Муниципальное автономное общеобразовательное учреждение
«Лицей №14 имени Заслуженного учителя Российской Федерации А.М. Кузьмина»

Приложение-игра «Длинные нарды»

Автор работы:

ученик 10 «К» класса Макаров Артём

Руководитель: Вязовов С.М.,
заведующий кафедрой информатики, учитель информатики

Актуальность проекта

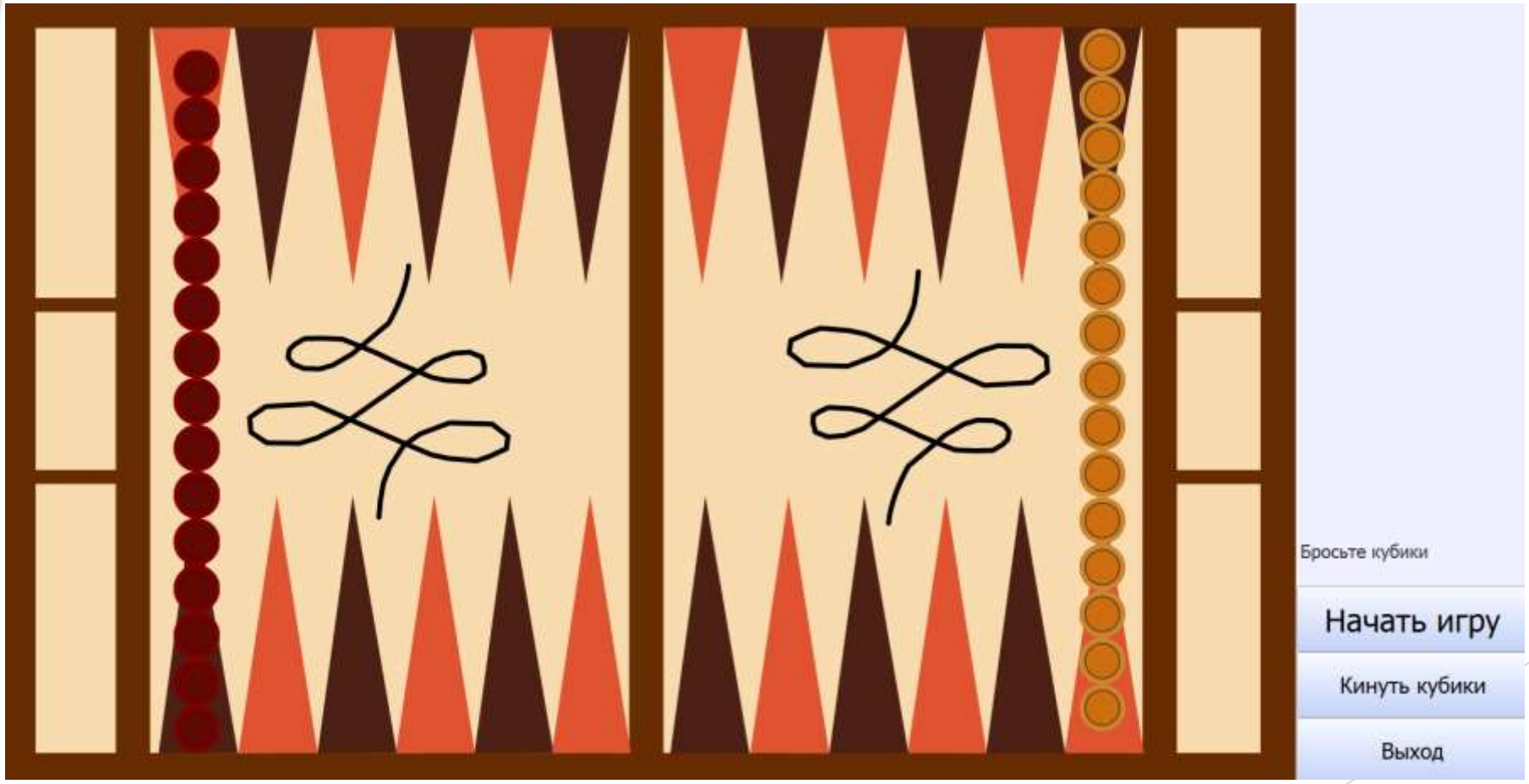
- Нарды – игра с многовековой историей. Она остается популярной благодаря простым правилам. Однако, у людей не всегда есть друг рядом, с которым можно поиграть, или место, где поиграть.
- Моё приложение решает проблему поиска партнера и отсутствие места, где поиграть. Играть можно в любое время и в любом месте.
- На данный момент реализованы два режима игры: «человек против человека» и «человек против бота»

Цель и задачи

- **Цель**: Разработать приложение-игру «Длинные нарды

- **Задачи:**
- Определить набор инструментов для разработки игры и нейросети.
- Изучить необходимые модули выбранного языка программирования.
- Создать графический интерфейс и реализовать игровой процесс.
- Разработать нейронную сеть на основе дерева принятия решений.
- Проверить работоспособность и исправить найденные ошибки.

Ход работы: создание версии, содержащей режим игры «Человек против человека» и простой UI

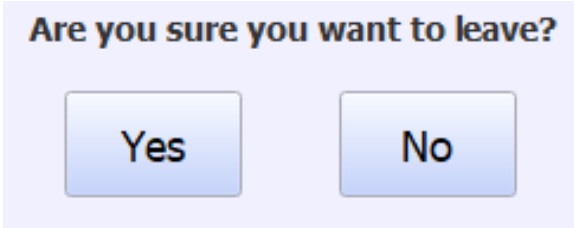


Реализованный функционал в данной версии

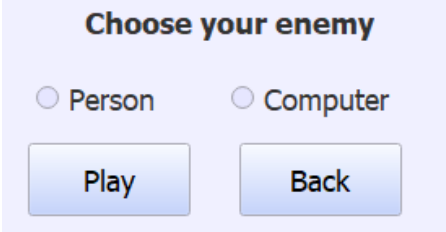
- Разработан UI игры, включая игровое поле, кубики и элементы управления
- Реализованы правила игры, которые определяют ход игры и условия победы
- Пользователь может выбрать язык интерфейса игры (русский или английский)
- Пользователь может выбрать цвет кубиков для игры
- Реализована система вывода сообщений об ошибках на экран
- Приложение поддерживает выбор из двух режимов игры:
  "Человек против человека" и "Человек против компьютера“,
  но реализован только "Человек против человека"
- Функциональный режим игры, где два игрока могут играть друг против друга

UI данной версии

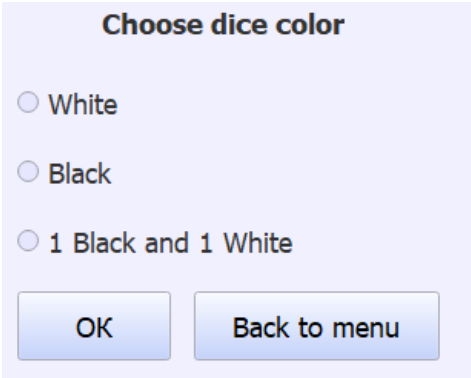
Меню



Выход из игры



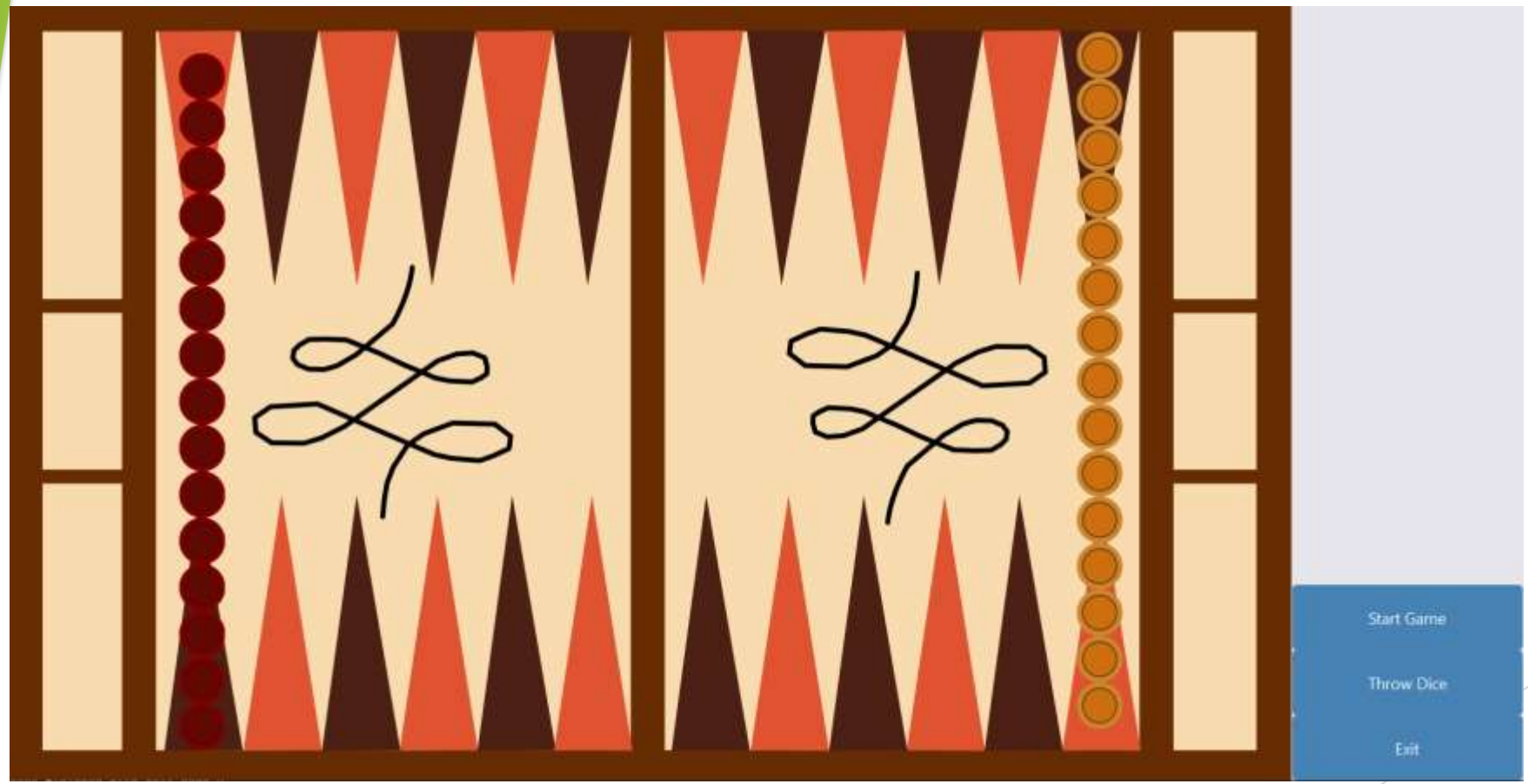
Выбор режима игры



Выбор цвета кубиков

Правила игры

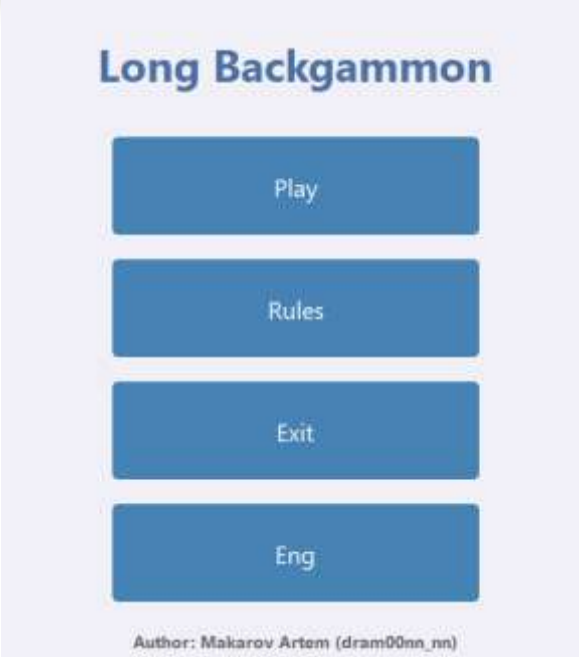
Ход работы: создание текущей версии, содержащей функционал предыдущей версии и режима игры «Человек против бота» и доработанный UI



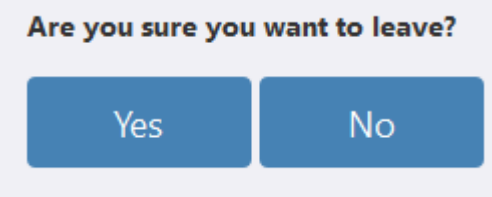
Реализованный и доработанный функционал в данной версии

- Весь функционал версии 1.0 («человек против человека») остался в данной версии, но был немного улучшен
- Функциональный режим игры, где игрок играет против бота
- Доработан UI, добавлена возможность выбора сложности бота, при выборе режима игры против него

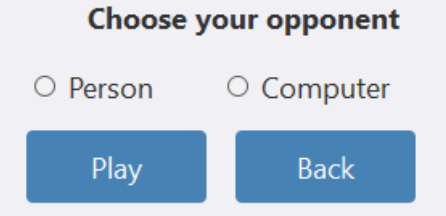
UI данной версии



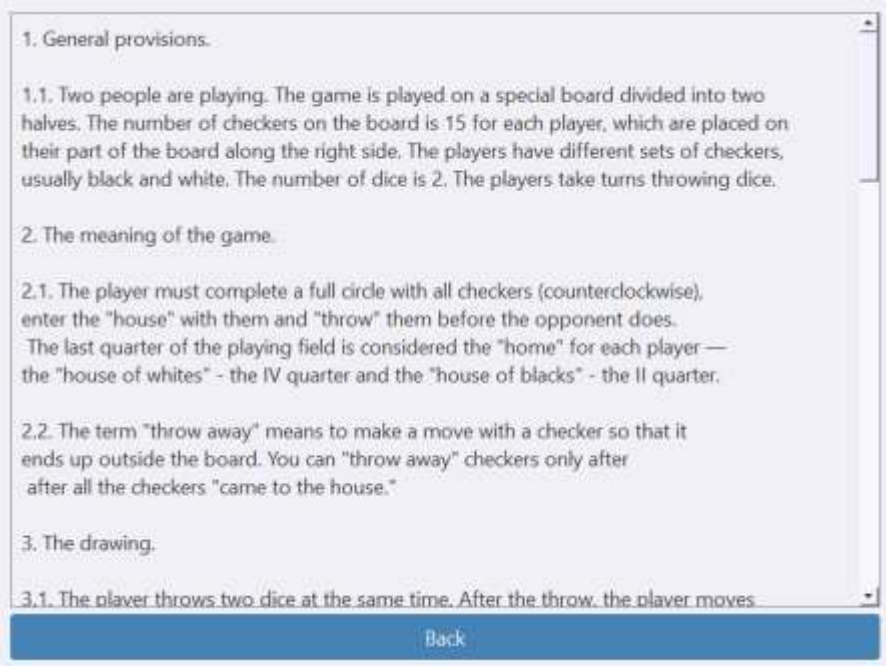
Меню



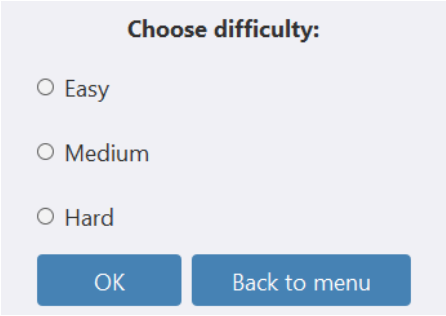
Выход из игры



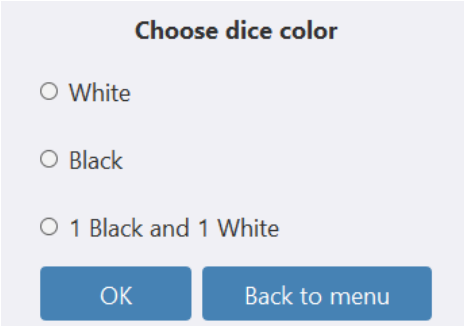
Выбор режима игры



Правила игры



Выбор сложности при игре с ботом



Выбор цвета кубиков

Бот, реализованный через дерево принятия решений

```python
# функции для реализации модели дерево принятия решений
def evaluate_board(self):  1 usage  ▲ Dramon
    """
    Оценка текущего состояния доски.
    Чем выше значение, тем лучше для бота.
    """
    score = 0
    for i, cell in enumerate(self.cells):
        if cell and 'white' in cell[0].objectName():  # Фишки бота (белые)
            score += len(cell) * (24 - i)  # Чем ближе к выходу, тем лучше
    return score

def minimax(self, depth, alpha, beta, maximizing_player):  3 usages  ▲ Dramon
    """
    Алгоритм Minimax с альфа-бета отсечением.
    """

    if depth == 0 or self.is_game_end('white') or self.is_game_end('reddd'):
        return self.evaluate_board()

    if maximizing_player:  # Ход бота (максимизация)
        max_eval = -float('inf')
        for move in self.get_all_possible_moves('white'):
            # Симулируем ход
            self.make_move(move[0], move[1])
            evall = self.minimax(depth - 1, alpha, beta, maximizing_player: False)
            self.undo_move(move[0], move[1])  # Отменяем ход
            max_eval = max(max_eval, evall)
            alpha = max(alpha, evall)
            if beta <= alpha:
                break  # Альфа-бета отсечение
        return max_eval
```

```python
    else:  # Ход игрока (минимизация)
        min_eval = float('inf')
        for move in self.get_all_possible_moves('reddd'):
            # Симулируем ход
            self.make_move(move[0], move[1])
            evall = self.minimax(depth - 1, alpha, beta, maximizing_player: True)
            self.undo_move(move[0], move[1])  # Отменяем ход
            min_eval = min(min_eval, evall)
            beta = min(beta, evall)
            if beta <= alpha:
                break  # Альфа-бета отсечение
        return min_eval

def get_best_move(self):  2 usages  ▲ Dramon
    """
    Возвращает лучший ход для бота на основе Minimax.
    """
    best_move = None
    max_eval = -float('inf')
    for move in self.get_all_possible_moves('white'):
        # Симулируем ход
        self.make_move(move[0], move[1])
        eval = self.minimax(self.depth - 1, -float('inf'), float('inf'), maximizing_player: False)
        self.undo_move(move[0], move[1])  # Отменяем ход
        if eval > max_eval:
            max_eval = eval
            best_move = move
    return best_move
```

```python
def get_all_possible_moves(self, color):  3 usages  ▲ Dramon
    """
    Возвращает все возможные ходы для текущего игрока.
    """

    print(23, [(x[0].objectName() if x else '----') for x in self.cells])
    moves = []
    for src in range(23):
        if not self.cells[src]: continue
        if color not in self.cells[src][0].objectName(): continue
        for dice in [self.first_dice, self.second_dice]:
            if dice == -1000: continue
            dst = (src + dice) % 24
            if self.is_move_valid(src, dice):
                moves.append((src, dst))
    print(31, moves, self.first_dice, self.second_dice)
    return moves
```

Про алгоритм minmax с альфа-бета отсечением

Алгоритм **Minimax с альфа-бета отсечением** — это способ принимать оптимальные решения в играх с двумя противниками (например, нарды, шахматы, крестики-нолики). Он помогает выбрать ход, который максимизирует ваш выигрыш, учитывая, что противник будет минимизировать его.

1) Базовый Minimax
Цель: Найти путь к листу дерева (конечной позиции) с максимальной оценкой, предполагая, что противник играет оптимально.
Проблема: В сложных играх дерево огромное, и перебирать все варианты слишком долго.
2) Альфа-бета отсечение
Это оптимизация Minimax, которая «отрезает» ветви дерева, которые точно не повлияют на итоговое решение.
•**Альфа (α)** — лучшая оценка для максимизирующего игрока на текущем уровне(изначально -∞).
•**Бета (β)** — лучшая оценка для минимизирующего игрока (изначально +∞).
Как это работает:
•Вы идете вглубь дерева и считаете оценку для текущего узла.
•Если натыкаетесь на ситуацию, где:

- **Для вашего хода (максимизация):** текущая оценка ≥ β → дальше можно не смотреть (ветка отсекается).
- **Для хода противника (минимизация):** текущая оценка ≤ α → ветка тоже отсекается.

3) Плюсы альфа-бета
•Не меняет результат — решение такое же, как у обычного Minimax.
•Ускоряет поиск — иногда в разы (особенно если проверять лучшие ходы первыми).

4) Аналогия из жизни:
• α — ваша минимальная цена («не меньше 100 рублей»).
• β — максимальная цена продавца («не больше 200 рублей»).
Если продавец предлагает товар за 150 рублей, а вы уже нашли вариант за 120 рублей, дальнейший торг бессмыслен — вы отсекаете переговоры.

Заключение

- Подробно изучен модуль PyQt5
- Изучено и реализовано устройство дерева принятия решений на основе алгоритма minmax(минимизация ходов соперника и максимизация своих ходов)
- Разработан UI приложения
- Разработаны два режима игры: «человек против человека» и «человек против бота»

В итоге была создана: приложение игра с режимами игры «человек против человека» и «человек против бота» с понятным аккуратным графическим интерфейсом

Список использованных источников

- https://habr.com/ru/companies/productstar/articles/523044/
- https://ru.wikipedia.org/wiki/%D0%9C%D0%B8%D0%BD%D0%B8%D0%BC%D0%B0%D0%BA%D1%81
- https://habr.com/ru/companies/otus/articles/785512/
- https://doc.qt.io/qtforpython-5/
- https://ru.stackoverflow.com/
- http://python-3.ru/category/pyqt
- https://python-scripts.com/pyqt5
- https://s.econf.rae.ru/pdf/2014/03/3245.pdf