ФГБОУ ВО «МОСКОВСКИЙ ПОЛИТЕХНИЧЕСКИЙ УНИВЕРСИТЕТ»

Лабораторная работа № 10

По теме: «Выполнение проекта»

Задание 1

По дисциплине:
«Основы программирования»

Выполнил
Студент 1 курса
Переверзев И. Д.
Донской Н. А.

Проверил
_____ Никишина И. Н.

МОСКВА 2022

Цель работы

Закрепление теоретических знаний по дисциплине «Основы программирования», а также практических навыков по программированию на языке Python путем написания проекта на тему: «Разработка идеальный движка для игры крестики-нолики».

Постановка задачи

Используя язык программирования Python, а также дополнительные графические библиотеки для него, написать программу, позволяющую пользователю играть в крестики-нолики с искусственным интеллектом.

Теоретическое введение

Инструментарий

Для выполнения поставленной цели нами были выбраны следующие инструменты:

- Язык программирования Python – основной инструмент
- Графический модуль pygame — библиотека для написания приложений с графическим интерфейсом на Python, преимущественно используется для написания игр
- Модуль pyistaller – модуль для компилирования файла .py в исполняемый файл .exe

Основа искусственного интеллекта

Для основы искусственного интеллекта был выбран алгоритм «Минимакс». Вкратце этот алгоритм можно описать следующим образом: программа просчитывает все возможные ходы «компьютерного игрока». Каждому ходу присваивается некоторое числовое значение. И чем больше данное значение, тем предпочтительнее считается соответствующий ход.

Описание реализации программы

Модуль pygame предоставляет разработчику возможность создавать приложения и игры с графическим интерфейсом. Это достигается путем создания отдельного окна и размещения на нем различных графических элементов (ГЭ), созданных самим разработчиком. В качестве примера таких элементов можно привести кнопки, поля для ввода, различные надписи и так далее.

К основным возможностям модуля pygame относятся:

- Отслеживание положения курсора мыши
- Рисование различных графических примитивов, из которых составляются все основные элементы интерфейса
- Создание дополнительных поверхностей, объединяющих в себе несколько ГЭ
- Наложение этих поверхностей на главное окно
- Обработка событий (нажатие на клавишу клавиатуры, на кнопку мыши, на кнопку выхода)
- работа с музыкой и звуковыми эффектами
- Добавление и использование в качестве ГЭ обычных растровых изображений

- Создание различных таймеров для отслеживания продолжительности некоторых пользовательских событий

Размещение и позиционирование графических элементов происходит относительно левого верхнего угла окна программы. Стоит отметить, что обсуждаемый модуль позволяет создавать и использовать так называемые «Поверхности», представляющие собой холсты, объединяющие несколько графических элементов. Такие холсты можно создавать в относительно большом количестве. И иногда их применение позволяет сильно упростить разработку программы.

Отрисовка всех поверхностей и ГЭ элементов осуществляется внутри главного исполняемого цикла. На каждой итерации отрисовка производится с нуля. Такой подход позволяет проще всего обновлять содержимое главного окна. Кроме того, можно регулировать частоту кадров, с которой производится отрисовка.

При написании кода использовалось Объектно Ориентированное Программирование (ООП) для описания свойств и поведения различных графических элементов, искусственного интеллекта, игрового и главного окон и так далее.

Листинг программы

Главный исполняемый файл

```
from pygame import display, time, event, mouse, init, image,
transform, quit
from settings import *
from modules.bot_for_game import *
from random import choice
from sys import exit
from modules.gui_elems import *
from modules.game_elems import *
init()
def play_symbol_sound(symbol):
    if symbol == 'O':
        SHIELD.play(0)
    else:
        SWORD_1.play(0)
        SWORD_2.play(0)
class Game:
    def __init__(self):
        self.window = display.set_mode([WIDTH, HEIGHT])
        display.set_caption("Крестики и нолики")
        display.set_icon(image.load('src/img/swords.png'))
        self.screen = surface.Surface([WIDTH, HEIGHT])
        self.clock = time.Clock()
    def run(self):
        start_screen = StartScreen()
        while True:
            start_screen.run()
            game_screen = GameScreen(*start_screen.get_chars())
            game_screen.run()
```

```
class GameScreen(Game):
    def __init__(self, hu_char: str, ai_char: str):
        super(GameScreen, self).__init__()
        self.field = Field(3, 3, 300, 55, self.screen)
        self.rule_place = RulePlace(780, 55, CELL_SIZE*1.9,
CELL_SIZE*3, self.screen)
        self.btn_reset = Button(20, 350, CELL_SIZE*1.66,
CELL_SIZE*0.4, 20, 'Начать новую игру', self.screen)
        self.btn_quit = Button(20, 445, CELL_SIZE * 1.66, CELL_SIZE
* 0.4, 20, 'Выйти из игры', self.screen)
        self.btn_return = Button(20, 10, CELL_SIZE * 1.66,
CELL_SIZE*0.2, 16,
                                 'Вернуться на главный экран',
self.screen)
        self.counter = Counter(20, 55, CELL_SIZE*1.66,
CELL_SIZE*0.85, self.screen)
        self.label_cur_player = Label(20, HEIGHT // 3, 20, 'Сейчас
ходит: игрок', self.screen)
        self.contrast_screen = surface.Surface([WIDTH, HEIGHT],
SRCALPHA)
        self.bg = image.load('src/img/game_screen_bg_pixel.png')
        self.bg = transform.scale(self.bg, (WIDTH, HEIGHT))
        self.font_header = font.Font("src/fonts/Sonic 1 Title Screen
Filled.ttf", 75)
        self.buttons = [
            self.btn_quit,
            self.btn_return,
            self.btn_reset
        ]
        self.comp_turn = False
        self.was_win = False
        self.bot_first = False
        self.winner = ""
        self.winner_text = None
        self.winner_text_rect = None
        self.want_back = False
        self.was_bot_turn = False
        self.begin = True
        self.can_click = True
        self.click_time = 0
        self.bot = Bot(hu_char, ai_char)
        self.winner_text_color = (0, 0, 0)
    def update_click(self):
        if not self.can_click:
            mouse.set_cursor(NORM_CURSOR)
            cur_time = time.get_ticks()
            if cur_time - self.click_time >= 1000:
                self.can_click = True
    def rising(self):
```

```
        for i in range(26):
            self.contrast_screen.fill((0, 0, 0, 250 - 10*i))
            mixer.music.set_volume(0 + 0.1 * i)
            self.window.blit(self.screen, (0, 0))
            self.window.blit(self.contrast_screen, (0, 0))
            self.clock.tick(FPS)
            display.update()
    def refresh(self):
        self.was_win = False
        self.comp_turn = False
        self.field.refresh_field()
        if self.bot.ai_char == 'X':
            self.field.query = 1
            self.bot_first = True
        self.change_cur_player()
    def attenuation(self):
        for i in range(1, 26):
            self.contrast_screen.fill((0, 0, 0, 10*i))
            mixer.music.set_volume(1 - 0.1 * i)
            self.window.blit(self.screen, (0, 0))
            self.window.blit(self.contrast_screen, (0, 0))
            self.clock.tick(FPS)
            display.update()
    def change_cur_player(self):
        if self.field.query % 2 == 0:
            self.label_cur_player.change_text('Сейчас ходит: игрок')
        else:
            self.label_cur_player.change_text('Сейчас ходит: бот')
    def gameplay(self):
        self.refresh()
        mouse_pos = mouse.get_pos()
        run = True
        while run:
            if self.comp_turn:
                self.update_game()
                time.delay(1000)
            for evt in event.get():
                if evt.type == QUIT:
                    self.attenuation()
                    exit_()
                if evt.type == MOUSEMOTION and not self.begin:
                    mouse_pos = mouse.get_pos()
                    hovers =
any([self.buttons[i].check_move(mouse_pos) for i in
range(len(self.buttons))])
                    if self.field.check_move(mouse_pos) or hovers:
                        mouse.set_cursor(FINGER_CURSOR)
```

```
                    else:
                        mouse.set_cursor(NORM_CURSOR)
                if evt.type == MOUSEBUTTONDOWN and self.can_click:
                    if self.btn_reset.check_press(mouse_pos):
                        run = False
                        self.can_click = False
                        self.click_time = time.get_ticks() + 1000
                        if not self.was_win:
                            self.counter.update_counters('computer')
                            self.change_cur_player()
                    if self.btn_quit.check_press(mouse_pos):
                        self.attenuation()
                        exit_()
                    if self.btn_return.check_press(mouse_pos):
                        run = False
                        self.want_back = True
                    if not self.was_win:
                        if
self.field.check_press_on_cells(mouse_pos, self.bot.hu_char):
                            play_symbol_sound(self.bot.hu_char)
                            self.change_cur_player()
                            self.comp_turn = True
                            self.can_click = False
                            self.click_time = time.get_ticks() - 500
            self.update_click()
            self.screen.blit(self.bg, (0, 0))
            self.field.draw()
            self.rule_place.draw()
            self.counter.draw()
            self.label_cur_player.draw()
            if self.was_win:
                self.screen.blit(self.contrast_screen, (0, 0))
                self.screen.blit(self.winner_text,
self.winner_text_rect)
            if self.was_bot_turn:
                play_symbol_sound(self.bot.ai_char)
                self.was_bot_turn = False
            self.btn_reset.draw()
            self.btn_quit.draw()
            self.btn_return.draw()
            if self.begin:
                self.rising()
                self.begin = False
            self.window.blit(self.screen, (0, 0))
            self.clock.tick(FPS)
            display.update()
            if self.bot_first:
```

```
                x = self.field.rect.x + CELL_SIZE * choice([0, 2])
                y = self.field.rect.y + CELL_SIZE * choice([0, 2])
                self.field.check_press_on_cells([x, y],
self.bot.hu_char)
                self.was_bot_turn = True
                self.change_cur_player()
                self.bot_first = False
                time.delay(1000)
    def run(self):
        mixer.music.load('src/audio/game_screen_theme.mp3')
        mixer.music.play(-1)
        while True:
            self.gameplay()
            if self.want_back:
                self.attenuation()
                break
    def update_game(self):
        move = self.bot.computer_position(self.field.board)
        if move is not None:
            row, col = move
            x = self.field.rect.x + CELL_SIZE * col
            y = self.field.rect.y + CELL_SIZE * row
            self.field.check_press_on_cells([x, y],
self.bot.hu_char)
            self.was_bot_turn = True
        if piece(self.field.board, self.bot.hu_char,
self.bot.ai_char):
            self.winner = 'piece'
            self.was_win = True
        if check_win(self.field.board, self.bot.hu_char):
            self.was_win = True
            self.winner = 'human'
        if check_win(self.field.board, self.bot.ai_char):
            self.was_win = True
            self.winner = 'computer'
        if self.was_win:
            self.counter.update_counters(self.winner)
            self.render_win_text()
        self.change_cur_player()
        self.comp_turn = False
    def render_win_text(self):
        if self.winner == 'human':
            message = 'ПОБЕДА'
        elif self.winner == 'computer':
            message = 'ПОРАЖЕНИЕ'
        else:
            message = 'НИЧЬЯ'
```

```
        self.contrast_screen.fill((255, 255, 255, 100))
        self.winner_text = self.font_header.render(message, True,
self.winner_text_color)
        self.winner_text_rect =
self.winner_text.get_rect(center=[WIDTH//2, HEIGHT // 3])
class StartScreen(Game):
    def __init__(self):
        super(StartScreen, self).__init__()
        self.header = Label(0, 0, 65, "Крестики и нолики",
self.screen)
        self.btn_next = Button(0, 0, 300, 70, 25, "Начать игру",
self.screen)
        self.btn_quit = Button(0, 0, 300, 70, 25, "Выйти из игры",
self.screen)
        self.question = Label(10, 10, 21, 'Каким знаком вы хотите
играть?', self.screen)
        self.bg = image.load('src/img/start_screen_bg_pixel.png')
        self.bg = transform.scale(self.bg, (WIDTH, HEIGHT))
        self.attenuation_screen = surface.Surface([WIDTH, HEIGHT],
SRCALPHA)
        self.btn_quit.set_center(WIDTH // 1.86, HEIGHT // 1.3)
        self.btn_next.set_center(WIDTH // 1.86, HEIGHT // 1.7)
        self.header.rect.center = [WIDTH // 1.86, HEIGHT // 9]
        self.question.rect.center = [WIDTH // 1.86, HEIGHT // 2.8]
        self.char_choice = [RadioButton(WIDTH // 2.3, HEIGHT //
2.35, 18, 10, 'Крестики', self.screen),
                            RadioButton(WIDTH//1.8, HEIGHT//2.35,
18, 10, 'Нолики', self.screen)]
        self.char_choice[0].active = True
        self.buttons = [self.btn_next, self.btn_quit,
*self.char_choice]
        self.begin = False
    def rising(self):
        for i in range(25):
            self.attenuation_screen.fill((0, 0, 0, 250 - 10 * i))
            mixer.music.set_volume(0 + 0.1 * i)
            self.window.blit(self.screen, (0, 0))
            self.window.blit(self.attenuation_screen, (0, 0))
            self.clock.tick(FPS // 1.5)
            display.update()
    def attenuation(self):
        for i in range(1, 26):
            self.attenuation_screen.fill((0, 0, 0, 10*i))
            mixer.music.set_volume(1 - 0.1 * i)
            self.window.blit(self.screen, (0, 0))
            self.window.blit(self.attenuation_screen, (0, 0))
            self.clock.tick(FPS // 1.5)
            display.update()
    def run(self):
```

```
        self.begin = True
        self.attenuation_screen.fill((0, 0, 0))
        mixer.music.load('src/audio/start_screen_theme.mp3')
        mixer.music.play(-1)
        mouse_pos = [0, 0]
        run = True
        while run:
            for evt in event.get():
                if evt.type == QUIT:
                    self.attenuation()
                    exit_()
                if evt.type == MOUSEMOTION and not self.begin:
                    mouse_pos = mouse.get_pos()
                    hovers =
any([self.buttons[i].check_move(mouse_pos) for i in
range(len(self.buttons))])
                    if hovers:
                        mouse.set_cursor(FINGER_CURSOR)
                    else:
                        mouse.set_cursor(NORM_CURSOR)
                if evt.type == MOUSEBUTTONDOWN:
                    if self.char_choice[0].check_press(mouse_pos):
                        self.update_char(0)
                    if self.char_choice[1].check_press(mouse_pos):
                        self.update_char(1)
                    if self.btn_next.check_press(mouse_pos):
                        run = False
                        mouse.set_cursor(NORM_CURSOR)
                        GATE_UNLOCK.play(0)
                    if self.btn_quit.check_press(mouse_pos):
                        self.attenuation()
                        exit_()
            self.screen.blit(self.bg, (0, 0))
            self.header.draw()
            self.question.draw()
            [radio.update() for radio in self.char_choice]
            self.btn_next.draw()
            self.btn_quit.draw()
            if self.begin:
                self.rising()
                self.begin = False
            self.window.blit(self.screen, (0, 0))
            self.clock.tick(FPS)
            display.update()
        time.delay(3000)
        self.attenuation()
    def update_char(self, index):
```

```
        self.char_choice[index].active = True
        self.char_choice[(index + 1) % 2].active = False
    def get_chars(self):
        if self.char_choice[0].active:
            return 'X', 'O'
        else:
            return 'O', 'X'
def exit_():
    file = open('src/count.txt', mode='w', encoding='utf-8')
    file.write('win=0\nlose=0\npiece=0')
    file.close()
    quit()
    exit()
if __name__ == '__main__':
    game = Game()
    game.run()
```

Файл с графическими элементами

```
from pygame import draw, surface, rect, font, SRCALPHA
from settings import CLICK
class RadioButton:
    def __init__(self, x: int, y: int, font_size: int, radius: int,
text: str, screen: surface.Surface):
        self.rect = rect.Rect(x, y, radius*2, radius*2)
        self.font = font.Font('src/fonts/centurygothic_bold.ttf',
font_size)
        self.text = Label(self.rect.right + 10, y, font_size, text,
screen)
        self.screen = screen
        self.radius = radius
        self.active = False
    def update(self):
        self.draw()
        self.text.draw()
    def draw(self):
        draw.ellipse(self.screen, (200, 200, 200), self.rect)
        if self.active:
            draw.circle(self.screen, (30, 30, 30), self.rect.center,
self.radius // 2)
    def check_press(self, mouse_pos: [int, int]):
        if self.rect.collidepoint(mouse_pos) or
self.text.rect.collidepoint(mouse_pos):
            CLICK.play(0)
            return True
        return False
    def check_move(self, mouse_pose: [int, int]):
        if mouse_pose is not None and
```

```
self.rect.collidepoint(mouse_pose):
            return True
        else:
            return False
class Label:
    def __init__(self, x: int, y: int, font_size: int, text: str,
screen: surface.Surface):
        self.font = font.Font('src/fonts/centurygothic_bold.ttf',
font_size)
        self.screen = screen
        self.text_color = (255, 255, 255)
        self.text = self.font.render(text, True, self.text_color)
        self.rect = self.text.get_rect(x=x, y=y)
    def draw(self):
        self.screen.blit(self.text, self.rect)
    def change_text(self, text: str):
        self.text = self.font.render(text, True, self.text_color)
class Button:
    def __init__(self, x: int, y: int, w: int, h: int, font_size:
int, text: str, screen: surface.Surface):
        self.rect = rect.Rect(x, y, w, h)
        self.screen = screen
        self.font = font.Font('src/fonts/centurygothic_bold.ttf',
font_size)
        self.text_color = (255, 255, 255)
        self.bg_color = (0, 0, 0, 210)
        self.bg_color_hover = (100, 100, 100, 200)
        self.bg = surface.Surface([self.rect.w, self.rect.h],
SRCALPHA)
        self.text_orig = text
        self.text = self.font.render(text, True, self.text_color)
        self.text_rect = self.text.get_rect(center=self.rect.center)
        self.hover = False
    def draw(self):
        self.bg.fill(self.bg_color)
        if self.hover:
            self.bg.fill(self.bg_color_hover)
        self.screen.blit(self.bg, self.rect)
        self.screen.blit(self.text, self.text_rect)
    def set_center(self, x: int, y: int):
        self.rect.center = [x, y]
        self.text_rect.center = [x, y]
    def check_move(self, mouse_pose: [int, int]):
        if mouse_pose is not None and
self.rect.collidepoint(mouse_pose):
            self.hover = True
            return True
        else:
```

```
            self.hover = False
            return False
    def check_press(self, mouse_pose: [int, int]):
        if self.rect.collidepoint(mouse_pose):
            CLICK.play(0)
            return True
        else:
            return False
```

Файл с игровыми элементами (поле, область с правилами, счетчик)

```
from pygame import draw, surface, rect, font, image, transform,
SRCALPHA
from settings import CELL_SIZE
from modules.bot_for_game import EMPTY_CHAR
def get_computer_char(user_char):
    return 'O' if user_char == 'X' else 'X'
class Field:
    def __init__(self, rows: int, cols: int, x: int, y: int, screen:
surface.Surface):
        self.screen = screen
        self.rect = rect.Rect(x, y, CELL_SIZE * rows, CELL_SIZE *
cols)
        self.field = [[0] * cols for _ in range(rows)]
        self.cells_coords = [[[x + CELL_SIZE * col, y + CELL_SIZE *
row]
                              for col in range(cols)] for row in
range(rows)]
        self.border_color = (100, 100, 100)
        self.cells = [[Cell(col[0], col[1], self.screen) for col in
row] for row in self.cells_coords]
        self.board = [[EMPTY_CHAR] * 3 for _ in range(3)]
        self.contrast_screen = surface.Surface([self.rect.w,
self.rect.h], SRCALPHA)
        self.contrast_screen.fill((0, 0, 0, 150))
        self.query = 0
    def refresh_field(self):
        for row in range(len(self.cells)):
            for col in range(len(self.cells[row])):
                self.cells[row][col].refresh()
        self.query = 0
        self.board = [[EMPTY_CHAR] * 3 for _ in range(3)]
    def change_query(self):
        self.query = (self.query + 1) % 2
    def update_board(self):
        for row in range(len(self.cells)):
            for col in range(len(self.cells[row])):
                self.board[row][col] = self.cells[row][col].char
```

```
    def check_move(self, mouse_pos: [int, int]):
        hovers = []
        for row in range(len(self.cells)):
            for col in range(len(self.cells[row])):
                hovers += [self.cells[row]
[col].check_move(mouse_pos)]
        return any(hovers)
    def draw(self):
        self.screen.blit(self.contrast_screen, self.rect)
        for row in range(len(self.cells)):
            for col in range(len(self.cells[row])):
                self.cells[row][col].draw()
        for _ in self.cells_coords:
            for row in range(1, 3):
                x1, y1 = self.cells_coords[row][0]
                draw.line(self.screen, self.border_color, [x1, y1],
[self.rect.right - 1, y1], width=4)
            for col in range(1, 3):
                x1, y1 = self.cells_coords[0][col]
                draw.line(self.screen, self.border_color, [x1, y1],
[x1, self.rect.bottom - 1], width=4)
        draw.rect(self.screen, self.border_color, self.rect,
width=3)
    def check_press_on_cells(self, mouse_pos, hu_char):
        for row in range(len(self.cells)):
            for col in range(len(self.cells[row])):
                if self.cells[row][col].check_press(mouse_pos,
self.query, hu_char):
                    self.update_board()
                    self.change_query()
                    return True
        return False
class Cell:
    def __init__(self, x: int, y: int, screen: surface.Surface):
        self.x = x
        self.y = y
        self.screen = screen
        self.char = EMPTY_CHAR
        self.hover = False
        self.rect = rect.Rect(x, y, CELL_SIZE, CELL_SIZE)
        self.font = font.Font(None, 100)
        self.cross =
transform.scale(image.load('src/img/swords.png'), [self.rect.w-10,
self.rect.h-10])
        self.circle =
transform.scale(image.load('src/img/shield.png'), [self.rect.w-10,
self.rect.h-10])
        self.image_rect = rect.Rect(x, y, CELL_SIZE - 10, CELL_SIZE
- 10)
```

```
        self.image_rect.center = self.rect.center
        self.contrast_screen = surface.Surface([self.rect.w,
self.rect.h], SRCALPHA)
        self.contrast_screen.fill((230, 230, 230, 30))
        self.slovar = {'X': self.cross, 'O': self.circle}
    def refresh(self):
        self.char = EMPTY_CHAR
        self.hover = False
    def draw(self):
        if self.hover:
            self.screen.blit(self.contrast_screen, self.rect)
        if self.char != EMPTY_CHAR:
            self.screen.blit(self.slovar[self.char],
self.image_rect)
    def check_move(self, mouse_pos: [int, int]):
        if mouse_pos is not None:
            if self.rect.collidepoint(mouse_pos):
                self.hover = True
                return True
            else:
                self.hover = False
                return False
    def check_press(self, mouse_pos: [int, int], query: int,
hu_char: str):
        if self.rect.collidepoint(mouse_pos):
            if self.char == EMPTY_CHAR:
                if query % 2 == 0:
                    self.char = hu_char
                else:
                    self.char = get_computer_char(hu_char)
                return True
        return False
class Counter:
    def __init__(self, x: int, y: int, w: int, h: int, screen:
surface.Surface):
        self.rect = rect.Rect(x, y, w, h)
        self.screen = screen
        self.font = font.Font('src/fonts/centurygothic_bold.ttf',
20)
        self.rects = [rect.Rect(0, 0, 0, 0) for _ in range(6)]
        self.contrast_screen = surface.Surface([self.rect.w,
self.rect.h], SRCALPHA)
        self.contrast_screen.fill((0, 0, 0, 150))
        self.white_contrast = [surface.Surface([CELL_SIZE * 0.5,
CELL_SIZE * 0.175], SRCALPHA) for _ in range(3)]
        [self.white_contrast[i].fill((255, 255, 255, 40)) for i in
range(3)]
        self.white_rects = [rect.Rect(0, 0, CELL_SIZE * 0.5,
CELL_SIZE * 0.175) for _ in range(3)]
```

```
        self.text_color = (255, 255, 255)
        self.border_color = (100, 100, 100)
        self.win, self.lose, self.piece = self.read_file()
        self.text = []
        self.rects = []
        self.render_text()
    def read_file(self):
        file = open('src/count.txt', mode='r', encoding='utf-8')
        data = [int(line.split('=')[1]) for line in
file.readlines()]
        file.close()
        return data
    def write_file(self):
        file = open('src/count.txt', mode='w', encoding='utf-8')

file.write(f'win={self.win}\nlose={self.lose}\npiece={self.piece}')
        file.close()
    def render_text(self):
        self.text = [
            self.font.render("Победы:", True, self.text_color),
            self.font.render("Поражения:", True, self.text_color),
            self.font.render("Ничьи:", True, self.text_color),
            self.font.render(str(self.win), True, self.text_color),
            self.font.render(str(self.lose), True, self.text_color),
            self.font.render(str(self.piece), True, self.text_color)
        ]
        self.rects = [line.get_rect() for line in self.text]
        for i in range(3):
            self.rects[i].left = self.rect.left + 10
            self.rects[i].top = self.rect.top + 20 + 30 * i
        for i in range(3, 6):
            self.rects[i].right = self.rect.right - 15
            self.white_rects[i - 3].right = self.rect.right - 10
            self.white_rects[i-3].top = self.rects[i].top =
self.rect.top + 20 + 30 * (i - 3)
    def update_counters(self, result):
        if result == 'human':
            self.win += 1
        elif result == 'computer':
            self.lose += 1
        else:
            self.piece += 1
        self.write_file()
        self.render_text()
    def draw(self):
        self.screen.blit(self.contrast_screen, self.rect)
        for j in range(3):
```

```
            self.screen.blit(self.white_contrast[j],
self.white_rects[j])
        draw.rect(self.screen, self.border_color, self.rect,
width=3)
        for i in range(6):
            self.screen.blit(self.text[i], self.rects[i])
class RulePlace:
    def __init__(self, x: int, y: int, w: int, h: int, screen:
surface.Surface):
        self.rect = rect.Rect(x, y, w, h)
        self.screen = screen
        self.font_header =
font.Font("src/fonts/centurygothic_bold.ttf", 20)
        self.font = font.Font("src/fonts/centurygothic_bold.ttf",
16)
        self.text_fish = ['• Игроки по очереди ставят',
                          '   на свободные клетки поля',
                          '   3×3 знаки (один всегда ',
                          '   крестики, другой всегда ',
                          '   нолики).',
                          '',
                          '• Первый ход делает игрок,',
                          '   ставящий крестики. ',
                          '',
                          '• Игра продолжается пока не ',
                          '   заполнятся все клетки, или ',
                          '   один из участников не сделает ',
                          '   цепочку из трёх одинаковых ',
                          '   символов. ',
                          '',
                          '• Причём ряд идёт по ',
                          '   горизонтали, вертикали',
                          '   или диагонали.']
        self.text_header = self.font_header.render('Правила игры',
True, (255, 255, 255))
        self.text = []
        self.contrast_screen = surface.Surface([self.rect.w,
self.rect.h], SRCALPHA)
        self.contrast_screen.fill((0, 0, 0, 150))
        self.border_color = (100, 100, 100)
        self.render_text()
    def render_text(self):
        for line in self.text_fish:
            self.text += [self.font.render(line, True, (255, 255,
255))]
    def draw(self):
        self.screen.blit(self.contrast_screen, self.rect)
        draw.rect(self.screen, self.border_color, self.rect,
width=3)
```

```
        x, y = self.rect[:2]
        x += 10
        y += 10
        self.screen.blit(self.text_header, [x, y])
        y += 20
        for line in self.text:
            y += 20
            self.screen.blit(line, [x, y])
```

Файл с логикой искусственного интеллекта

```
import sys
EMPTY_CHAR = '_'
AI_TURN = True
HU_TURN = False
class Bot:
    def __init__(self, hu_char, ai_char):
        self.hu_char = hu_char
        self.ai_char = ai_char
        self.scores = {
            hu_char: -10,
            ai_char: 10,
            'piece': 0
        }
    def computer_position(self, field):
        move = None
        best_score = -sys.maxsize
        board = [field[i].copy() for i in range(3)]
        for row in range(3):
            for col in range(3):
                if board[row][col] == EMPTY_CHAR:
                    board[row][col] = self.ai_char
                    score = self.minimax(board, 0, HU_TURN)
                    board[row][col] = EMPTY_CHAR
                    if score > best_score:
                        best_score = score
                        move = (row, col)
        return move
    def minimax(self, board, depth, is_ai_turn):
        if check_win(board, self.ai_char):
            return self.scores[self.ai_char]
        if check_win(board, self.hu_char):
            return self.scores[self.hu_char]
        if piece(board, self.hu_char, self.ai_char):
            return self.scores['piece']
        if is_ai_turn:
            # Выбираем ход, который выгоднее нам
```

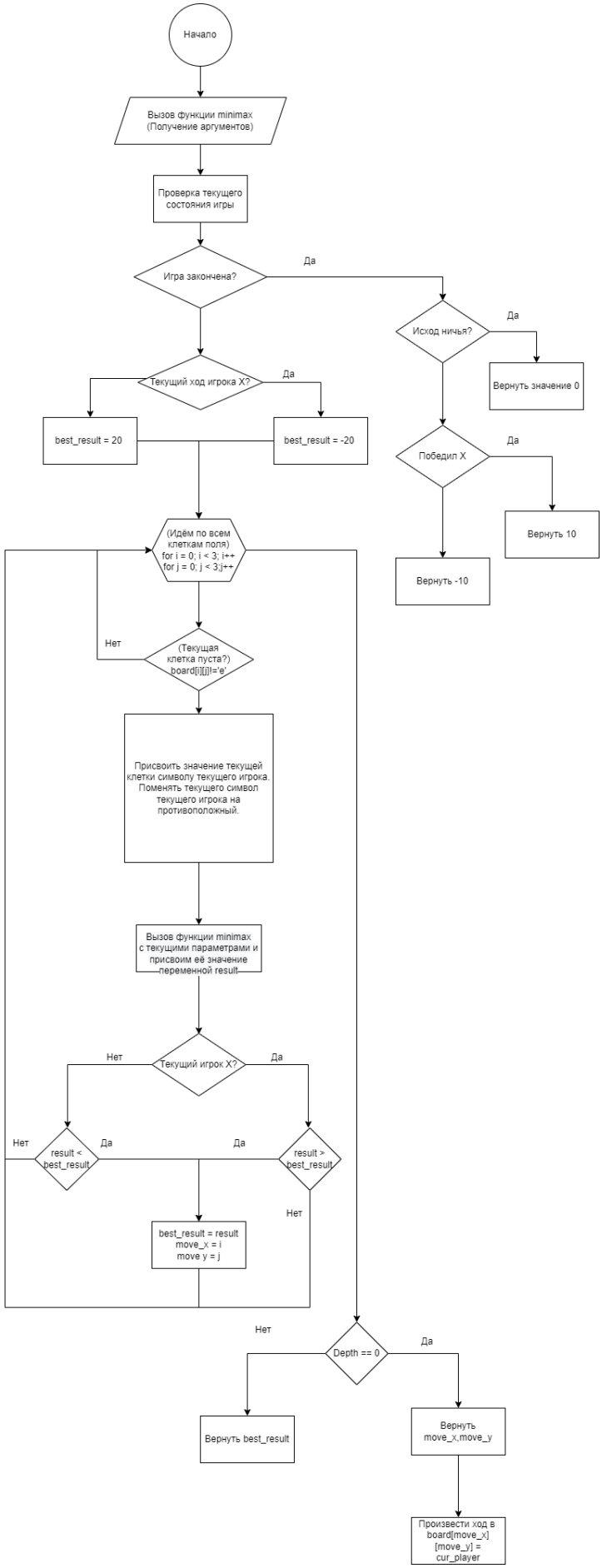
```
            best_score = -sys.maxsize
            for row in range(3):
                for col in range(3):
                    if board[row][col] == EMPTY_CHAR:
                        board[row][col] = self.ai_char
                        score = self.minimax(board, depth + 1,
HU_TURN)
                        board[row][col] = EMPTY_CHAR
                        best_score = max(best_score, score)
        else:
            best_score = sys.maxsize
            for row in range(3):
                for col in range(3):
                    if board[row][col] == EMPTY_CHAR:
                        board[row][col] = self.hu_char
                        score = self.minimax(board, depth + 1,
AI_TURN)
                        board[row][col] = EMPTY_CHAR
                        best_score = min(best_score, score)
        return best_score
def check_win(field, player):
    win = False
    for row in field:
        if row.count(player) == 3:
            win = True
    for col in range(3):
        if field[0][col] == field[1][col] == field[2][col] ==
player:
            win = True
    if (field[0][0] == field[1][1] == field[2][2] == player) or
(field[0][2] == field[1][1] == field[2][0] == player):
        win = True
    return win
def piece(board, hu_char, ai_char):
    if not check_win(board, ai_char):
        if not check_win(board, hu_char):
            if sum([i.count(EMPTY_CHAR) for i in board]) == 0:
                return True
    return False
```

Файл с основными настройками

```
from pygame import mixer, cursors
from pygame.locals import *
mixer.pre_init(44100, -16, 1, 512)
mixer.init()
mixer.music.set_volume(1)
FPS = 30
```

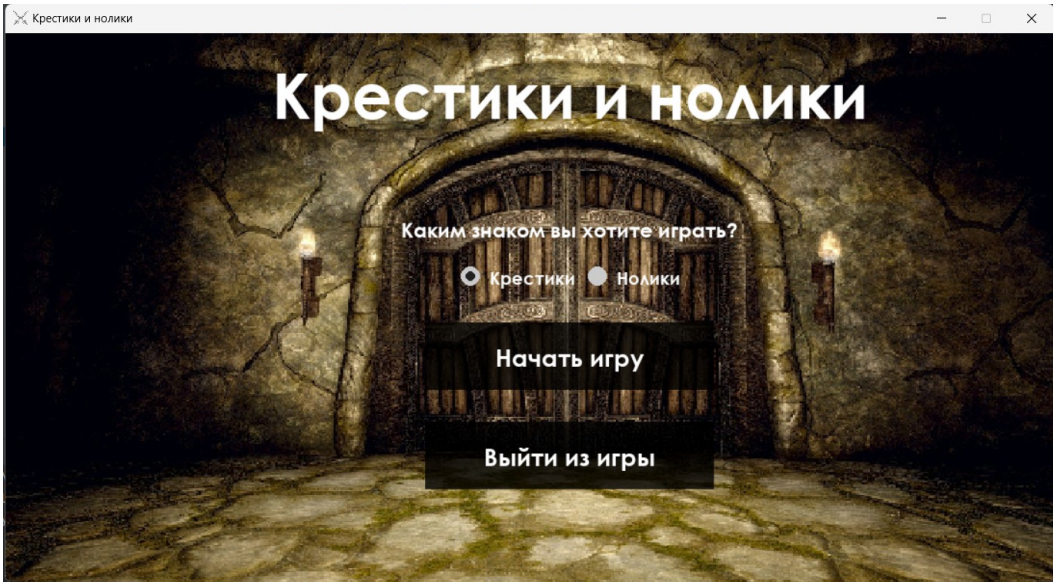
```
CELL_SIZE = 150
MARGIN_X = 160
MARGIN_Y = 60
WIDTH = CELL_SIZE * 3 + MARGIN_X * 4
HEIGHT = CELL_SIZE * 3 + MARGIN_Y * 2
GATE_UNLOCK = mixer.Sound('src/audio/gate_unlock.wav')
SWORD_1 = mixer.Sound('src/audio/sword_1.wav')
SWORD_2 = mixer.Sound('src/audio/sword_2.wav')
SHIELD = mixer.Sound('src/audio/shield.wav')
CLICK = mixer.Sound('src/audio/click.wav')
GATE_UNLOCK.set_volume(0.5)
SWORD_1.set_volume(0.5)
SWORD_2.set_volume(0.5)
SHIELD.set_volume(0.5)
CLICK.set_volume(0.2)
NORM_CURSOR = cursors.Cursor(SYSTEM_CURSOR_ARROW)
FINGER_CURSOR = cursors.Cursor(SYSTEM_CURSOR_HAND)
```

Блок-схема программы

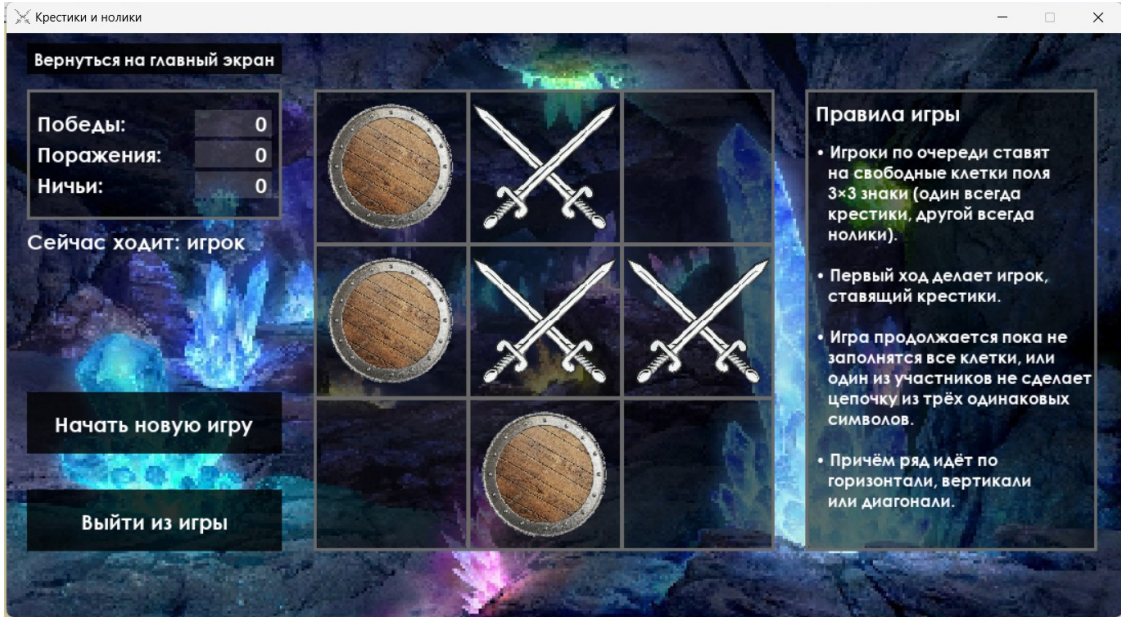


Результат работы программы

Главный экран



Игровой экран



Выводы

В результате работы над проектом было разработано приложение, полностью соответствующее описанным в техническом задании критериям. Во время реализации были выполнены следующие задачи:

- Был написан основной алгоритм для корректной работы искусственного интеллекта
- Был разработан графический интерфейс, состоящий из обычных и радиокнопок, надписей и игровых полей
- Были подобраны соответствующие выбранной тематике изображения фона и

самих крестиков и ноликов

Список используемой литературы

- Методические материалы, прилагаемые к описанию лабораторной работы.
- Официальная документация для работы с модулем pygame.
- Официальная документация для работы с модулем pyinstaller.