**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 «Доджем»

Руководство программиста

**Инв. № подл.**

**Подп. и дата**

**Взам. инв. №**

**Инв. № дубл.**

**Подп. и дата**

Р.02069337.21/1892-11 РП-01

Листов 26

**Исполнитель**:

студент гр. ИСТбд-22

*Скворцов Андрей Александрович*

« » 2024 г.

**2024**

**1. Назначение и условия применения программы**

**1.1 Назначение и функции, выполняемые приложением**

Приложение предназначено для игры в доджем с компьютером.

Правила игры:

На доске размером 5x5 игроки двигают по одной шашке за ход. Фигура может ходить только на 1 клетку горизонтально или вперед, если там пустая клетка. Если шашка доходит до противоположного края поля, она выходит из игры. Цель игры в доджем — первым убрать с поля свои шашки.

Функциональные возможности приложения:

1. Регистрация / авторизация пользователя;
2. Проверка введённых данных на корректность;
3. Шифрование логина и пароля;
4. Проверка правильности хода;
5. Возможность поиграть с компьютером;
6. Определение победителя;
7. Возможность начать игру снова.

**1.2 Условия, необходимые для использования приложения**

Приложение можно использовать на персональном компьютере. Необходимо 300 мб свободной памяти на компьютере.

При разработке приложения использовались:

1. OC Windows 10 Профессиональная;
2. Python версии 3.9;
3. PyCharm Version: 2021.3.2.
4. Библиотеки: PyGame, Tkinter.

**2. Характеристики программы**

**2.1 Характеристики приложения**

Количество значимых строк кода – 436

Количество алгоритмов – 14

Количество сторонних библиотек – 0

Используются библиотеки:

1. PyGame – для работы с графическим интерфейсом самой игры;
2. PyQt5 – для работы с интерфейсом форм;
3. tkinter – для вывода окон уведомлений;

Работа приложения:

При запуске приложения появляется окно регистрации/авторизации (рис. 1), пользователь может ввести от 5 до 15 символов как в поле для логина, так и в поле пароля, без пробелов.

В случае если пользователь не найден, были введены неверные данные или вовсе оставил поля для ввода логина и пароля пустыми, появится окно с предупреждением (рис. 2).

В случае если пользователь ввел пробел в логине или пароле (рис. 3), появится, окно с предупреждением.

В случае если пользователь попытается зарегистрироваться введя логин который уже существует, будет выведено окно с предупреждением (рис. 4).

После авторизации открывается окно с игрой, в которой пользователь играет против искусственного интеллекта, который ходит случайным образом (рис. 5).

Кнопки игрового поля позволяют пользователю совершать ходы в соответствии с правилами игры. Если шашки одной из сторон выйдут с поля, появится оповещение о победе (рис. 6).

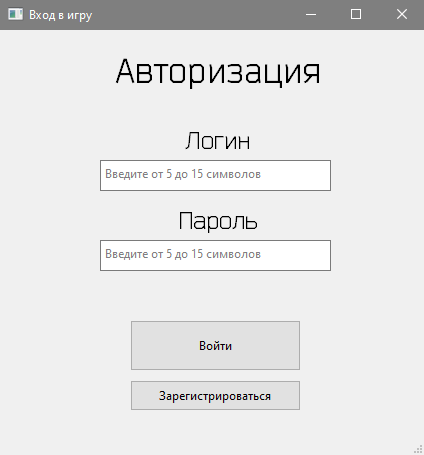


Рис. 1. – Форма регистрации.

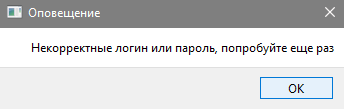


Рис.2. – Оповещение при неудачном входе.

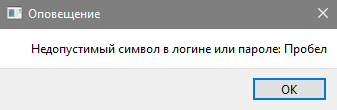


Рис.3. – Оповещение при регистрации с пробелом в пароле или логине.

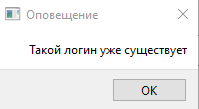


Рис.4. – Оповещение при попытке регистрации уже существующей учетной записи

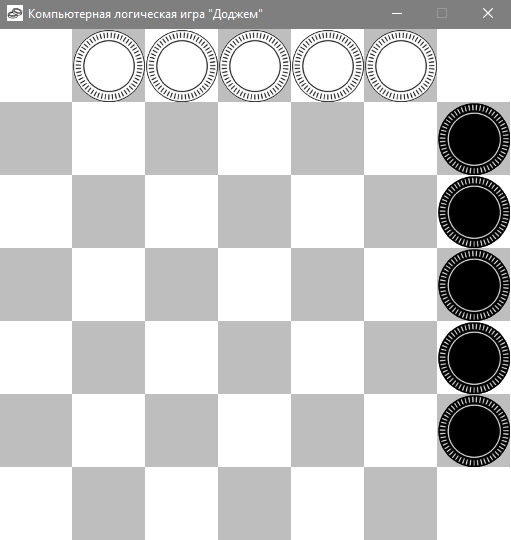


Рис.5. – Игра.

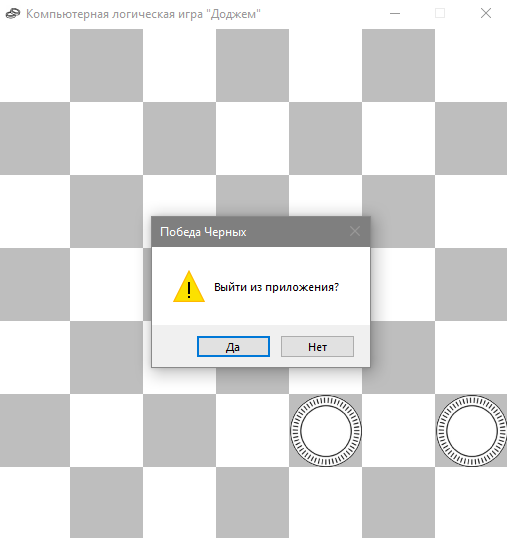


Рис.6. – Победа черных.

**2.2 Особенности реализации приложения**

В приложении используется массив для реализации 2D доски.

Возможности пользовательского и графического интерфейса:

1. Шифрование данных, вводимых пользователем при регистрации;
2. Сохранение зашифрованных данных в текстовом файле;

Функциональные возможности:

1. Реализация возможности вывода на экран поля и игровых объектов;
2. Реализация возможности расставления и передвижения фигур;

Для графического отображения в приложении используется библиотека PyQt5, PyGame и tkinter.

**3. Обращение к программе**

Методы:

1. click\_reg – Регистрация.
2. click\_auth – Авторизация.
3. main – Основной цикл.
4. drawGameState – Генерация содержимого игрового окна.
5. drawBoard – Генерация игровой доски.
6. drawPieces – Заполнение доски шашками.
7. highlightSquares – Подсвечивание доступных ходов.
8. GetBoardNotation – Получение ходов по нотации Эдвардса.
9. findOptimalMove – Нахождение выгодного хода для компьютера.
10. makePlayerMove – Осуществление хода игрока.
11. clear – Очищение полей авторизации.
12. message – Отображение сообщений.
13. Encrypt – Шифрование данных.
14. Decipher – Дешифрование данных.
15. log – Получение логина из поля авторизации.
16. pas – Получение пароля из поля авторизации.
17. loadImages – Нахождение выгодного хода для компьютера.
18. undoMove – Отмена последнего хода компьютера.

В данном приложении используются библиотеки:

1. PyGame – для работы с графическим интерфейсом самой игры;
2. PyQt5 – для работы с интерфейсом форм;
3. tkinter – для вывода окон уведомлений;

**4. Сообщения**

1. «Неверный логин или пароль, попробуйте еще раз» - В случае если пользователь не найден, были введены неверные данные или вовсе оставил поля для ввода логина и пароля пустыми;
2. «Такой логин уже существует» - В случае если пользователь попытается зарегистрироваться введя логин который уже существует;
3. «Недопустимый символ в логине или пароле: Пробел» - В случае если пользователь ввел пробел в логине или пароле;
4. «Победа Черных» - В случае победы стороны черных шашек;
5. «Победа Белых» - В случае победы стороны белых шашек;

Текст программы

import pygame as p

import tkinter as tk

from tkinter import messagebox

import random

from PyQt6 import QtWidgets

from PyQt6.QtWidgets import QMessageBox

from AuthWindow import Ui\_MainWindow

import numpy as np

import sys

# Конфигурация

WIDTH = HEIGHT = 512

DIMENSION = 7

SQ\_SIZE = HEIGHT // DIMENSION

MAX\_FPS = 15

IMAGES = {}

fullEscape = 5

# Окно авторизации

app = QtWidgets.QApplication(sys.argv)

MainWindow = QtWidgets.QMainWindow()

ui = Ui\_MainWindow()

ui.setupUi(MainWindow)

MainWindow.show()

# Всплывающие оповещения

def message(text):

msg = QMessageBox()

msg.setWindowTitle("Оповещение")

msg.setText(text)

msg.exec()

# Шифрование

def Encrypt(word):

word = list(word)

key = 7

word = np.array(word)

pass\_len = len(word)

matrix = np.full((5 + key, key), ' ')

flag = 0

for i in range(5 + key):

for j in range(key):

if flag < pass\_len:

matrix[i][j] = word[flag]

flag += 1

matrix = matrix.transpose()

result = ""

for i in range(key):

for j in range(5 + key):

result += matrix[i][j]

return result

# Дешифрование

def Decipher(word):

word = list(word)

key = 7

pass\_len = len(word)

matrix = np.full((key, 5 + key), ' ')

flag = 0

for i in range(key):

for j in range(5 + key):

if flag < pass\_len:

matrix[i][j] = word[flag]

flag += 1

matrix = matrix.transpose()

result = ""

for i in range(5 + key):

for j in range(key):

result += matrix[i][j]

return result

# Считывание логина и пароля

def log():

log = ui.plainTextEditLog.toPlainText()

return log

def pas():

pas = ui.plainTextEditPass.toPlainText()

return pas

# Очистка полей логина и пароля

def clear():

ui.plainTextEditLog.setPlainText("")

ui.plainTextEditPass.setPlainText("")

# Авторизации + работа личного кабинета

def click\_auth():

with open('credentials.txt', 'r') as f:

flag = 0

credentials = log() + pas()

for row in f:

if row == Encrypt(credentials) + "\n":

flag = 1

break

clear()

if flag == 1:

MainWindow.close()

main()

else:

message("Неверный логин или пароль, попробуйте еще раз")

# Регистрация

def click\_reg():

with open('login.txt', 'r') as f:

for row in f:

flag = 0

if (Encrypt(log()) + "\n") == row:

message("Такой логин уже существует")

flag = 1

break

if flag == 0:

checkLog = len(log())

checkPas = len(pas())

if checkLog < 16 and checkLog > 4 and checkPas < 16 and checkPas > 4:

space = " "

if space in log() or space in pas():

message("Недопустимый символ в логине или пароле: Пробел")

else:

with open('login.txt', 'a') as f:

f.write(Encrypt(log()) + "\n")

with open('credentials.txt', 'a') as d:

credentials = log() + pas()

d.write(Encrypt(credentials) + '\n')

clear()

else:

message("Некорректные логин или пароль, попробуйте еще раз")

ui.pushButtonRegistration.clicked.connect(click\_reg)

ui.pushButtonLogin.clicked.connect(click\_auth)

# Подгрузка изображений

def loadImages():

pieces = ["wp", "bp"]

for piece in pieces:

IMAGES[piece] = p.transform.scale(p.image.load("images/" + piece + ".png"), (SQ\_SIZE, SQ\_SIZE))

# Состояние игры

class GameState:

def \_\_init\_\_(self):

self.board = [

["--", "wp", "wp", "wp", "wp", "wp", "--"],

["--", "--", "--", "--", "--", "--", "bp"],

["--", "--", "--", "--", "--", "--", "bp"],

["--", "--", "--", "--", "--", "--", "bp"],

["--", "--", "--", "--", "--", "--", "bp"],

["--", "--", "--", "--", "--", "--", "bp"],

["--", "--", "--", "--", "--", "--", "--"],

]

self.whiteToMove = True # Текущий ход

self.moveLog = [] # История ходов

self.gameOver = False # Переменная окончания игры

self.whiteScore = 0 # Очки белых

self.blackScore = 0 # Очки черных

self.pieceEscaped = False # Отслеживание сбегающих шашек

self.moveForward = False # Отслеживание результативных ходов

# Получение допустимых ходов

def getValidMoves(self):

moves = []

for r in range(len(self.board)):

for c in range(len(self.board[r])):

turn = self.board[r][c][0]

if (turn == 'w' and self.whiteToMove) or (turn == 'b' and not self.whiteToMove):

self.getPieceMoves(r, c, moves)

return moves

# Получение ходов фигуры

def getPieceMoves(self, r, c, moves):

if self.whiteToMove:

if r + 1 <= 7:

if self.board[r + 1][c] == "--":

moves.append(Move((r, c), (r + 1, c), self.board))

if c + 1 <= 6:

if self.board[r][c + 1] == "--":

moves.append(Move((r, c), (r, c + 1), self.board))

if c - 1 >= 1:

if self.board[r][c - 1] == "--":

moves.append(Move((r, c), (r, c - 1), self.board))

else:

if c - 1 >= 0:

if self.board[r][c - 1] == "--":

moves.append(Move((r, c), (r, c - 1), self.board))

if r + 1 <= 5:

if self.board[r + 1][c] == "--":

moves.append(Move((r, c), (r + 1, c), self.board))

if r - 1 >= 0:

if self.board[r - 1][c] == "--":

moves.append(Move((r, c), (r - 1, c), self.board))

# Пробный ход

def testMove(self, move):

self.board[move.startRow][move.startCol] = "--"

self.board[move.endRow][move.endCol] = move.pieceMoved

self.moveLog.append(move)

self.whiteToMove = not self.whiteToMove

if move.isPieceEscape:

self.board[move.endRow][move.endCol] = "--"

self.pieceEscaped = True

if move.isMoveForward:

self.moveForward = True

# Ход игрока

def makePlayerMove(self, move):

self.board[move.startRow][move.startCol] = "--"

self.board[move.endRow][move.endCol] = move.pieceMoved

self.moveLog.append(move)

self.whiteToMove = not self.whiteToMove

if move.isPieceEscape:

self.board[move.endRow][move.endCol] = "--"

if self.whiteToMove:

self.blackScore += 1

if self.blackScore == 5:

self.gameOver = True

else:

self.whiteScore += 1

if self.whiteScore == 5:

self.gameOver = True

# Ход компьютера

def makeComputerMove(self, move):

self.board[move.startRow][move.startCol] = "--"

self.board[move.endRow][move.endCol] = move.pieceMoved

self.moveLog.append(move)

self.whiteToMove = not self.whiteToMove

if move.isPieceEscape:

self.board[move.endRow][move.endCol] = "--"

if self.whiteToMove:

self.blackScore += 1

if self.blackScore == 5:

self.gameOver = True

else:

self.whiteScore += 1

if self.whiteScore == 5:

self.gameOver = True

# Удаление последнего хода

def undoMove(self):

if len(self.moveLog) != 0:

move = self.moveLog.pop()

self.board[move.startRow][move.startCol] = move.pieceMoved

self.board[move.endRow][move.endCol] = "--"

self.whiteToMove = not self.whiteToMove

class Move():

ranksToRows = {"1": 6, "2": 5, "3": 4, "4": 3, "5": 2, "6": 1, "7": 0}

rowsToRanks = {v: k for k, v in ranksToRows.items()}

filesToCols = {"a": 0, "b": 1, "c": 2, "d": 3, "e": 4, "f": 5, "g": 6}

colsToFiles = {v: k for k, v in filesToCols.items()}

def \_\_init\_\_(self, startSq, endSq, board):

self.startRow = startSq[0]

self.startCol = startSq[1]

self.endRow = endSq[0]

self.endCol = endSq[1]

self.pieceMoved = board[self.startRow][self.startCol]

self.isPieceEscape = False

self.isMoveForward = False

if (self.pieceMoved == 'wp' and self.endRow == 6) or (self.pieceMoved == 'bp' and self.endCol == 0):

self.isPieceEscape = True

if (self.pieceMoved == 'wp' and self.endRow > self.startRow) or (self.pieceMoved == 'bp' and self.endCol < self.startCol):

self.isMoveForward = True

self.moveID = self.startRow \* 1000 + self.startCol \* 100 + self.endRow \* 10 + self.endCol

# Переопределение метода equals для правильного сравнения ходов

def \_\_eq\_\_(self, other):

if isinstance(other, Move):

return self.moveID == other.moveID

return False

# Получение координат по нотации Эдвардса

def getBoardNotation(self):

return self.getRankFile(self.startRow, self.startCol) + self.getRankFile(self.endRow, self.endCol)

def getRankFile(self, r, c):

return self.colsToFiles[c] + self.rowsToRanks[r]

# Основной цикл

def main():

p.init()

p.display.set\_caption('Компьютерная логическая игра "Доджем"')

p.display.set\_icon(p.image.load("images/icon.png"))

screen = p.display.set\_mode((WIDTH, HEIGHT))

clock = p.time.Clock()

screen.fill(p.Color("white"))

gs = GameState()

validMoves = gs.getValidMoves()

moveMade = False

loadImages()

running = True

sqSelected = ()

playerClicks = []

firstPlayer = True

secondPlayer = False

while running:

humanTurn = (gs.whiteToMove and firstPlayer) or (not gs.whiteToMove and secondPlayer)

for e in p.event.get():

if e.type == p.QUIT:

running = False

elif e.type == p.MOUSEBUTTONDOWN:

if not gs.gameOver and humanTurn:

location = p.mouse.get\_pos()

col = location[0] // SQ\_SIZE

row = location[1] // SQ\_SIZE

if sqSelected == (row, col):

sqSelected = ()

playerClicks = []

else:

sqSelected = (row, col)

playerClicks.append(sqSelected)

if len(playerClicks) == 2:

move = Move(playerClicks[0], playerClicks[1], gs.board)

if move in validMoves:

gs.makePlayerMove(move)

moveMade = True

sqSelected = ()

playerClicks = []

else:

playerClicks = [sqSelected]

if not gs.gameOver and not humanTurn:

computerMove = findOptimalMove(gs, validMoves)

if computerMove is not None:

gs.makeComputerMove(computerMove)

gs.pieceEscaped = False

gs.moveForward = False

moveMade = True

else:

moveMade = True

if moveMade:

validMoves = gs.getValidMoves()

moveMade = False

drawGameState(screen, gs, validMoves, sqSelected)

clock.tick(MAX\_FPS)

p.display.flip()

# Вывод уведомления об окончании игры

if gs.gameOver:

if gs.whiteToMove:

msg\_box = tk.messagebox.askquestion('Победа Черных',

'Выйти из приложения?',

icon='warning')

if msg\_box == 'yes':

p.display.quit()

p.quit()

else:

gs = GameState()

validMoves = gs.getValidMoves()

sqSelected = ()

playerClicks = []

moveMade = False

else:

msg\_box = tk.messagebox.askquestion('Победа Белых',

'Выйти из приложения?',

icon='warning')

if msg\_box == 'yes':

p.display.quit()

p.quit()

else:

p.display.set\_caption('Компьютерная логическая игра "Доджем"')

gs = GameState()

validMoves = gs.getValidMoves()

sqSelected = ()

playerClicks = []

moveMade = False

# Выделение квадратов с доступными ходами

def highlightSquares(screen, gs, validMoves, sqSelected):

if sqSelected != ():

r, c = sqSelected

if gs.board[r][c][0] == ('w' if gs.whiteToMove else 'b'):

s = p.Surface((SQ\_SIZE, SQ\_SIZE))

s.set\_alpha(50)

s.fill(p.Color('Blue'))

screen.blit(s, (c\*SQ\_SIZE, r\*SQ\_SIZE))

s.fill(p.Color("Yellow"))

for move in validMoves:

if move.startRow == r and move.startCol == c:

screen.blit(s, (move.endCol\*SQ\_SIZE, move.endRow\*SQ\_SIZE))

# Нахождение оптимального хода для компьютера

def findOptimalMove(gs, validMoves):

score = 0

maxScore = -fullEscape

bestMove = None

moves = validMoves

random.shuffle(moves)

for playerMove in moves:

gs.testMove(playerMove)

if gs.moveForward:

score = 1

if gs.pieceEscaped:

score = 2

if gs.gameOver:

score = fullEscape

if score > maxScore:

maxScore = score

bestMove = playerMove

gs.undoMove()

return bestMove

# Отрисовка графики при текущем состоянии игры

def drawGameState(screen, gs, validMoves, sqSelected):

drawBoard(screen)

highlightSquares(screen, gs, validMoves, sqSelected)

drawPieces(screen, gs.board)

# Отрисовка игровой доски

def drawBoard(screen):

colors = [p.Color("white"), p.Color("gray")]

for r in range(DIMENSION):

for c in range(DIMENSION):

color = colors[((r+c) % 2)]

p.draw.rect(screen, color, p.Rect(c\*SQ\_SIZE, r\*SQ\_SIZE, SQ\_SIZE, SQ\_SIZE))

# Отрисовка фигур

def drawPieces(screen, board):

for r in range(DIMENSION):

for c in range(DIMENSION):

piece = board[r][c]

if piece != "--":

screen.blit(IMAGES[piece], p.Rect(c\*SQ\_SIZE, r\*SQ\_SIZE, SQ\_SIZE, SQ\_SIZE))

sys.exit(app.exec())