью производится размещение виджетов внутри объемлющего их виджета. Объемлющий виджет передается функции, создающей виджет, в качестве первого позиционного параметра. Если он не задан, то виджет размещается непосредственно в корневом виджете. В библиотеке tkinter реализовано 3 менеджера компоновки:

-pack – размещает виджеты в порядке вызова их методов;

-grid – размещает подчиненные виджеты в ячейки таблицы, представляющей структуру объемлющего виджета;

-place – размещает виджеты по явно указанным координатам.

Каждый из виджетов и методов компоновки обладает своими атрибутами (параметрами). С полным списком атрибутов можно ознакомиться в документации на официальном сайте библиотеки [12].

Мы будем рассматривать только те атрибуты, которые нам будут нужны для реализации примеров.

Разработка основных функций приложения

Рассмотрим разработку нашего приложения по игре в уголки. Так как приложение не является очень сложным, то дальнейшее проектирование отдельно можно не проводить, а совместить его с кодированием. То есть проводить проектирование непосредственно в среде программирования в соответствии с одной из версий гибкой разработки. Это нам позволяет разработанная, достаточно детализированная архитектура приложения, из которой виден перечень функций, требующих реализации, их спецификации и взаимодействие. Проектирование, совмещенное с

кодированием, будем проводить по методу нисходящего проектирования, известного также как метод пошаговой детализации [13]. Суть этого метода заключается в том, что на начальном шаге в соответствии с функциональным назначением приложения разрабатываются необходимые модели и архитектура приложения в форме структуры взаимосвязанных функций и общий алгоритм приложения. При этом реализация отдельных функций не детализируется. Детальная разработка функций проводится далее по ходу разработки приложения. Так как модели и архитектуру мы разработали, то следующим шагом разработаем скрипт (алгоритм) основного модуля, который представим в листинге 4, а результат его работы на рисунке 4.

Прокомментируем представленный скрипт. После подключения библиотеки tkinter, объявляем необходимые нам для работы приложения функции. Их объявляем в виде пустых функций (заглушек), – только название и команда pass, которая используется для задания пустых функций. Далее в процессе разработки пустые функции будем наполнять конкретной функциональностью в соответствии с их назначением.

Далее задаем размер клетки игрового поля (К). Следующими тремя командами определяем основное графическое окно (root), его размер и положение, заголовок. Следующими двумя строками формируем два графических объекта, в которых находятся изображения шашек для последующей визуализации.

Следующим блоком определяем основные глобальные структуры данных, нужные нам для работы приложения: двумерный массив для моделирования игрового поля, вспомогательный двумерный массив для расчета стоимости позиции и ряд переменных, смысл которых будет пояснен в дальнейшем.

Листинг 4

```
import tkinter as tk
from tkinter import *
import os
def regis():# регистрация пользователя
   pass
def eog(): # конец игры
   pass
def board_dr():# отрисовка поля
   pass
def score(pole1): # расчет стоимости позиции
   pass
def black_go(): # расчет хода компьютера
   pass
def click_button(i): # обработчик событий
   pass
root = Tk()
k = 100  # размер клетки
root.geometry("840x940+500+50")
root.title('Курсовая Работа: У Г О Л К И') #заголовок окна
pixel = tk.PhotoImage(width=1, height=1)
im1, im3= PhotoImage(file="res\\1b.gif"), PhotoImage(file="res\\1h.gif")
pole_sc = [[7, 6, 5, 4, 3, 2, 1, 0],
          [8, 7, 6, 5, 4, 3, 2, 1],
          [9, 8, 7, 6, 5, 4, 3, 2],
          [10, 9, 8, 7, 6, 5, 4, 3],
          [11, 10, 9, 8, 7, 6, 5, 4],
          [12, 11, 10, 9, 8, 7, 6, 5],
          [13, 12, 11, 10, 9, 8, 7, 6],
          [14, 13, 12, 11, 10, 9, 8, 7]]
pole = [[0, 0, 0, 0, 0, 1, 1, 1],
        [0, 0, 0, 0, 0, 1, 1, 1],
        [0, 0, 0, 0, 0, 1, 1, 1],
        [0, 0, 0, 0, 0, 0, 0, 0],
        [0, 0, 0, 0, 0, 0, 0, 0],
        [-1, -1, -1, 0, 0, 0, 0, 0],
        [-1, -1, -1, 0, 0, 0, 0, 0],
        [-1, -1, -1, 0, 0, 0, 0, 0]]
num, sc, ngo, x1, y1, x2, y2 = 0, 0, 1, 0,0,0,0
end_game, fl_game, wh  = False, False, True
s_l, s_p, kl = 'name','pas',[]
txt = Label(root, text="Для игры введите  Ваш логин и пароль\nДля регистрации введите  новый
логин и пароль",font=("Arial", 14,'bold'))
txt.place(x=200,y=200)
txtl = Label(root, text='Логин',font=("Arial", 14,'bold'))
txtl.place(x=340, y=260)
login = tk.Entry(root,width=10,bd=3, textvariable =s_l)
login.place(x=420, y=260)
txtp = Label(root, text='Пароль',font=("Arial", 14,'bold'))
txtp.place(x=340, y=300)
password = tk.Entry(root,width=10,bd=3, textvariable =s_p)
password.place(x=420, y=300)
close_but = tk.Button(root, text="Зарегистрироваться / начать игру", command= regis)
close_but.place(x=315, y=340)
lab = Label(root, text="" ,font=("Arial", 14,'bold'))
lab.grid(row=8, column=3)
board_dr()
mainloop()
```

Следующий блок – это разработка содержимого первого окна – окна регистрации. Сначала с помощью виджета метка (Label) создаем информационное сообщение. Для этого используем параметры: имя окна, содержимое текстового поля и характеристики шрифта. Созданное информационное сообщение располагаем с помощью менеджера компоновки place по абсолютным координатам (x=200, y=200).

Далее формируем два информационных сообщения (Label) и два поля ввода (Entry) для получения от пользователя значений логина и пароля.

Для полей ввода используем следующие параметры: имя окна, ширина поля ввода в символах и ширина границы. Заданные графические объекты также располагаем с помощью менеджера компоновки place по абсолютным координатам.

Далее задаем кнопку «зарегистрироваться / начать игру».

Ее параметры: имя окна, текст на кнопке, функция, которая будет вызываться по нажатию кнопки. Имя функции задается через именованный параметр command. Для многих виджетов в библиотеке имеются типовые реакции, с которыми можно ознакомиться в документации на библиотеку. В случае если типовая реакция на устраивает разработчика (как в нашем случае), можно написать свою функцию реакции на событие и связать ее имя с данным виджетом через параметр command. Созданную кнопку также располагаем по абсолютным координатам.

В примере использовался менеджер компоновки place, который разместит все виджеты по явно указанным координатам внутри главного окна. Можно было организовать данный графический интерфейс с помощью другого менеджера. Не рекомендуется одновременно пользоваться несколькими менеджерами внутри одного окна.

Далее внизу основного графического окна формируем информационное поле для сообщений пользователю. Для его размещения

используем менеджер компоновки grid. Это сделано потому, что все виджеты, размещенные менеджером компоновки place в основном игровом цикле, будут удалены, а при отрисовке игрового поля более эффективным менеджером будет grid.

И последней командой в данном примере запускаем бесконечный цикл работы нашего графического приложения.

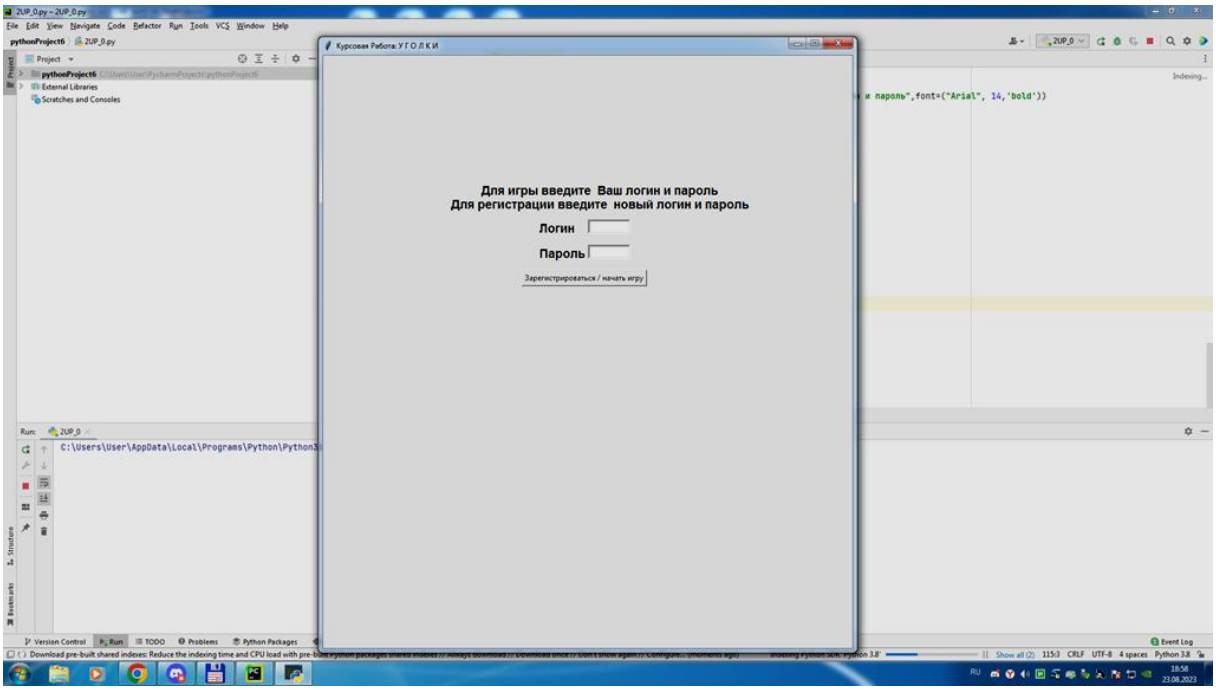

Рисунок 4. Скриншот экрана с сформированным графическим интерфейсом начального окна

После проверки дизайна и правильности работы основного графического окна переходим к последовательной разработке необходимых функций в соответствии с ТЗ и разработанной архитектурой.

Начнем с функции «Регистрации / Авторизация». Начальное окно данной функции уже реализовано в основном модуле (листинг 4, рисунок 4). При нажатии кнопки «Зарегистрироваться / начать игру» должна вызваться функция regis.

В простейшем варианте функция должна взять данные из полей ввода логина и пароля начального окна, проверить их на пустые данные и, в случае наличия информации, проверить ее совпадение с информацией в файле логинов. При отсутствии совпадения завести в файле нового игрока. Функция должна информировать пользователя о результатах своей работы с помощью дополнительных информационных окон. Скрипт данной функции представлен в листинге 5, а скриншот ее работы (одно из окон) – на рисунке 5. Остальные окна будут выглядеть аналогично

Кратко прокомментируем скрипт данной функции. В первой строчке определяем глобальные структуры данных, инициированные в основном модуле, которые будут необходимы для работы данной функции. Далее задается функция dismiss, смысл которой поясним чуть позже.

Далее читаем данные из полей ввода login и password. Если хотя бы в одном поле данные отсутствуют, то перехватываем событие «нажатие на крестик» и формируем новое информационное окно и захватываем на него пользовательский ввод. Теперь, что бы пользователь ни пытался сделать в основном окне, приложение его проигнорирует.

Выход из этого окна производится по нажатию на кнопку close_button, с которой связана функция dismiss, которая вернет управление в основное окно и уничтожит новое информационное окно.

В случае наличия информации в обоих полях ввода открываем файл l_p.txt, который должен находиться в каталоге проекта, и проверяем наличие в нем введенного логина и пароля. В случае их отсутствия считаем, что это новый пользователь, и регистрируем его. Для этого указатель файла ставим в конец и дописываем в конец файла введенный логин и пароль. В обоих случаях информируем пользователя о сложившейся ситуации с помощью нового информационного окна (аналогично ситуации с пустым полем ввода). После удачной регистрации/авторизации последним блоком стираем все виджеты

регистрации, информируем пользователя внизу после игрового поля о дальнейших действиях и рисуем игровое поле.

Листинг 5

```
def regis():
    global txt,txtl,txtp, login, password, lab
    def dismiss(win_t):
        win_t.grab_release()
        win_t.destroy()

    s_l  = login.get()
    s_p  = password.get()
    if len(s_l) == 0 or len(s_p) == 0:
        win = Toplevel(root, relief=SUNKEN)
        win.geometry("400x100+730+420")
        win.title("Регистрация / Авторизация")
        win.minsize(width=400, height=100)
        win.maxsize(width=400, height=100)
        win.protocol("WM_DELETE_WINDOW", lambda: dismiss(win))          # перехватываем нажатие
на крестик
        Label(win, text='Пустое поле "Логин" или "Пароль"',font=("Arial", 14,'bold')).place(x=30, y=10)
        close_button = tk.Button(win, text="Повторить ввод", command=lambda: dismiss(win))
        close_button.place(x=150, y=50)
        win.grab_set()      # захватываем пользовательский ввод
    else:
        f_reg =False
        file = open("l_p.txt", "r+")    # открываем файл
        a = file.read().split()         # читаем файл
        for j in range(len(a)):         # ищем совпадение логин и пароль
            if a[j] == s_l and a[j+1] == s_p:
                f_reg = True
                break
        if not f_reg:                   # совпадения нет
            file.seek(0, os.SEEK_END)
            file.write(s_l+' '+s_p+' ')  # записываем новые логин и пароль
        file.close()
        win_r = Toplevel(root, relief=SUNKEN)
        win_r.geometry("400x100+730+420")
        win_r.title("Регистрация / Авторизация")
        win_r.minsize(width=400, height=100)
        win_r.maxsize(width=400, height=100)
        win_r.protocol("WM_DELETE_WINDOW", lambda: dismiss(win_r))          # перехватываем
нажатие на крестик
        Label(win_r, text="Уважаемый(-ая) "+s_l+", вы успешно"+"\n зарегистрировались /
авторизовались",font=("Arial", 14,'bold')).place(x=5, y=10)
        close_button = tk.Button(win_r, text="Начать игру", command=lambda: dismiss(win_r))
        close_button.place(x=150, y=65)
        win_r.grab_set()      # захватываем пользовательский ввод

        txt.place_forget()  # стираем виджеты регистрации
        txtl.place_forget()
        txtp.place_forget()
        login.place_forget()
        password.place_forget()
        close_but.place_forget()
        lab['text'] = 'выбор \n шашки'
        board_dr()          # рисуем поле
```

В функции регистрации не рассмотрена функциональность шифрования логина и пароля (заявленная в ТЗ). Это сделано по двум причинам. Во-первых, функция шифрования с точки зрения цели приложения является вспомогательной и принципиально не влияет на его работу. А ее изложение будет занимать место и отвлекать внимание от принципиальных особенностей реализации приложения. Во-вторых, алгоритмы шифрования достаточно подробно рассматриваются на лекциях и отрабатываются на лабораторных работах. Поэтому студентам не составит особенного труда реализовать функцию шифрования самостоятельно и включить ее в функцию регистрации, пропустив через нее введенный логин и пароль. После этого все логины и пароли в приложении будут использоваться только в зашифрованном виде.

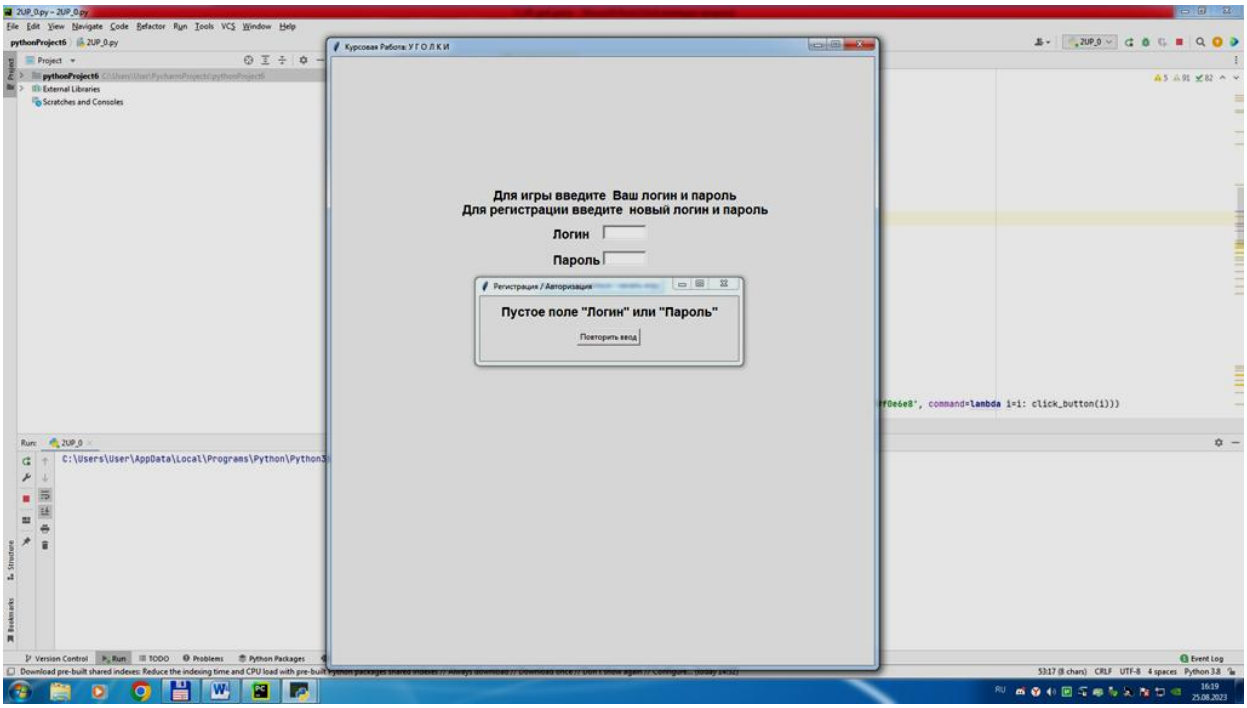

Рисунок 5. Скриншот экрана с дополнительным информационным окном

После проверки дизайна и правильности работы функции регистрации в рамках основного модуля переходим к разработке следующей функции в соответствии с ТЗ и разработанной архитектурой.

По логике разработки такой функцией является функция «Отрисовки игрового поля» board_dr. Данная функция вызывается последней строчкой в функции регистрации. Скрипт функции отрисовки игрового поля представлен в листинге 6, а начальный скриншот ее работы – на рисунке 6.

Кратко прокомментируем данный скрипт. В начале функции объявляем 3 глобальных структуры данных:

kl – список кнопок , который является графической моделью игрового поля;

im1 и im3 – переменные, хранящие изображения шашек, которые были считаны из графических файлов в основном модуле.

Далее инициализируем список кнопок kl и в двойном вложенном цикле, на основе модели игрового поля pole, с помощью метода append заполняем его. При этом используются следующие атрибуты виджета кнопка:

- Первый неименованный атрибут – имя родительского виджета;
- Image – ссылка на изображение;
- Text – ссылка на текст, который отображается на кнопке;
- height, width – высота и ширина виджета (количество букв/строк для текстовых кнопок или количеством пикселей для изображений);
- compound – расположение картинки на кнопке;
- bg – цвет фона кнопки;
- command – вызываемая при нажатии на кнопку функция или метод.

Для правильного формирования графической модели поля kl она связана с моделью игрового поля pole через переменные i, i1, j1. Номер кнопки i пересчитывается из номера строки (i1) и столбца (j1) массива pole. Как видно из скрипта, графические кнопки формируются трех типов: для пустой клетки, для клетки с белой шашкой и для клетки с черной шашкой. При этом цвет клетки у всех один и тот же, что определяется атрибутом bg.

В ТЗ не заявлено, что игра должна вестись на шахматной доске (клетки разных цветов). Если бы данное требование в ТЗ присутствовало, то его можно было бы легко реализовать. В этом случае в двойном цикле было бы не три условия, а шесть, чтобы учитывать тип клетки (черная/белая), что легко определяется через ее номер строки и номер столбца.

Листинг 6

```
def board_dr():
    global kl, im1, im3
    kl = [] # рисуем доску
    for i1 in range(8):
        for j1 in range(8):
            i = i1*8 + j1
            if pole [i1][j1] == 1:
                kl.append(Button(root,image=im3,text="",height=k,width=k,
                        compound="c",command=lambda i=i: click_button(i)))
            elif pole [i1][j1] == -1:
                kl.append(Button(root,image=im1,text="",height=k,width=k,
                        compound="c",command=lambda i=i: click_button(i),
                        activebackground = 'red'))
            else:
                kl.append(Button(root,image=pixel,text="",height=k,width=k,
                        compound="c",command=lambda i=i: click_button(i)))
    for i, x in enumerate(kl):
        x.grid(column=i%8, row=(i//8), sticky=NW)
```

После формирования списка кнопок для изображения клеток игрового поля их размещаем в основном графическом окне с помощью менеджера размещения grid, который будет размещать кнопки последовательно по соответствующим строкам и столбцам. Это делаем в цикле for, используя функцию enumerate, которая позволяет обрабатывать одновременно и индекс элемента списка, и его значение.

Следующей функцией для разработки будет функция «Обработчик событий». Надо определить, какие события для нас будут принципиальны. Это два события: закрытие приложения и отработка хода пользователя. Закрытие приложения осуществляется по нажатию на крестик в заголовке окна. Данное событие является стандартным. Функция-обработчик

закрытия окна присутствует в графической библиотеке и поэтому не требует разработки. Она подключается автоматически при создании окна.

Для второго события в нашем случае стандартного обработчика нет – его нужно разрабатывать. Обработчик должен взять информацию о ходе пользователя, отобразить его в модели поля игры, визуализировать его на экране дисплея, рассчитать и выдать ответный ход компьютера. При этом при необходимости он должен выдавать пользователю информацию о процессе игры.

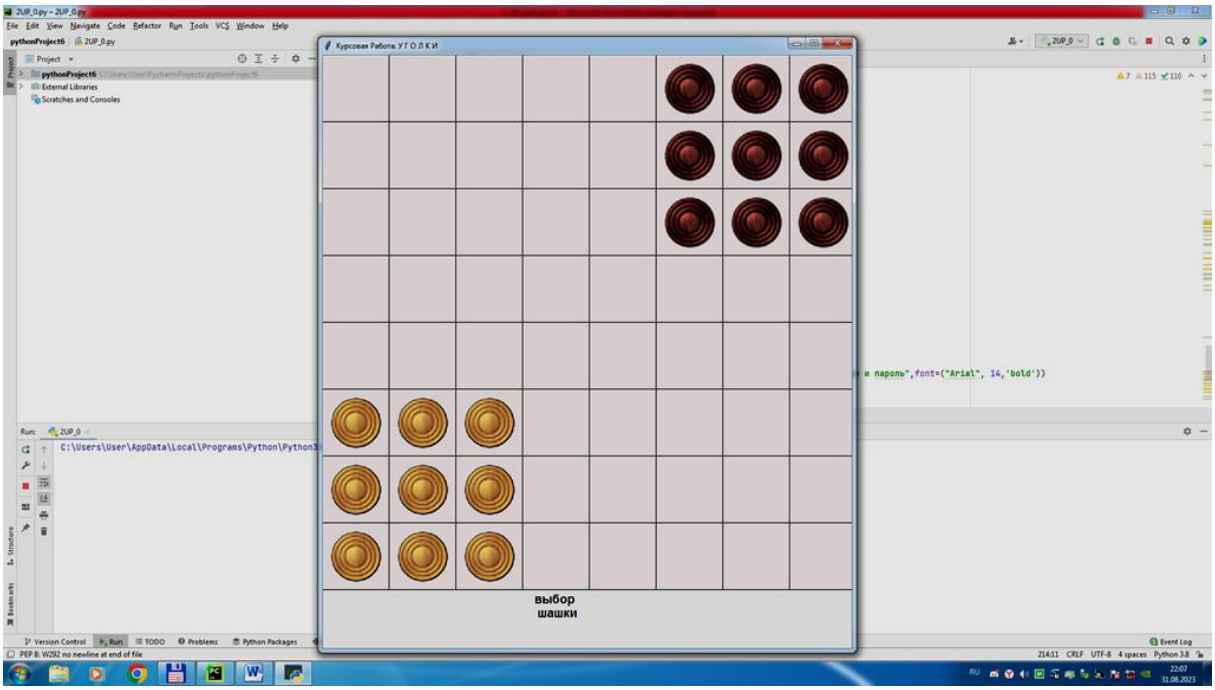

Рисунок 6. Скриншот экрана с игровым полем

Прежде чем начать разрабатывать функцию обработки событий, нужно принять ряд проектных решений: как пользователь будет информировать приложение о своем ходе; как визуализировать процесс хода; как отображать дополнительную информация для пользователя. По этим вопросам могут быть разные варианты реализации, которые будут существенно различаться как сложностью алгоритмов, так и визуальным представлением. Поэтому данные проектные решения лучше согласовать с

заказчиком, т. к. у него может быть свое мнение по этим вопросам. И во время сдачи приложения он может не согласиться с вашими решениями, и вам придется все переделывать. Мы принимаем следующие проектные решения:

- для осуществления хода пользователь левой кнопкой мыши щелкает на шашке, которой собирается ходить, и следующим действием также определяет конечную клетку, в которую он хочет переместить шашку;
- для визуализации хода клетка с активной шашкой после ее выбора меняет цвет фона, а после выбора конечной клетки активная шашка визуализируется в конечной клетке, при этом начальная клетка визуализируется как пустая с нормальным фоном;
- для информирования пользователя о необходимых действиях зададим специальное окно ниже игрового поля, что соответствует требованиям ТЗ.

Из принятых проектных решений следует, что обработчик должен реагировать на два события – выбор активной шашки и выбор клетки для ее перемещения. То есть ход как бы разделяется на две фазы. И вторая фаза (и соответственно второе событие) возможна только после правильной отработки первой. Для отслеживания этого (управления обработкой) в приложение введем глобальную переменную ngo. При этом следует учитывать, что ходы бывают двух видов: простой – перемещение шашки на соседнюю свободную клетку и сложный – с «прыжками» через занятые клетки. Если алгоритм для хода первого типа достаточно простой и не предусматривает особого разнообразия, то для визуализации хода второго типа возможны как минимум два варианта. Первый вариант – сложный ход разбивается на шаги (одиночные «прыжки») и последовательно реализуется с пошаговой отрисовкой. Второй вариант – для хода показываются только начальная и конечная клетки. Их алгоритмическая реализация будет отличаться не только алгоритмами

визуализации, но и алгоритмами проверки правильности ходов. В первом варианте нужно проверять в цикле только правильность одного прыжка и то, что пользователь в сложном ходе не использует простой ход. Во втором варианте нужно найти правильный путь по пустым клеткам между начальной и конечной клетками. Как видно из проектных решений, выбран второй вариант. В этом случае алгоритмически для поиска пути нам нужно решить задачу построения пути между двумя вершинами в графе, который ищется при построении (обходе) дерева, корнем которого является клетка с активной шашкой. Поскольку у активной шашки возможно множество ходов и при сложном ходе шашка может остановиться в любой клетке, получаем достаточно разветвленное дерево ходов активной шашки. Существуют два основных алгоритма построения (и обхода) деревьев: перебор в ширину и перебор в глубину.

Как правило, первый алгоритм реализуют итерационно, второй – рекурсивно. Рассмотрим оба варианта.

Скрипт функции обработчика событий clikc_button с итерационным алгоритмом представлен в листинге 7, а скриншот одного из экранов его работы – на рисунке 7. Кратко прокомментируем его.

Данная функция динамически строит дерево возможных ходов активной шашки, проверяя при этом, является ли новая вершина конечной клеткой хода. В случае успешной проверки построение останавливается, и ход признается допустимым. В случае перебора всех вершин дерева и отрицательной проверки ход признается недопустимым.

В качестве параметра функция получает номер кнопки в списке kl, нажатие на которую требует обработки. Вначале идет описание глобальных структур данных, которые потребуются нам в данной функции. Далее сбрасываем два флага – для сложного и простого ходов.

Листинг 7

```
def click_button(i):
    global wh,ngo,x1,x2,y1,y2,end_go, num
    global pole,kl,lab
    smp_go, cmx_go = False, False
    if ngo == 1: # выбор шашки для хода
        x1 = i // 8
        y1 = i % 8
        if pole[x1][y1] == -1 and kl[i]["state"] == tk.NORMAL:
            kl[i].config(state=tk.NORMAL, bg="red")
            ngo = 2
            lab['text'] = 'выберите\n поле для\n шашки '
    else: # выбор/ проверка конечной позиции хода
        x2 = i // 8
        y2 = i % 8
        if pole [x2][y2] == 0 and kl[i]["state"] == tk.NORMAL:
            if abs(x2-x1)==1 and y2==y1 or abs(y2-y1)==1 and x2==x1:#np.x
                smp_go = True
            else:                                   #сл. ход
                lst_go = [[x1,y1]]
                while len(lst_go) > 0 and not cmx_go:
                    if  lst_go[0][0] == x2 and lst_go[0][1] == y2:
                        cmx_go =  True
                    else:
                        x = lst_go[0][0]
                        y = lst_go[0][1]
                        for ix, iy in (0, -2), (-2, 0), (0, 2), (2,0):#прыжки
                            if (0 <= y + iy <= 7 and 0 <= x + ix <= 7) and \
                            (abs(pole[x + ix // 2][y + iy // 2]) == 1) and \
                                (pole[x + ix][y + iy] == 0) and \
                                not [x+ix,y+iy] in lst_go:
                                lst_go.append([x + ix, y + iy])
                        del lst_go[0]
            ngo = 1
            if cmx_go or smp_go:
                pole[x1][y1], pole[x2][y2] = 0, -1
                black_go()
                eog()
                num += 1
            else:
                lab['text'] = 'невер ход\n выберите\n шашку'
        else:
            ngo = 1
            lab['text'] = 'невер ход\n выберите\n шашку'
        board_dr()
```

Скрипт должен отрабатывать нажатие на любую кнопку (клетку), а реакция на них должна быть разная. Поэтому мы должны выяснить, какое событие требует реакции, и запустить на выполнение соответствующий участок алгоритма.

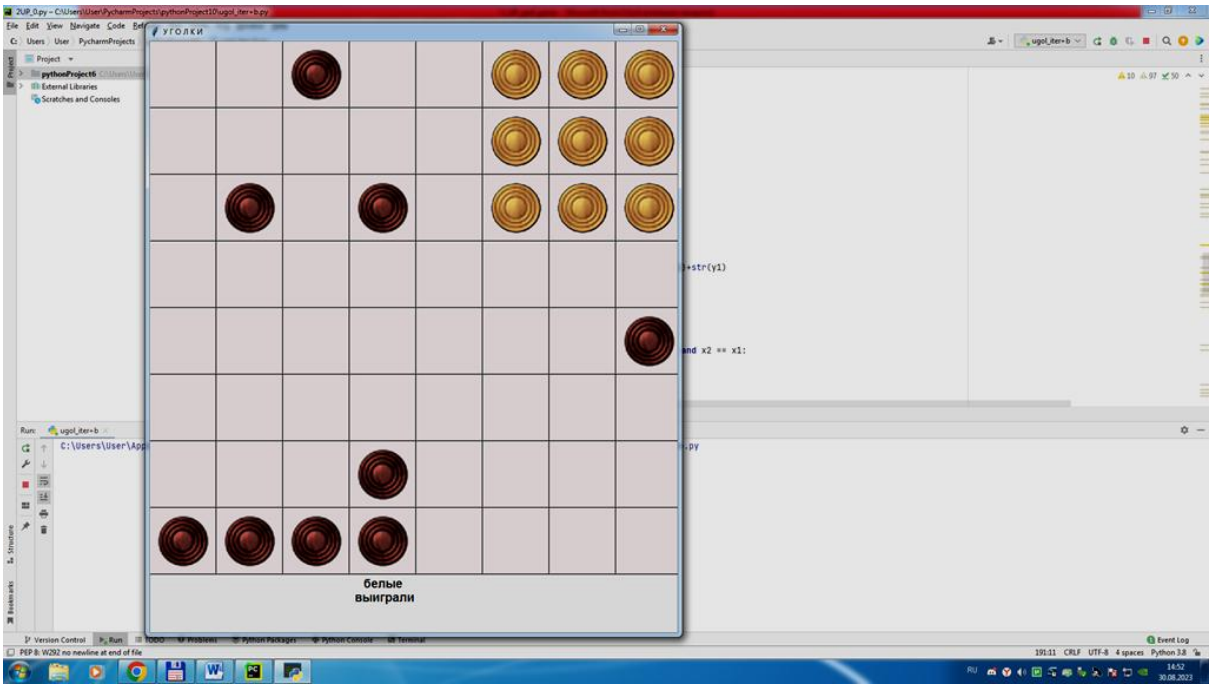

Рисунок 7. Скриншот экрана с игровым полем

Сначала определяем, является ли событие выбором шашки для хода – первая фаза хода (ngo = 1). В этом случае определяем координаты шашки на игровом поле, высчитывая их из номера кнопки, с которой связана эта шашка. Далее проверяем, что это клетка с белой шашкой и соответствующая ей кнопка находится в нормальном состоянии (не нажата). При этом меняем цвет фона клетки (кнопки), не меняя статуса кнопки. Далее информируем пользователя о необходимости выбора конечной клетки для завершения хода и устанавливаем значение ngo = 2 (вторая фаза хода). После чего данная ветка алгоритма закачивается, функция передает управление в основной бесконечный цикл, и приложение ждет нового события (нажатия на кнопку).

Теперь при нажатии на любую клетку с шашкой обработчик не будет реагировать, т. к. установлен флаг ngo = 2, а в этом случае алгоритм будет реагировать только на клетки, не занятые шашками. При нажатии на

пустую клетку функция преобразует номер кнопки в номер строки и столбца и проверяет, является ли данный ход простым. В этом случае устанавливается флаг простого хода. В противном случае осуществляется динамическое построение дерева возможных сложных ходов активной шашки с одновременной проверкой на правильность сложного хода. То есть проверяем, находится ли сделанный пользователем ход в полном дереве возможных сложных ходов данной шашки. Пример фрагмента дерева ходов (дерева игры) представлен на рисунке 8. Алгоритмически это решается полным перебором дерева ходов (в ширину или в глубину).

Сначала рассмотрим итерационный алгоритм обхода (построения) дерева в ширину, который начинается с 19 строки скрипта. Сначала задаем вспомогательный список lst_go, в котором будем хранить динамически формируемое дерево возможных сложных ходов активной шашки для исключения зацикливания, поместив в него координаты активной шашки.

Алгоритм реализуется циклом while, пока не закончился список вершин дерева и не установлен флаг правильности сложного хода. Алгоритм достаточно прост. Вначале координаты активной шашки заносятся в список вершин, и далее в цикле для очередной вершины все возможные ходы из нее также записываются в список вершин. Таким образом, реализуется перебор в ширину.

При этом вершина записывается в список, если она проходит проверку на правильность хода и отсутствует в списке. После обработки текущая вершина из списка удаляется. Процесс завершается либо при исчерпании списка, либо при совпадении координат текущей вершины координатам клетки, указанной пользователем. В любом случае вторая фаза хода считается завершенной (успешно или нет) и устанавливается первая фаза хода (флаг ngo = 1).

Далее проверяется, установлен ли флаг простого или сложного хода.

В случае установки ход отрабатывается в модели игрового поля pole, увеличивается счетчик ходов, информируется пользователь о выполнении хода и вызывается функция расчета хода компьютера. В противном случае информируется пользователь о том, что ход неверный и предлагается выбрать шашку (т. е. сделать новый ход).

Далее вызываются функции отрисовки поля и определения окончания игры. Код функции определения окончания игры eog представлен на Листинге 8. Алгоритм функции настолько прост, что не требует особых комментариев. Из листинга видно, что определяется флаг выигрыша белых и флаг выигрыша черных. Далее на основании их соотношения по таблице решений для пользователя выводится одно из четырех сообщений: ничья; белые выиграли; черные выиграли; игра не окончена.

Листинг 8

```
def eog():
    global pole,num,lab
    w_b, w_w = False,False
    if (pole[0][5] == -1 and pole[0][6] == -1 and pole[0][7] == -1 and
        pole[1][5] == -1 and pole[1][6] == -1 and pole[1][7] == -1 and
        pole[2][5] == -1 and pole[2][6] == -1 and pole[2][7] == -1) or (
       num==39) and (pole[0][5] == 1 or pole[0][6] == 1 or pole[0][7] == 1 or
                     pole[1][5] == 1 or pole[1][6] == 1 or pole[1][7] == 1 or
                     pole[2][5] == 1 or pole[2][6] == 1 or pole[2][7] == 1):
        w_w = True
    if (pole[5][0] == 1 and pole[5][1] == 1 and pole[5][2] == 1 and
        pole[6][0] == 1 and pole[6][1] == 1 and pole[6][2] == 1 and
        pole[7][0] == 1 and pole[7][1] == 1 and pole[7][2] == 1) or (
    num==39) and (pole[5][0]==-1 or pole[5][1]==-1 or pole[5][2]==-1 or
                  pole[6][0]==-1 or pole[6][1]==-1 or pole[6][2]==-1 or
                  pole[7][0]==-1 or pole[7][1]==-1 or pole[7][2]==-1):
        w_b = True
    if w_w:
        if w_b:
            lab['text'] = 'ничья'
        else:
            lab['text'] = 'белые\n выиграли\n'
    else:
        if w_b:
            lab['text'] = 'черные\n выиграли\n'
        else:
            lab['text'] = 'игра не\n  окончена\n'
```

Рассмотрим альтернативный вариант обработчика событий, заменив в нем проверку правильности сложного хода с итерационного алгоритма (поиск в ширину) на рекурсивный алгоритм (поиск в глубину). Итерационный алгоритм в листинге 7 представлен строками 19-32. Рекурсивный алгоритм, заменяющий эти строки, представлен в листинге 9 и обеспечивающая его рекурсивная функция в листинге 10. Для простоты сравнения кодов в листинге 9 возьмем по одной строке до и после изменяемого кода. Как видно в листинге 9, задаем пустой список для хранения рекурсивного дерева ходов vert_lst и вызываем рекурсивную функцию проверки правильности сложного хода. Для корректности работы данный список должен быть объявлен глобальным.

Листинг 9

```
else:                                   #сл. ход
    vert_lst = []
    test_cs_go(x1,y1,pole)
ngo = 1
```

Листинг 10

```
def test_cs_go(x,y,pole2):
    global kl,cmx_go,smp_go,x1,x2,y1,y2,vert_lst
    new_pole = pole2.copy()
    if not (x,y) in vert_lst:
        vert_lst.append((x,y))
    if cmx_go:
        return
    elif x == x2 and y == y2:
        cmx_go = True
        vert_lst = []
        return
    else:
        for ix, iy in (0, -2), (-2, 0), (0, 2), (2,0):
            if 0 <= y + iy <= 7 and 0 <= x + ix <= 7:
                if abs(pole2[x+ix//2][y+iy//2])==1 and pole2[x+ix][y+iy]==0: #прыжок
                        if not (x+ix, y+iy) in vert_lst:
                            test_cs_go(x + ix,y + iy,new_pole)
        return
```

Рекурсивная функция проверки правильности сложного хода (Листинг 10) получает на вход координаты активной шашки и матрицу

(модель игрового поля). В самой функции объявляются необходимые для работы глобальные структуры данных и создается копия матрицы игрового поля. Копия матрицы нужна для того, чтобы каждый вызов рекурсивной функции работал со своей матрицей игрового поля. Далее в случае отсутствия полученных координат клетки в списке ходов шашки они в него записываются. Проверка на отсутствие нужна для исключения зацикливания. После этого идет тройная проверка. Первый if проверяет установку флага правильности сложного хода. В случае установки происходит возврат в вызывающую функцию, что обеспечивает обратный ход рекурсии при успешной проверке хода. Второй if проверяет совпадение текущей клетки и конечной клетки сложного хода. При совпадении устанавливается флаг правильности сложного хода, очищается список ходов шашки и происходит возврат в вызывающую функцию, что запускает процесс обратного хода рекурсии. При отрицательных результатах этих проверок для текущей клетки рассматриваются все возможные допустимые ходы (прыжки) и для каждого запускается эта же рекурсивная функция. После перебора всех возможных ходов шашки происходит возврат в вызывающую функцию. Это обеспечивает корректную работу в случае ошибочного хода, т. е. проверки на того, что указанная клетка не может быть достигнута из клетки активной шашки.

Осталось рассмотреть функцию вычисления хода компьютера. В силу ее сложности и необходимости рассмотрения ряда специальных алгоритмов мы будем рассматривать ее в следующем параграфе.

Разработка игрового «интеллекта» приложения

С точки зрения теории игр в игре «уголки» принимают участие два игрока: человек и компьютер. Целью игры каждого игрока является более быстрое, чем соперник, занятие своими шашками дома соперника. Данная

игра относится к играм с нулевой суммой, т. е. выигрыш одного игрока равняется проигрышу другого.

Для подобных игр важным является выбор правильной стратегии игры, т. е. последовательности ходов, приводящей к своему выигрышу и соответственно проигрышу противника. Информация по вариантам выбора игровой стратегии (процедура расчета очередного хода) представлена в разделе «Элементы теории игр». В учебном пособии рассмотрим третий вариант расчета стратегии игры компьютера, т. е. алгоритм МиниМакса для выбора очередного хода.

Алгоритм МиниМакса основывается на дереве игры (рисунок 8).

Как видно из рисунка 8, у игрока имеется некоторая позиция, на основании которой он может сделать один из n ходов. Тогда на каждый возможный ход игрока компьютер может ответить одним из своих k или t возможных ходов. И так далее. Дерево игры строится на определенную глубину просчета ходов.

Суть алгоритма МиниМакса заключается в том, что он для себя выбирает лучший ход (с наибольшим счетом), а для противника наихудший для него ход (с наименьшим счетом). И так последовательно на необходимую глубину расчета.

Рассмотрим данный алгоритм с глубиной 1, т. е. некоторую упрощенную версию. В упрощенном виде алгоритм будет выбирать лучший ход, основываясь на текущей ситуации на игровом поле, т. е. не просчитывая ходы в глубину.

В этом случае алгоритму не надо строить все дерево игры. Достаточно построить всего одну ветку, начиная с уже сделанного хода противника и перебрав все свои возможные ходы в текущей ситуации. В силу того, что у шашки может быть множество ходов и сложные ходы могут сильно разветвляться, получается, что нужно построить (рассмотреть) достаточно разветвленное дерево ходов.

Данный алгоритм будет хорошо играть в тактическом плане – он выбирает лучший текущий ход, но абсолютно не просчитывает ход игры стратегически. Поэтому такой алгоритм проиграет даже среднему игроку.

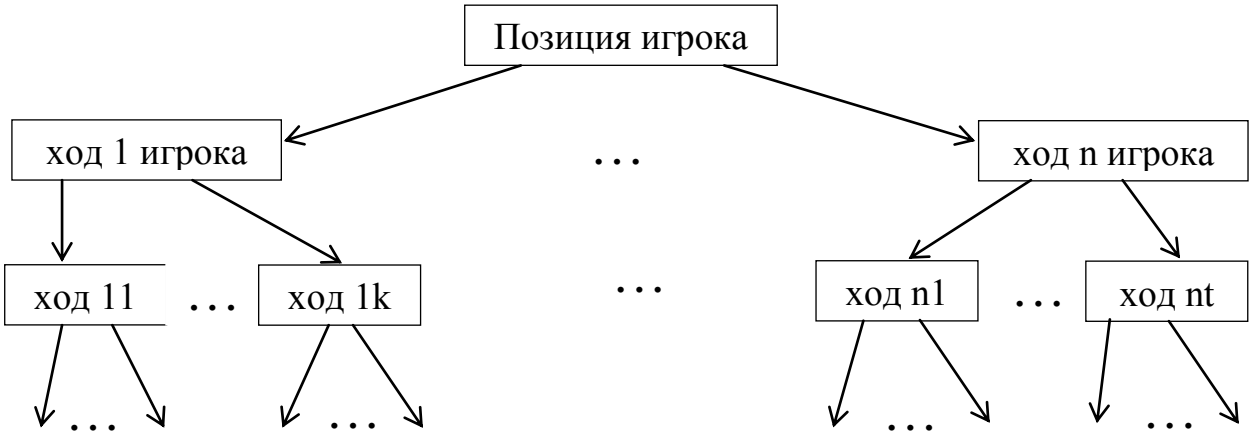

Рисунок 8. Дерево игры

Поскольку данный алгоритм основывается на анализе динамически построенного дерева ходов, то в простейшем варианте видна его аналогия с алгоритмом проверки правильности хода. И тот и другой должны динамически обойти построенную ветку дерева. Только с той разницей, что один должен найти в ней совпадение с конкретной вершиной, а другой найти вершину с лучшей стоимостью (лучший ход). Как мы уже видели, дерево можно обходить в ширину (итерационно) или в глубину (рекурсивно).

Для начала рассмотрим итерационный алгоритм расчета хода компьютера с глубиной просмотра, равной 1. Листинг алгоритма расчета хода компьютера приведен в листинге 11. Прокомментируем данный программный код. Сначала мы объявляем глобальные структуры данных, которые нам будут нужны при расчете хода. Далее устанавливаем x координату черной шашки, которой компьютер будет ходить, равной «-1».

Это позволит не вводить дополнительный флаг того, что ход не найден, т. к. реальные координаты не могут быть отрицательными.

Листинг 11

```
def black_go():
    global sc,pole_sc,pole
    xb1 = -1
    sc_bl = score() # начальный счет
    for x in range(8): # перебор всех черных шашек
        for y in range(8):
            if pole[x][y] == 1:
                for ix, iy in (0, -1), (1, 0), (0, 1), (-1, 0): # прост. ходы
                    if 0 <= y + iy <= 7 and 0 <= x + ix <= 7:
                        if pole[x + ix][y + iy] == 0:
                            pole[x][y], pole[x + ix][y + iy] = 0, 1 # ход
                            sc_t = score()                  # счет хода
                            pole[x][y], pole[x + ix][y + iy] = 1, 0 # откат
                            if sc_t >= sc_bl:          # сохр. лучшего хода
                                sc_bl = sc_t
                                xb1, yb1= x, y
                                xb2, yb2= x + ix, y + iy
                lst_bl_go = []
                lst_bl_go.append([x, y])
                i_cx = 0
                while len(lst_bl_go) > i_cx:
                    xs = lst_bl_go[i_cx][0]
                    ys = lst_bl_go[i_cx][1]
                    for ix, iy in (0, -2), (2, 0), (0, 2), (-2,0):
                        if 0 <= ys + iy <= 7 and 0 <= xs + ix <= 7:
                            if abs(pole[xs + ix // 2][ys + iy // 2]) == 1 and pole[xs + ix][ys + iy] == 0:
                                if not [xs + ix, ys + iy] in lst_bl_go:
                                    lst_bl_go.append([xs + ix, ys + iy])
                                    xb2_cx, yb2_cx= xs + ix, ys + iy
                                    pole[x][y], pole[xb2_cx][yb2_cx]=0,1 #ход
                                    sc_t = score()  # счет хода
                                    pole[x][y],pole[xb2_cx][yb2_cx]=1,0#откат
                                    if sc_t >= sc_bl:  # сохр. лучшего хода
                                        sc_bl = sc_t
                                        xb1, yb1 = x, y
                                        xb2, yb2 = xb2_cx, yb2_cx
                    i_cx += 1
    if xb1 == -1:
        lab['text'] = 'у черных\n нет\nходов'+str(sc_bl)+str(sc_t)
    else:
        pole[xb1][yb1], pole[xb2][yb2]= 0, 1
```

Далее вычисляем значение счета по текущей ситуации на игровом поле с помощью функции score() и двойным циклом перебираем все клетки на игровом поле. В данном двойном цикле выбираем только клетки с черными шашками (это обеспечивает первый if). Для каждой клетки с

черной шашкой проверяем ее на простые ходы, вычисляя счет хода и сохраняя лучший ход. Комментарии в скрипте детально объясняют данный процесс. После перебора всех возможных простых ходов данной шашки переходим к рассмотрению ее сложных ходов.

Для этого создаем список клеток, в котором последовательно сохраняются координаты клеток возможных «прыжков» активной шашки. Для начала итерационного процесса в список помещаем координаты активной клетки. Итерационный процесс реализуется циклом while до тех пор, пока не переберем все возможные клетки из динамически формируемого списка. В цикле берем очередную клетку возможного «прыжка» и для нее проверяем все возможные «прыжки» из нее. При этом помещаем возможный «прыжок» в формируемый список (если он там отсутствует) и рассчитываем счет данного возможного хода и сохраняем лучший ход. После обработки очередной клетки сдвигаем указатель списка, увеличивая его на единицу.

После перебора всех черных шашек проверяем, был ли рассчитан очередной ход. В случае отсутствия хода информируем об этом пользователя, а иначе реализуем этот ход в модели игрового поля.

Теперь рассмотрим альтернативный вариант расчета хода компьютера с глубиной на один ход, используя перебор в глубину, т. е. рекурсивно.

Листинг данного алгоритма приведен в листинге 12. Прокомментируем данный программный код. Он очень похож на предыдущий вариант. Сначала объявляем глобальные структуры данных, которые будут нужны при расчете хода.

Далее устанавливаем x координату черной шашки, которой компьютер будет ходить, равной «-1». Это позволит не вводить дополнительный флаг того, что ход не найден, т. к. реальные координаты не могут быть отрицательными. Далее вычисляем значение счета по текущей ситуации на игровом поле и двойным циклом перебираем все

клетки на игровом поле. В данном двойном цикле выбираем только клетки с черными шашками (это обеспечивает первый if). Для каждой клетки с черной шашкой проверяем ее на простые ходы, рассчитывая счет хода и сохраняя лучший ход. Комментарии в скрипте детально объясняют данный процесс. После перебора всех возможных простых ходов данной шашки переходим к рассмотрению ее сложных ходов, но уже с помощью рекурсии.

Листинг 12

```
def black_go():
    global sc_bl,pole_sc,pole,lst_bl_go,xb1,yb1,xb2,yb2,fl_cmx
    xb1 = -1
    sc_bl = score(pole) # начальный счет
    for x in range(8): # перебор всех черных шашек
        for y in range(8):
            if pole[x][y] == 1:
                for ix, iy in (0, -1), (1, 0), (0, 1), (-1, 0): # пр. ходы
                    if 0 <= y + iy <= 7 and 0 <= x + ix <= 7:
                        if pole[x + ix][y + iy] == 0:
                            pole[x][y], pole[x + ix][y + iy] = 0, 1 # ход
                            sc_t = score(pole)              # счет хода
                            pole[x][y], pole[x + ix][y + iy] = 1, 0 # откат
                            if sc_t >= sc_bl:          # сохр. лучшего хода
                                sc_bl = sc_t
                                xb1, yb1= x, y
                                xb2, yb2= x + ix, y + iy
                lst_bl_go = [(x,y)]
                xc, yc = x, y
                fl_cmx = False
                find_cs_go(x,y,pole)
                if fl_cmx:
                    xb1, yb1= xc, yc
    if xb1 == -1:
        lab['text'] = 'у черных\n нет\nходов' + str(sc_bl) + str(sc_t)
    else:
        pole[xb1][yb1], pole[xb2][yb2]= 0, 1
```

Для этого создаем список клеток, в котором последовательно сохраняются координаты клеток возможных «прыжков» активной шашки. Это делается для исключения зацикливания. Для запуска рекурсии помещаем в формируемый список координаты активной клетки, сохраняем координаты активной шашки и сбрасываем флаг сложного хода. Следующим шагом запускаем рекурсивную функцию find_cs_go(),

передавая ей в качестве параметров координаты активной шашки и модель игрового поля. После возврата из рекурсивной функции в случае наличия сложного хода (флаг установлен) запоминаем координаты активной шашки для дальнейшей реализации хода. Код рекурсивной функции find_cs_go() представлен в листинге 13.

Листинг 13

```
def find_cs_go(xr,yr,pole_r):
    global xb2,yb2,lst_bl_go,sc_bl,fl_cmx
    new_pole = pole_r.copy()
    for ix, iy in (0, -2), (-2, 0), (0, 2), (2,0):
        if 0 <= yr + iy <= 7 and 0 <= xr + ix <= 7:
            if abs(new_pole[xr+ix//2][yr+iy//2])==1 and \
                    new_pole[xr+ix][yr+iy]==0: #прыжок
                if not (xr+ix, yr+iy) in lst_bl_go:
                    lst_bl_go.append((xr+ix, yr+iy))
                    new_pole[xr][yr], new_pole[xr + ix][yr + iy] = 0, 1 # ход
                    sc_t = score(new_pole)  # счет хода
                    if sc_t >= sc_bl:  # сохр. лучшего хода
                        sc_bl = sc_t
                        xb2, yb2 = xr + ix, yr + iy
                        fl_cmx =True
                    find_cs_go(xr + ix,yr + iy,new_pole)
                    new_pole[xr][yr], new_pole[xr + ix][yr + iy] = 1, 0 #отк
    return
```

Прокомментируем данный код. Рекурсивная функция получает на вход координаты последнего «прыжка» активной шашки и матрицу (модель игрового поля). В самой функции объявляются необходимые для работы глобальные структуры данных и создается копия матрицы игрового поля. Копия матрицы нужна для того, чтобы каждый вызов рекурсивной функции работал со своей моделью игрового поля. Далее рассматриваем возможные «прыжки» из полученных координат. Если такой «прыжок» возможен и он отсутствует в списке возможных «прыжков» шашки, что обеспечивается тремя последовательными условными операторами, выполняются следующие действия. Координаты возможного «прыжка» записываются в список, производится ход, вычисляется счет, сохраняется лучший ход, запускается рекурсия для данного «прыжка» и осуществляется откат в предыдущую позицию для

проверки оставшихся «прыжков». После проверки всех возможных четырех «прыжков» производится возврат в вызывающую функцию. Как видно, данный алгоритм обеспечивает динамическое построение и анализ дерева ходов шашки методом перебора в глубину.

Как отмечалось ранее, приведенные алгоритмы обеспечивают выбор наилучшего хода в текущей позиции. Они хорошо играют в тактическом плане. Для игры с учетом некоторой стратегии нужно, чтобы они просчитывали ходы на определенную глубину, например два или три. Причем, чем больше глубина просчета, тем более «умным» будет алгоритм. Для этого необходимо на каждом уровне запоминать не только лучший ход, но все ходы с их счетом. После расчета своих ходов компьютер должен просчитать все ходы противника, также запомнив как сами ходы, так и их счет. И так далее, пока не будет достигнута требуемая глубина просчета. Далее нужно выбрать лучший счет (для себя) и уже по построенному дереву игры вернуться в исходное состояние, запомнив путь по дереву с учетом принципов МиниМакса.

Во всех функциях определения хода компьютера используется функция score() для вычисления счета хода. Поскольку данная функция вызывается очень часто, к ней предъявляются два основных противоречивых требования: точность и простота. Вариант такой функции для итерационного варианта представлен на листинге 14. Он достаточно прост, но учитывает только положение шашек, не принимая во внимание их взаимное расположение, т. е. возможные комбинаторные ходы. Но в силу простоты этого будет достаточно. Прокомментируем данную функцию.

Вначале определяем глобальные структуры данных: модель игрового поля pole и матрицу весов клеток игрового поля. Далее двойным циклом, проходя по игровому полю, накапливаем сумму счета. Если на клетке стоит белая шашка, то к сумме прибавляется вес этой клетки. Если на

клетке стоит черная шашка, из суммы вычитается значение (14 – вес клетки).

Листинг 14

```
def score():
    global pole, pole_sc
    sc1 = 0
    for isc in range(8):
        for jsc in range(8):
            if pole[isc][jsc] == 1:
                sc1 += pole_sc[isc][jsc]
            elif pole[isc][jsc] == -1:
                sc1 -= 14 - pole_sc[isc][jsc]
    return sc1
```

Практическая часть

1. Разработайте оценочную функцию.
2. Напишите Питон приложение с графическим интерфейсом для приложения в соответствии с вашим вариантом курсовой работы.

Контрольные вопросы

1. Дайте характеристику GUI приложения.
2. Перечислите основные виджеты библиотеки tkinter.
3. На чем основывается и алгоритм МиниМакса?
4. В чем суть алгоритма МиниМакса?

ТЕСТИРОВАНИЕ ПРИЛОЖЕНИЯ

Общие сведения

Как уже отмечалось, в процессе развития программирования расширялись сферы его применения, развивались инструментальные системы, увеличивались объемы программных приложений. Все это приводило к увеличению сложности процесса разработки ПО. Что, в свою очередь, отрицательно сказывалось как на самом процессе разработки (не обеспечивалась прогнозируемая эффективность процесса разработки ПО), так и на качестве разрабатываемых приложений (не обеспечивалось требуемое качество приложений). Это приводило к необходимости развития подходов к эффективному управлению разработкой ПО. Суть данного процесса заключалась в разработке и исследовании различных моделей, методов и инструментальных средств, обеспечивающих перевод интуитивного хаотичного процесса разработки в процесс, управляемый по строгой технологии.

С некоторыми технологическими аспектами разработки мы познакомились ранее – технология разработки ПО, модель ПО, архитектура ПО. Существенную роль в разработке ПО играет процесс тестирования.

Он не только обеспечивает повышение качества ПО, но позволяет повысить эффективность разработки при постоянном изменении требований к ПО и увеличении его объема.

Тестирование, как элемент разработки ПО, появилось практически с появлением программирования. Первыми программистами были математики и инженеры. Первые привыкли, что все новые теории, гипотезы должны проверяться. Вторые привыкли, что новые изделия должны проходить обязательный контроль. Развитие программирования

привело к развитию тестирования от простых приемов проверки и контроля к целому направлению информационных технологий – QA (от английского quality assurance – обеспечение качества), основным инструментом которого является тестирование. Обеспечению качества ПО и его тестированию посвящено достаточно много как научной, так и учебной литературы [15-20].

Тестирование ПО относится к большому научно-техническому направлению – технической диагностике. Вопросы технической диагностики присутствуют практически в любом инженерном проекте [15,16]. Оснований для этого достаточно много. В том числе и ошибки проектных решений разработчиков. В технической диагностике различают понятия дефекта, ошибки, сбоя, отказа. Под дефектом понимается неисправность какого-либо элемента системы или неверное проектное решение (ошибка проекта). Дефект может приводить или не приводить к ошибке работы системы. Например, в каких-то состояниях дефект влияет на работу системы, а в каких-то – нет. Под отказом понимается нерабочее состояние системы. А под сбоем кратковременный самоустраняющийся отказ.

Особенности разработки ПО, как вида деятельности, уточняют как терминологию, так и общие подходы, присущие технической диагностике. При этом обычно, кроме особых случаев, считается, что приложение выполняется на идеально работающей аппаратуре (компьютере). И все дефекты, которые присутствуют в приложении, – это ошибки процесса проектирования ПО. Дефект в ПО называется багом. Он может приводить к ошибке (на определенных данных или в определенных режимах работы приложения) или не приводить к ней. Наличие бага в приложении (даже не приводящего к ошибке) недопустимо, т. к. он может появиться в самый неподходящий момент времени.

В настоящее время этап тестирования присутствует практически в любой технологии разработки ПО, как, например, в каскадной технологии (рисунок 2). Под тестированием ПО (Software Testing) обычно понимают проверку соответствия реальных и ожидаемых результатов поведения программы, проводимую на специальном конечном наборе входных данных, называемых тестами [19]. Целью тестирования является проверка соответствия ПО предъявляемым требованиям (ТЗ). Оно способствует поиску ошибок в ПО, которые должны быть выявлены и исправлены до того, как их обнаружат пользователи программы.

Основная задача тестирования заключается в предоставлении информации о том, как работает программа и какие в ней найдены ошибки на текущий момент времени. Общеизвестно, что чем раньше ошибка обнаружена, тем меньше стоит процесс ее устранения (по времени и по ресурсам).

Существуют разные классификации видов тестирования. Перечислим виды тестирования по книге [18]:

1) По знанию системы:

1.1) Черный ящик;

1.2) Белый ящик.

2) По позитивности:

2.1) Позитивное;

2.2) Негативное.

3) По целям:

3.1) Функциональное;

3.2) Нефункциональное.

4) По исполнителям (по субъекту):

4.1) Альфа-тестирование;

4.2) Бета-тестирование.

5) По степени автоматизации:

5.1) Ручное;

5.2) Автоматизированное.

6) По состоянию системы:

6.1) Статическое;

6.2) Динамическое.

Разберем кратко каждый вид.

Тестирование «черного ящика» – это метод, при котором тестировщик не имеет доступа к исходному коду программы. Целью тестирования в этом случае является выявление багов в приложении при отсутствии знания ее внутренней структуры. Тестирование «белого ящика», напротив, основывается на знании внутренней структуры приложения. Тестировщик имеет доступ к исходному коду программы и может использовать его для создания тестовых сценариев. Целью тестирования «белого ящика» является проверка работы программы на уровне отдельных функций и компонентов.

Позитивное тестирование проверяет, что основной функционал работает в соответствии с техническим заданием, т. е. для каждого пользователя все сценарии использования приложения выполнимы и приводят к ожидаемому результату. Негативное тестирование направлено на определение поведения приложения в нетипичных ситуациях. Например, отработка при вводе чисел вместо текста, или наоборот. Негативное тестирование не менее важно, чем позитивное, но приоритет его ниже.

Функциональное тестирование – это проверка функциональности программного обеспечения согласно техническому заданию. Тестировщик проверяет, выполняет ли программа требуемые функции и работает ли она правильно в различных ситуациях. Нефункциональное тестирование – это проверка аспектов программного обеспечения, не связанных с его функциональностью. К таким аспектам относятся производительность,

надежность, безопасность, удобство использования и другие характеристики, которые могут влиять на качество и эффективность программы.

Альфа-тестирование – это тип тестирования, проводимый внутри организации-разработчика программного обеспечения. Оно выполняется перед публичным релизом программы и позволяет выявить ошибки и недостатки, которые могут быть исправлены до выпуска программы на рынок. Бета-тестирование проводиться внешними пользователями программного обеспечения. Оно выполняется после альфа-тестирования и позволяет получить обратную связь от реальных пользователей, чтобы выявить ошибки и недостатки, которые могут быть исправлены перед официальным выпуском программного обеспечения.

Ручной процесс тестирования осуществляется тестировщиком.

Он составляет тесты и управляет всеми стадиями тестирования.

При автоматизированном тестировании проверку функциональности приложения выполняет программа. От тестировщика требуется грамотно составить тест, перевести его в исполняемый код и запустить программу.

Статическое тестирование осуществляется, когда приложение не запущено. В него входит тестирование технической документации и анализ исходного текста программы. Динамическое тестирование выполняется при работающем приложении и включает в себя запуск программы с различными входными данными и проверку результатов выполнения.

В рамках курсовой работы приведем примеры практической реализации трех видов тестирования:

- ручное тестирование позитивных и негативных сценариев работы приложения;

- статическое тестирование исходного кода и технических документов;

- автоматизированное тестирование функций программного кода.

В результате сможем оценить эффективность и надежность разработанного приложение.

Тестирование, как неотъемлемая часть разработки ПО, в последнее время существенно развилось и не ограничивается вопросами, затронутыми в учебном пособии. Мы рассмотрели лишь ограниченную часть методов, знание которых позволит студентам провести разработку в соответствии с промышленной технологией создания ПО.

Статическое тестирование

Статическое тестирование позволяет выявить потенциальные проблемы и ошибки в коде программы еще до ее запуска. Помогает повысить качество и надежность ПО, а также улучшить процессы ведения технической документации.

Шаги статического тестирования курсовой работы:

1. Проверьте завершенность подготовки проектной документации (техническое задание, пояснительная записка и руководство программиста) и исходного кода программы.
2. Внимательно изучите проектную документацию и убедитесь, что они являются полными, однозначными и согласованными.
3. Проверьте проектную документацию на соответствие требованиям ГОСТ, наличие орфографических, синтаксических и пунктуационных ошибок.
4. Выполните анализ кода. Проверьте, соответствует ли код программы установленным стандартам и рекомендациям разработки. Проанализируйте синтаксис и структуру кода: правильное использование ключевых слов, операторов, скобок; выполните поиск потенциальность ошибок, таких как неправильное присваивание значений переменных,

деление на ноль, выход за границы массива, наличие неиспользуемых переменных, дублирование кода, неоптимальные алгоритмы, корректность комментариев и т. д.

Ручное тестирование

Существуют различные варианты оформления ручных тестов. Подробнее остановимся на двух основных: тест-кейс и чек-лист.

Тест-кейсы – это детальные инструкции, которые описывают шаги, ожидаемые результаты и ожидаемое поведение программы при выполнении определенного теста. Они используются для тщательного и систематического тестирования программы. Тест-кейсы полезны, когда требуется провести множество тестов в различных сценариях использования программы.

Тест-кейс имеет следующую структуру:

1. Номер тест-кейса и его название;
2. Предварительные шаги (необязательный пункт) – все то, что может помочь пройти тест-кейс, но прямого отношения к нему не имеет;
3. Шаги – действия которые нужно выполнить, чтобы получить ожидаемый результат;
4. Ожидаемый результат – что именно мы должны проверить;
5. Фактический результат – полученный результат теста (добавляется тестировщиком по результату прохождения).

Рекомендации по составлению тест-кейсов [18].

1) Хорошее заглавие. Руководствоваться правилом: «Кратко, но емко»;
2) Писать обезличено;
3) Писать в одном стиле.

Чек-лист – это список проверок. Главная задача чек-листа – напоминать тестировщику провести определенную проверку.

Чек-листы используются для более общего и поверхностного тестирования программы. Чек-листы полезны, когда требуется проверить выполнение определенных функций с разным набором входных данных.

Чек лист не имеет определенной структуры написания. Рекомендуется использовать следующую структуру [18]:

1) Описание тестируемого элемента;

2) Пример – конкретные входные данные для тестирования;

3) Результат – выходные данные (результат тестирования).

Тест-кейс и чек-лист удобные и простые инструменты ручного тестирования вашего приложения. В таблице 2 приведена их сравнительная характеристика.

Таким образом, тест-кейсы и чек-листы используются в разных ситуациях в зависимости от целей и требований тестирования. Тест-кейсы подходят для более детального и систематического тестирования различных функций приложения, в то время как чек-листы могут быть полезны для более общего и поверхностного тестирования одной функции с разнообразными наборами входных данных.

Таблица 2

Сравнение вариантов оформления ручных тестов

Характеристика	Тест-кейс	Чек-лист
Описание	Подробное	Краткое
Количество текста	Много	Мало
Простота понимания	Да	Нет
Поддержка	Сложная	Простая

Автоматизированное тестирование

При частом изменении функциональности приложения и увеличении программного кода, сложность и трудоемкость ручного тестирования увеличивается. В этом случае может быть применено автоматизированное тестирование. Автоматизация значительно сокращает время и ресурсы, которые требуются для тестирования вручную, повышает качество программного продукта и ускоряет его выпуск в эксплуатацию. Автоматизированное тестирование позволяет повторно использовать тестовые сценарии, что упрощает процесс тестирования при внесении изменений в код или добавлении новых функций. Кроме того, автоматизированные тесты могут быть запущены в любое время и на любой стадии разработки, позволяя выявлять проблемы на ранних этапах и ускорить процесс разработки.

Одним из распространенных методов автоматизации тестирования является unit-тестирование. Целью unit-тестирования является проверка отдельных компонентов программного обеспечения, таких как функции или модули в соответствии с проектной документацией. Unit тестирование выявляет баги на ранних стадиях разработки, позволяя быстро их исправить и улучшить качество кода. Способствует повышению надежности и стабильности программного продукта, а также облегчает его сопровождение и развитие.

В соответствии с [19] Unit-тестом называется автоматизированный тест, который проверяет правильность работы небольшого фрагмента кода (также называемого юнитом). Он делает это быстро, поддерживая изоляцию от другого кода.

В языке Python имеется готовый встроенный модуль unittest, обеспечивающий разработчика ПО необходимыми инструментами для написания и запуска unit-тестов.

Чтобы написать unit-тест, создайте новый файл в репозитории вашего проекта с названием, обязательно содержащим начальную приставку «test_». Например, для приложения «Ugolki.py» это будет «test_Ugolki.py». Данный файл будет содержать блок тестируемых функций приложения.

Для использования библиотеки, ее нужно импортировать (Листинг 15).

Листинг 15

```
import unittest
```

Структура unit-теста выглядит следующим образом (Листинг 16).

Листинг 16

```
class TestStringMethods(unittest.TestCase):
   def test_namefuncion(self):
       # Код тестируемых функций
if __name__ == '__main__':
   unittest.main()
```

После импорта библиотеки, создается класс TestStringMethods, который наследуется от класса unittest.TestCase. Этот класс предоставляет множество методов для написания тестов. Внутри класса TestStringMethods определен метод test_namefuncion(). Здесь пишутся сами тесты. Обратите внимание, что название метода должно начинаться с префикса "test_", чтобы unittest мог автоматически определить его как тестовый метод.

Внутри метода test_namefuncion() пишется код тестируемых функций. Здесь вызываются функции проверки библиотеки, передаются им аргументы и проверяются ожидаемые результаты с помощью специальных утверждений, предоставляемых unittest.TestCase.

В конце добавляется условие «if name == 'main':», чтобы запустить тесты только в случае, если файл был запущен напрямую, а не импортирован как модуль. В этом случае вызываем метод unittest.main(), который запускает все тесты в классе TestStringMethods.

Принцип каждого теста – это вызов встроенной функции проверки. Модуль unittest предоставляет ряд возможностей для самых разных проверок. Основные функции проверок приведены в таблице 3. Весь перечень функций содержится в официальной документации [20].

Таблица 3

Основные функции проверок

Функция	**Проверка**
assertEqual(a, b)	a == b
assertNotEqual(a, b)	a != b
assertTrue(x)	bool(x) is True
assertFalse(x)	bool(x) is False
assertLess(a, b)	a < b
assertGreater(a, b)	a > b

В листинге 17 приведен пример unit-тестов для функции проверки четности числа.

Листинг 17

```
import unittest

def is_even(number):
    return number % 2 == 0
class TestIsEven(unittest.TestCase):
    def test_is_even(self):
        self.assertTrue(is_even(4))
        self.assertFalse(is_even(3))
if __name__ == '__main__':
    unittest.main()
```

В случае успешного прохождения тестов, получим в терминале сообщение о положительном результате (Листинг 18):

Листинг 18

```
Ran 1 test in 0.000s
OK
```

В случае ошибки хотя бы одной тестовой функции, получим в терминале сообщение об отрицательном результате (Листинг 19):

Листинг 19

```
FAIL: test_is_even (__main__.TestIsEven)
----------------------------------------------------------------------
Traceback (most recent call last):
  File "D:\Python3\Unittest\test_isEven.py", line 9, in test_is_even
    self.assertFalse(is_even(4))
AssertionError: True is not false

----------------------------------------------------------------------
Ran 1 test in 0.001s

FAILED (failures=1)
```

Отчет о тестировании

Отчет о тестировании ПО – это документ, который содержит информацию о проведенных тестах, их результатах, выявленных дефектах и рекомендации по улучшению качества программного продукта. Главная цель отчета о тестировании ПО – предоставить заказчику и команде разработки полную информацию о текущем состоянии качества ПО.

Назначение отчета о тестировании ПО:

1. Дать общую оценку качества программного продукта;
2. Предоставить информацию о выполненных тестах и их результатах;
3. Документировать выявленные баги;
4. Предложить рекомендации по улучшению качества ПО;
5. Обеспечить прозрачность и отслеживаемость процесса тестирования.

Перед началом тестирования необходимо определить, что тестировать. Полезным инструментом для этого является Mindmap. Mindmap – это интеллектуальная карта приложения в виде диаграммы для визуализации ее основных функций. Перед началом тестирования рекомендуется составлять Mindmap с точки зрения различных сценариев использования приложения. Выделите основные функции приложения, зарисуйте их, а затем детализируйте и дополните второстепенными сценариями, как на рис. 9. Рекомендуется записывать функции, отвечающие на вопрос «Что можно сделать в приложении?». Mindmap создается с помощью специальных инструментов. Например, можно использовать некоммерческие инструменты: XMind, Figma.

Рисунок 9. Mindmap приложения Уголки

Не существует определенного шаблона отчета о тестировании ПО. В зависимости от специфики проекта, структура отчета может сильно отличаться. Для курсовой работы предлагается следующий шаблон.

1. Описание отчета

В данном отчете представлены результаты тестирования программы на основе разработанных test-case и чек-листов, написанных unit-тестов и

статического тестирования документации и программного кода. Описаны проведенные тесты, их результаты и обнаруженные дефекты.

2. Цель тестирования

Целью тестирования является проверка соответствия ПО предъявляемым требованиям, а также выявление возможных багов. По результатам тестирования следует исправление выявленных багов.

3. Методика тестирования

Тестирование проводилось с использованием следующих методов:

– Статическое тестирование: анализ и проверка кода без его запуска, выявление ошибок в технической документации.

– Ручное тестирование: запуск пользовательских сценариев программы с различными входными данными и проверка корректности полученных результатов.

– Автоматизированное тестирование: написание и запуск набора unit-тестов отдельных модулей программы.

4. Проведенные тесты

В ходе тестирования были проведены следующие тесты:

4.1) Статическое тестирование

Количество обнаруженных и исправленных ошибок в документации: [количество ошибок].

Количество обнаруженных и исправленных ошибок в программном коде: [количество ошибок].

4.2) Ручное тестирование.

Написаны и проведены следующие тест-кейсы и чек-листы:

ТК1. Отработка авторизации.

Предварительные шаги:

Зарегистрироваться с логином: TestTest и паролем 12345678.

Шаги:

1. Запустить приложение.

2. В окне регистрации, в поле «логин» ввести TestTest, а в поле «пароль» – 12345678.

3. Нажать кнопку «Авторизоваться».

Ожидаемый результат:

Пользователь начнет игру.

Фактический результат:

Пользователь начал новую игру.

ТК2. Прохождение регистрации односимвольными логином и паролем.

Шаги:

1. Запустить приложение.

2. В окне регистрации, в поле «логин» ввести 1, а в поле «пароль» – 1.

3. Нажать на кнопку «Зарегистрироваться/ начать игру».

Ожидаемый результат:

Пользователь получит сообщение об ошибке.

Фактический результат:

Пользователь зарегистрировался.

Таблица 4

Чек-лист для формы регистрации

Описание	Пример	Результат
Проверка поля «логин»		
Корректный логин	TestTest	Сообщение об успешной регистрации
Цифры внутри логина	Te1235st36Test	Сообщение об успешной регистрации
Пустая строка		Ошибка
Количество символов 1	T	Ошибка

4.3) Автоматизированное тестирование

Для курсовой работы было составлено n-количество unit-тестов. Тесты покрывают следующие функции приложения: [перечислить тестируемые функции].

Все unit-тесты прошли успешную проверку.

5. Выводы

На основе проведенных тестов сделаны следующие выводы:

– Программа успешно прошла все тесты и работает корректно.

– Обнаружены и исправлены следующие дефекты: [краткий список исправленных дефектов, если есть].

– Рекомендации по дальнейшему развитию программы: добавление ограничения времени на ход пользователя, звукового сопровождения, таблицы лидеров.

Практическая часть

1) Составить Mindmap приложения.

2) Провести статическое тестирование документации и кода курсовой работы.

3) Составить и провести ручное тестирование по следующим сценариям:

– Позитивное тестирование: регистрация, авторизация, игровой процесс.

– Негативное тестирование: регистрация, авторизация, игровой процесс.

4) Составить unit-тест, покрывающий основные функции приложения.

5) Составить отчет о тестировании.

Контрольные вопросы

1. Что такое тестирование?
2. В чем заключаются цель и задачи тестирования?
3. Зачем проводить тестирование?
4. Какие бывают виды тестирования?
5. Что такое тест-кейсы и чек-листы?
6. Когда применяются тест-кейсы, а когда чек-листы?
7. Что такое статическое тестирование?
8. Что такое unit-тесты?
9. В чем назначение отчета о тестировании?
10. Что такое Mindmap приложения?

ОБЩИЕ ТРЕБОВАНИЯ К КУРСОВОЙ РАБОТЕ

ПОСТАНОВКА ЗАДАЧИ НА КУРСОВУЮ РАБОТУ

Общие требования к работе:

1) Инструментальная среда разработки – Visual Studio Code/PyCharm (или другая современная среда разработки);

2) Язык программирования – Python 3.9.0 и выше (для индивидуальных заданий может использоваться другой язык программирования);

3) Функциональная целостность и согласованность всех элементов приложения;

4) Состав и особенности реализации приложения определить в соответствии с вариантом (приложение 1).

Требуется разработать приложение, которое:

1) Реализует базовую логику настольной игры через графический интерфейс (отрисовка игрового поля, начальная расстановка фигур, ход фигуры, рубка фигуры противника и т. п.);

2) Реализует формы регистрации/авторизации;

3) В качестве базы данных использует текстовый файл;

4) Содержит как минимум следующие обязательные алгоритмы: шифрование, расшифровки, проверка доступности хода, проверка возможности хода, оценочный, проверка конца игры;

5) Использует библиотеку для написания пользовательского интерфейса и логики игры tkinter;

6) Выступает в качестве второго игрока.

Требования к программной документации:

1) «Техническое задание» на реализуемое приложение должно соответствовать ГОСТ 19.201-78 «Техническое задание. Требования к содержанию и оформлению»;

2) «Пояснительная записка» должна соответствовать ГОСТ 19.404-79 «Пояснительная записка. Требования к содержанию и оформлению»;

3) «Руководство программиста» должна соответствовать ГОСТ 19.504-79 «Руководство программиста. Требования к содержанию и оформлению»;

4) Оформление программного кода приложения должно быть в соответствии с ГОСТ 19.401-79 «Текст программы. Требования к содержанию и оформлению».

Повышению итоговой оценки соответствует:

1) Читабельность, документированность и понятность программного кода;

2) Оснащение созданного приложения дополнительной функциональностью, повышающей удобство и универсальность его

использования (Пример: игровая статистика, отображение счетчика ходов, кнопка «Рестарт» и т. п.);

3) Реализация расчетного алгоритма с помощью алгоритмов минимакса и альфа-бета-отсечения;

4) Оригинальность работы т. е. отсутствие существенного влияния работ других студентов;

5) Сдача работы в установленный срокю

ЭТАПЫ ВЫПОЛНЕНИЯ РАБОТЫ

Выполнение работы проходит следующие этапы:

1) Написание технического задания;

2) Написание пояснительной записки;

3) Разработка и отладка прототипа приложения;

4) Разработка и отладка окончательного варианта приложения;

5) Написание руководства программиста;

6) Оформление пояснительной записки курсовой работы;

7) Подготовка презентации.

8) Защита курсовой работы.

После утверждения и закрепления тем за студентами руководитель проводит необходимые консультации на практических занятиях, контролирует ход выполнения работы в целом.

СОСТАВ ОТЧЕТНОСТИ ПО РАБОТЕ

Результатом выполнения курсовой работы является программный продукт – совокупность созданного приложения и следующих проектных и эксплуатационных документов:

1) Работающее приложение;

2) Техническое задание;
3) Пояснительная записка;
4) Руководство программиста;
5) Текст программы.

ОФОРМЛЕНИЕ РАБОТЫ

Пояснительная записка оформляется в виде совокупности документов под одним титульным листом и с общим содержанием. Текст программных и эксплуатационных документов оформляется с использованием текстового редактора в соответствии с требованиями ГОСТ 19.105-78 «Общие требования к программным документам», ГОСТ 19.106-78 «Требования к программным документам, выполненным печатным способом», а также на основе предоставленных шаблонов (Приложения 3 - 5).

Текст делится на разделы, подразделы и пункты. Межстрочный интервал – полтора. Размер шрифта – 14, поля слева – 20 мм, сверху – 25 мм, снизу 15 мм, справа – 10 мм. Нумерация страниц – снизу по центру. Титульный лист каждого документа учитывается в общей нумерации, но не нумеруется. Исходное форматирование разделов и подразделов шаблона не изменяется.

ПОРЯДОК ЗАЩИТЫ. КРИТЕРИИ ОЦЕНКИ РАБОТЫ

Созданный программный продукт предоставляется студентом руководителю с помощью репозитория GitHub как в исходном виде, так и преобразованный в исполняемый файл «exe». В случае наличия каких-либо замечаний студент может доработать свой программный продукт, записав новую версию по той же ссылке. Программные документы также

размещаются в репозитории GitHub. Окончательная (принятая) версия документации предоставляется в печатном виде.

График защиты курсовых работ составляется руководителем и доводится до сведения студентов. Студент обязан убедиться в корректности функционирования приложения до защиты. Проблемы, обнаруженные непосредственно при защите, являются основанием для ее переноса (в соответствии с графиком).

В начале защиты студент сообщает заглавие приложения, кратко формулирует его назначение и указывает основные особенности. В процессе защиты – демонстрирует используемую(мые) структуры данных и алгоритмы, акцентируя внимание на наиболее важных и интересных, демонстрирует работоспособность самого приложения.

Оценка работы осуществляется руководителем с учетом качества ее выполнения, включая качество документации, полноты учета предъявляемых требований, выступления с презентацией и ответов на вопросы в ходе защиты, а также соблюдения сроков выполнения и защиты курсовой работы, а также включая работу на практических занятиях.

Основаниями для получения неудовлетворительной оценки могут являться серьезное несоответствие программного продукта предъявляемым требованиям, неработоспособность приложения, наличие существенных элементов заимствования из чужих работ как в программном коде или интерфейсе приложения, так и в документации, а также слабая ориентация студента в представляемой работе. При неудовлетворительной оценке руководитель определяет направления и объем доработки программного продукта.

ЗАКЛЮЧЕНИЕ

В данном учебном пособии рассмотрены вопросы разработки компьютерных приложений с графическим интерфейсом на языке Питон. В качестве предметной области выбраны настольные логические игры уровня шашек. Рассмотрены элементы теории игр и базовые технологические вопросы разработки программного обеспечения. Это позволяет использовать данное учебное пособие в качестве методических материалов при разработке нетривиальных компьютерных приложений. Теоретический материал учебного пособия сопровождается детальной пошаговой иллюстрацией разработки десктопного компьютерного приложения по игре в «уголки». Кроме непосредственно разработки приложения показывается разработка некоторых программных документов в соответствии с 19 группой государственных стандартов – единой системы программной документации (ЕСПД). На основе изложенного материала даются методические рекомендации по выполнению курсовой работы по дисциплине «Алгоритмы и структуры данных».

Авторы надеются, что степень охвата и подробности изложения материала будут достаточны для успешного выполнения студентами курсовой работы. Также авторы рекомендуют использовать материал данного учебного пособия при выполнении других проектов по разработке программного обеспечения.

БИБЛИОГРАФИЧЕСКИЙ СПИСОК

1. Difficulty of Various Games for Computers [Электронный ресурс]. – Режим доступа: URL:https://blog.carlmjohnson.net/post/xkcd-game-ais-once-again-a-webcomic/ (дата обращения : 29.10.2023)

2. Российский Институт Стандартизации // ГОСТ 19 ЕСПД (ГОСТ 19) Единая система программной документации [Электронный ресурс]. – Режим доступа: URL: https://www.standards.ru/collection.aspx?control=40&id=868075&catalogid=temat-sbor/ (дата обращения : 29.10.2023)

3. Зараменских, Е. П. Управление жизненным циклом информационных систем : учебник и практикум для вузов / Е. П. Зараменских. – 2-е изд. – Москва : Издательство «Юрайт», 2023. – 497 с. – Текст: электронный // Образовательная платформа «Юрайт» [сайт]. [Электронный ресурс]. – Режим доступа: URL: https://urait.ru/bcode/511960 (дата обращения : 29.10.2023)

4. Стандарты и шаблоны для ТЗ на разработку ПО [Электронный ресурс]. – Режим доступа: URL: https://habr.com/ru/articles/328822/ (дата обращения : 29.10.2023)

5. ГОСТ 19.201-78 Техническое задание. Требования к содержанию и оформлению [Электронный ресурс]. – Режим доступа: URL: https://www.prj-exp.ru/gost/gost_19-201-78.php (дата обращения : 29.10.2023)

6. Уголки (игра) [Электронный ресурс]. – Режим доступа: URL: https://ru.wikipedia.org/wiki/Уголки(игра) (дата обращения : 29.10.2023)

7. Вирт Н. Алгоритмы + структуры данных = программы: Пер. с англ. / Н. Вирт. – Москва : Мир, 1985. – 406 с.

8. Мартин, Р. Чистая архитектура. Искусство разработки программного обеспечения / Р.Мартин. – Санкт-Петербург : Питер, 2022. – 352 с.

9. Архитектура программного обеспечения [Электронный ресурс]. – Режим доступа: URL: https://dic.academic.ru/dic.nsf/ruwiki/146913 (дата обращения : 29.10.2023)

10. Блог WEZOM // Архитектура программного обеспечения [Электронный ресурс]. – Режим доступа: URL:

https://wezom.com.ua/blog/arhitektura-programmnogo-obespecheniya (дата обращения : 29.10.2023)

11. Josh Smith / Creating Tic-tac-toe in Swift: User interface [Электронный ресурс]. – Режим доступа: URL: https://ijoshsmith.com/2015/12/22/creating-tic-tac-toe-in-swift-user-interface/ (дата обращения : 29.10.2023)

12. Документация библиотеки tkinter [Электронный ресурс]. – Режим доступа: URL: https://tkdocs.com/index.html (дата обращения : 29.10.2023)

13. Классические технологии проектирования программ [Электронный ресурс]. Режим доступа: URL: https://studfile.net/preview/1444530/page:14/#31 (дата обращения : 29.10.2023)

14. Доусон, М. Программируем на Pyton / М.Доусон. – Санкт-Петербург : Питер, 2021. – 416 с.

15. Технические средства диагностирования: Справочник / В.В.Клюев и др. – Москва : Машиностроение, 1989. – 672 с., ил.

16. Шишкин, В.В. Автоматизация проектирования диагностического обеспечения и диагностирования авиационных бортовых информационных систем / В.В.Шишкин, С.В.Черкашин. – Ульяновск : УлГТУ, 2010. – 174 с.

17. Большой учебник по тестированию [Электронный ресурс]. – Доступ: URL: https://qarocks.ru/big-software-testing-textbook/#fundamentals (дата обращения : 29.10.2023)

18. Назина, О.Е. Что такое тестирование. Курс молодого бойца / О.Е. Назина. – Санкт-Петербург : БХВ-Петербург, 2022. – 592 с.: ил.

19. Хорников, В. Принципы юнит-тестирования / В.Хорников. – Санкт-Петербург : Питер, 2021. – 320 с.: ил. – (Серия «Для профессионалов»).

20. Документация библиотеки Unittest [Электронный ресурс]. – Режим доступа: URL: https://docs.python.org/3/library/unittest.html (дата обращения : 29.10.2023)

Приложение 1

ВАРИАНТЫ ЗАДАНИЙ НА КУРСОВУЮ РАБОТУ

Ознакомится с правилами настольных игр можно на сайте по адресу http://lotos-khv.ru/game/games/games.htm или на других ресурсах.

Вариант	Название настольной игры:
1	Бразильские шашки – Поддавки*
2	Турецкие шашки – Поддавки*
3	Шашки - Самоеды
4	Доджем
5	Петтейя
6	Шашки Вигмана
7	Кены – Поддавки*
8	Турецкие шашки
9	Рэндзю
10	Канадские шашки
11	Апит Содок
12	Реверси
13	Русские шашки
14	Фризские шашки – Поддавки*
15	Международные шашки – Поддавки*
16	Шашки Артамонова
17	Сиджа
18	Атари – Го
19	Болотуду
20	Крестики-нолики до пяти в ряд
21	Евразийские шашки
22	Царские башни – Поддавки*
25	Киммерийские шашки
26	Го-Бан
27	Русские циклические шашки
28	Шашки Клещи – Поддавки*
29	Мак Йек
30	Ставропольские шашки
31	80 – клеточные русские шашки
32	Шашки Клещи
33	Готические шашки
34	Русские шашки – Поддавки*
35	Скифские шашки – Поддавки*
36	Бразильские шашки
37	Киммерийские шашки – Поддавки*

Окончание прил. 1

Вариант	Название настольной игры:
38	Так–Тиль
39	Двухходовые шашки
40	Ставропольские шашки – Поддавки*
41	Скифские шашки
42	Царские башни
43	Волки и Овцы
44	Канадские шашки – Поддавки*
45	Фризские шашки
46	Английские шашки
47	Английские шашки – Поддавки*
48	Шашки Вигмана – Поддавки*
49	Двухходовые шашки – Поддавки*
50	80 – клеточные русские шашки – Поддавки*
51	Мак Йек – Поддавки*
52	Международные шашки
53	Халма
54	Итальянские шашки
55	Итальянские шашки – Поддавки*

***Поддавки –** данная приписка означает что выигрывает не тот, кто заберет, или запрет все шашки противника, а тот, кто сам отдаст все свои шашки или позволит их запереть противнику.

Приложение 2

ШАБЛОН ТИТУЛЬНОГО ЛИСТА И СОДЕРЖАНИЯ

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение
высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

ПОЯСНИТЕЛЬНАЯ ЗАПИСКА

на курсовую работу

по дисциплине «Алгоритмы и структуры данных»

Тема <по распоряжению по факультету>

Р.02069337.<номер зачетки>-<2-зн. номер варианта> ТЗ-<2-зн. номер редакции>

Листов <количество листов документа>

Руководитель разработки:

<должность и ФИО преподавателя>

«____» _____ 202 г.

Исполнитель:
студент гр. ИСТбд-21

<ФИО>

«____» _____ 202 г.

<год>

Инв. | Полп. и | Взд | Инв | Полп. и

Содержание

Приложение 3

ШАБЛОН ТЕХНИЧЕСКОГО ЗАДАНИЯ

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение
высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

ТЕХНИЧЕСКОЕ ЗАДАНИЕ

на курсовую работу

по дисциплине «Алгоритмы и структуры данных»

Тема <по распоряжению по факультету>

Р.02069337.<номер зачетки>-<2-зн. номер варианта> ТЗ-<2-зн. номер редакции>

Листов <количество листов документа>

Исполнитель:
студент гр. ИСТбд-21
<ФИО>
«____» _____ 202 г.

<год>

Полп. и	
Инв	
Взз	
Полп. и	
Инв.	

Введение

Указывается наименование и условное обозначение разрабатываемого приложения, наименования реализуемой игры. Даётся краткий свод правил игры, общая характеристика функциональных возможностей, которое должно предоставлять приложение.

1. Основания для разработки

В качестве оснований для разработки указывается учебный план направления 09.03.02 «Информационные системы и технологии» и распоряжение по факультету.

2. Требования к программе или программному изделию

2.1. Функциональное назначение

Функциональное назначение, перечень автоматизируемых процессов (без излишней детализации), группы пользователей.

2.2. Требования к функциональным характеристикам

2.2.1 Требования к структуре приложения

Приводятся требования к модульной организации приложения.

2.2.2 Требования к составу функций приложения

Указываются наиболее важные функции приложения (возможно с детализацией по группам пользователей).

2.2.3 Требования к организации информационного обеспечения, входных и выходных данных

Указываются наиболее важные требования к пользовательскому интерфейсу и файлам. Структура и алгоритм обмена.

2.3. Требования к надежности

Указываются требования к работоспособности и способам восстановления при сбоях.

2.4. Требования к информационной и программной совместимости

Указываются версия операционной системы, используемых библиотек, базы данных (при наличии), языка Python и используемой среды разработки.

При необходимости указываются стандартные структуры, перечисления, классы, интерфейсы и сфера их применения (с точки зрения организации входных и выходных данных).

2.5. Требования к маркировке и упаковке

Ограничиться фразой «Определяются заданием на курсовую работу».

2.6. Требования к транспортированию и хранению

2.6.1 Условия транспортирования

Ограничиться фразой «Требования к условиям транспортирования не предъявляются».

2.6.2 Условия хранения

Ограничиться фразой «Обеспечение свободного доступа к проекту в репозитории до окончания срока учебы».

2.6.3 Сроки хранения

Ограничиться фразой «Срок хранения – до окончания срока учебы».

3. Требования к программной документации

Ограничиться фразой «Определяются заданием на курсовую работу».

4. Стадии и этапы разработки

Ограничиться фразой «Определяются заданием на курсовую работу».

5. Порядок контроля и приемки

Ограничиться фразой «Определяются заданием на курсовую работу».

Приложение 4

ШАБЛОН ПОЯСНИТЕЛЬНОЙ ЗАПИСКИ

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение
высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

Курсовая работа

По дисциплине «Алгоритмы и структуры данных»

Тема <по распоряжению по факультету>

Пояснительная записка

Р.02069337.<номер зачетки>-<2-зн. номер варианта> ПЗ-<2-зн. номер редакции>

Листов <количество листов документа>

Исполнитель:
студент гр. ИСТбд-21
<ФИО>
«____» _____ 202 г.

<год>

Полп. и | Инв | Взa | Полп. и | Инв.

Введение

Указывается наименование и условное обозначение разрабатываемого приложения, наименования реализованной игры. Приводится описание и обоснование выбранного подхода, краткое описание реализованного приложения.

1. Проектная часть

1.1. Постановка задачи на разработку приложения

Ограничиться фразой «Определяется заданием на курсовую работу. Детализируется в разработанном техническом задании (приложение 1)».

1.2. Математические методы

Приводится описание математической модели, которая будет применяться при реализации функциональности приложения, дается обоснование выбора.

1.3. Архитектура и алгоритмы

1.3.1 Архитектура

Приводится архитектура приложения – основные структуры данных, функции и их взаимодействие.

1.3.2 Алгоритм <название алгоритма>

Приводятся алгоритмы основных функций приложения. Схемы алгоритмов оформляются в соответствии с ГОСТ 19.701-90 «Схемы алгоритмов, программ, данных и систем. Условные обозначения и правила выполнения», либо с помощью другого формального известного инструмента. Даются комментарии и обоснования к алгоритмам.

1.3.2 Алгоритм <название алгоритма>

Количество рассмотренных алгоритмов должно полностью соответствовать техническому заданию. Для алгоритмов шифрования, дешифрования, проверки доступности хода, хода и оценочного добавить блок-схему. В схемах и комментариях не должны использоваться конструкции языка программирования, конкретные названия объектов программы и т. п. (пояснительная записка создаётся до реализации).

1.4. Тестирование

Приводится интеллектуальная карта приложения.

1.4.1 Описание отчета о тестировании

Описывается назначение отчета.

1.4.2 Цель тестирования

Описываются цели тестирования.

1.4.3 Методика тестирования

Описываются методы тестирования.

1.4.4 Проведенные тесты

Приводятся сценарии тестирования с полученными результатами.

1.4.5 Выводы

Описываются выводы тестирования, приводятся рекомендации по улучшению.

2. Источники, использованные при разработке

Приводится перечень источников (книг, статей и т. д.), которые будут использоваться при реализации курсовой работы. Описание источников должно соответствовать ГОСТ Р 7.0.100-2018 «Библиографическая запись. Библиографическое описание. Общие требования и правила составления».

Приложение 5

ШАБЛОН РУКОВОДСТВА ПРОГРАММИСТА

МИНИСТЕРСТВО НАУКИ И ВЫСШЕГО ОБРАЗОВАНИЯ РОССИЙСКОЙ ФЕДЕРАЦИИ

федеральное государственное бюджетное образовательное учреждение
высшего образования

«УЛЬЯНОВСКИЙ ГОСУДАРСТВЕННЫЙ ТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Кафедра «Измерительно-вычислительные комплексы»

Курсовая работа

По дисциплине «Алгоритмы и структуры данных»

Тема <по распоряжению по факультету>

Полп. и

Инв

Взз

Полп. и

Инв.

Руководство программиста

Р.02069337.<номер зачетки>-<2-зн. номер варианта> РП-<2-зн. номер редакции>

Листов <количество листов документа>

Исполнитель:
студент гр. ИСТбд-21
<ФИО>
«____» _____ 202 г.

<год>

1. Назначение и условия применения программы

1.1 Назначение и функции, выполняемые приложением

Формулируется назначение приложения. Дается свод правил реализуемой игры, общая характеристика конкретных функциональных возможностей, которые предоставляет приложение.

1.2 Условия, необходимые для использования приложения

Приводятся требования к операционной системе, платформе, инструментальной среде, библиотекам, необходимым для использования приложения.

2. Характеристики программы

2.1 Характеристики приложения

Указывается количество значимых (т. е. выполняющих какие-то действия) строк программного кода. Количество структур данных, алгоритмов.

Описываются используемые библиотеки.

Описывается работа приложения. Его внешний вид иллюстрируется копиями экранов.

Описываются применяемые средства контроля корректности ввода/вывода (при использовании).

2.2 Особенности реализации приложения

Приводится описание структур данных, использованных в программе (массивов, списков и т. д.), с рассмотрением возможных альтернативных вариантов и обоснованием сделанного выбора.

При необходимости, отмечается и обосновывается заимствование из общедоступных источников нетривиальных программных решений (с точным указанием источников из перечня в «Пояснительной записке»), даётся подробный разбор работы соответствующих программных конструкций.

3. Обращение к программе

Приводятся наименование и полное описание методов, алгоритмов.

Приводятся наименование и полное описание используемых библиотек.

4. Сообщения

Перечисляются сообщения, выдаваемые по результатам контроля корректности ввода/вывода.

Учебное издание

ШИШКИН Вадим Викторинович
АФОНИН Дмитрий Сергеевич

РАЗРАБОТКА ЛОГИЧЕСКИХ КОМПЬЮТЕРНЫХ ИГР С ГРАФИЧЕСКИМ ИНТЕРФЕЙСОМ В СРЕДЕ ПИТОН

Учебное пособие

Редактор Н. А. Евдокимова
ЛР № 020640 от 22.10.97
Подписано в печать 29.11.2023. Формат 60Х84/16
Усл. печ. л. 5,35. Тираж 50 экз. Заказ 534. ЭИ № 1867.

Ульяновский государственный технический университет
432027, Ульяновская область, г.Ульяновск, ул. Сев. Венец, д. 32.
ИПК «Венец» УлГТУ, 432027, Ульяновская область, г.Ульяновск, ул. Сев. Венец, д. 32