Xây dựng chương trình chơi Cờ toán Việt Nam

Bài tập lớn môn học
Nhập môn trí tuệ nhân tạo

Trường Công nghệ Thông tin và Truyền thông
Đại học Bách Khoa Hà Nội

Ngày 2 tháng 2 năm 2023

Nhóm 5

Các thành viên:

- Phan Minh Anh Tuấn – 20205227
- Nguyễn Thị Hoài Linh – 20205231
- Vũ Tuấn Kiệt - 20200308
- Nguyễn Hoàng Long - 20200364
- Đồng Gang Thép - 20205130

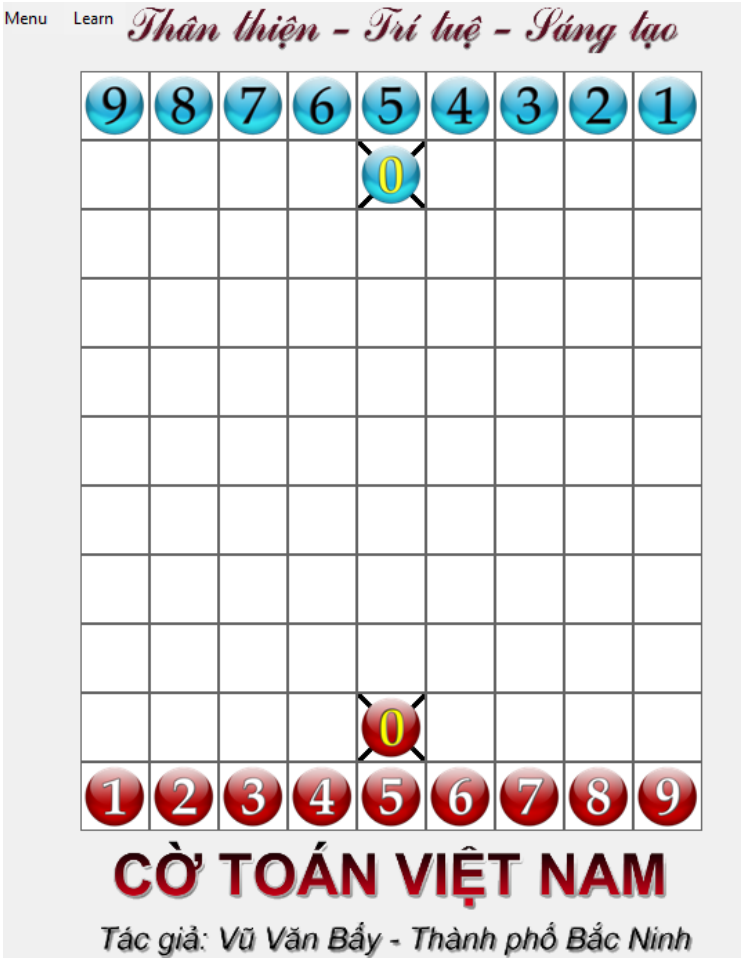
Mục lục

Mục lục

Giới thiệu cờ toán Việt Nam

Cờ toán Việt Nam là sản phẩm của ông Vũ Văn Bảy. Trò chơi đã được Cục Bản quyền tác giả văn học nghệ thuật, Bộ Văn hóa, Thể thao và Du lịch chính thức công nhận sản phẩm trí tuệ vào tháng 5-2005.

Đây là trò chơi đối kháng trên bàn cờ có kích thước 9x11 giữa 2 bên: quân xanh và quân đỏ. Mỗi bên có các quân từ 0 đến 9 được bố trí như hình vẽ.



Hình 1: Bàn cờ toán Việt Nam

Luật chơi cờ toán Việt Nam

Luật đi quân

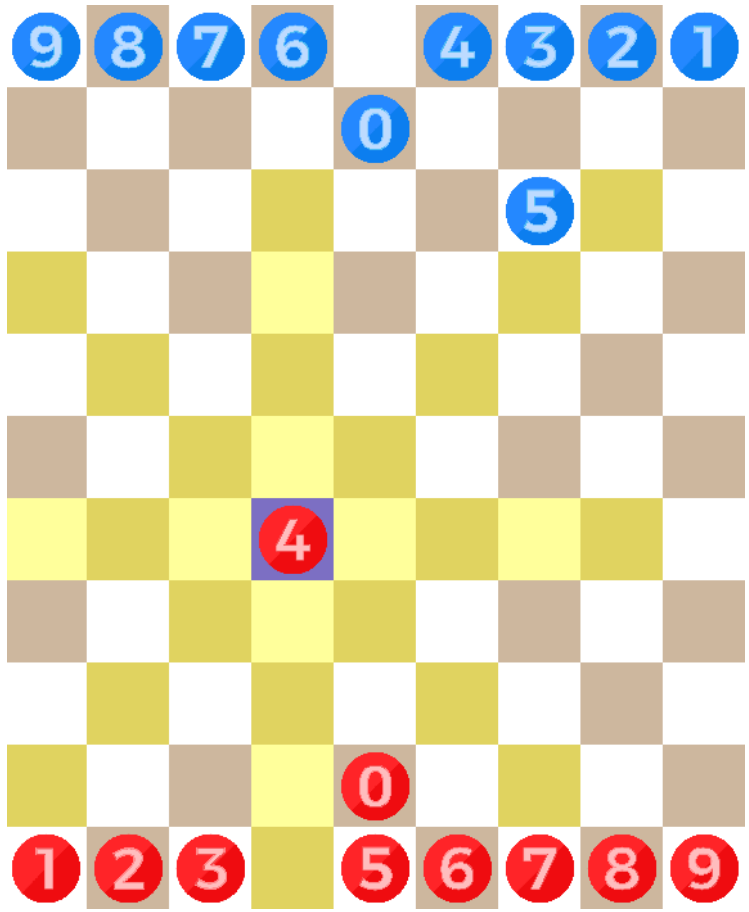
- Quân 0 cố định
- Quân 1-9 đi được theo tám hướng và mỗi hướng đi được tất cả các ô theo giá trị của quân cờ

Luật ăn quân

- Hai quân phải kề nhau
- Thực hiện các phép '+' '-' '*' '/' để tìm ra ô có thể ăn của đối thủ

Luật thắng thua

- Mất quân 0
- Tính theo tổng điểm



Hình 2: Ví dụ về di chuyển

Luật chơi cờ toán Việt Nam

Luật đi quân

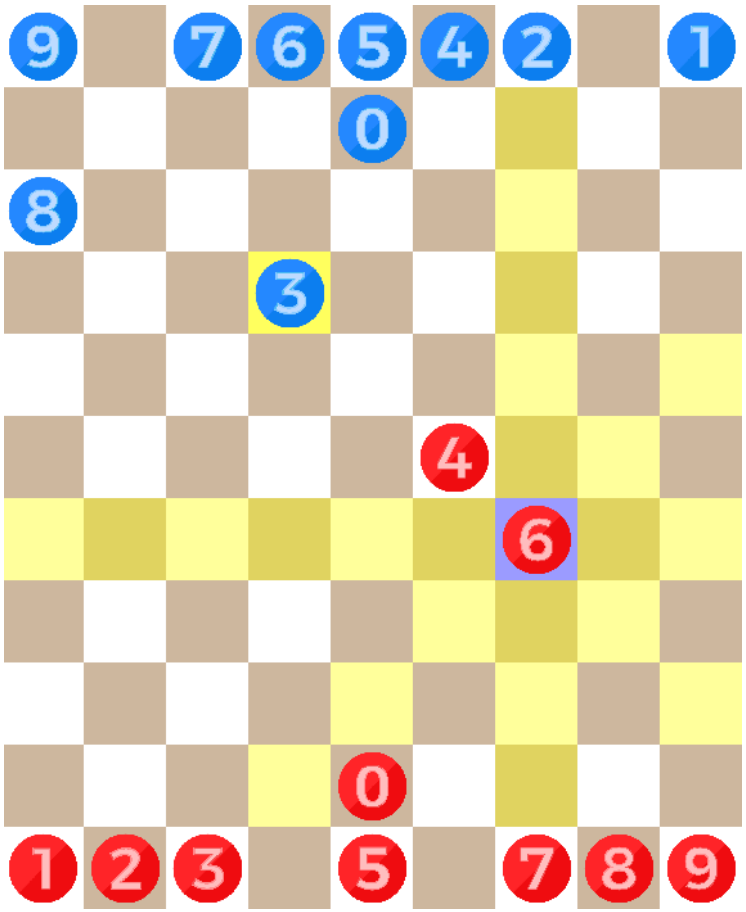
- Quân 0 cố định
- Quân 1-9 đi được theo tám hướng và mỗi hướng đi được tất cả các ô theo giá trị của quân cờ

Luật ăn quân

- Hai quân phải kề nhau
- Thực hiện các phép '+' '-' '*' '/' để tìm ra ô có thể ăn của đối thủ

Luật thắng thua

- Mất quân 0
- Tính theo tổng điểm



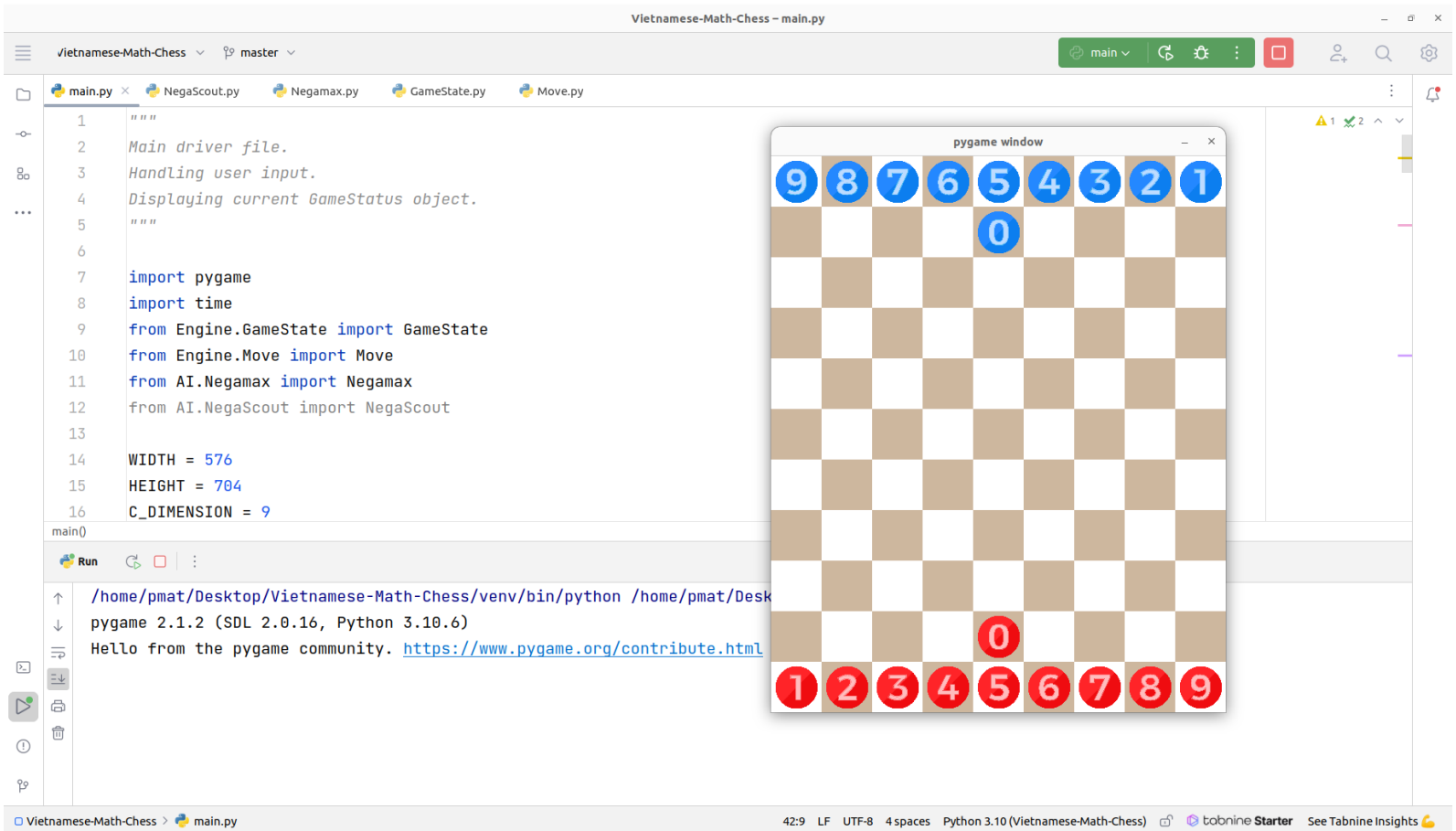
Hình 3: Ví dụ về ăn quân

Mục lục

Thiết kế trò chơi

Chia trò chơi làm 3 module chính:

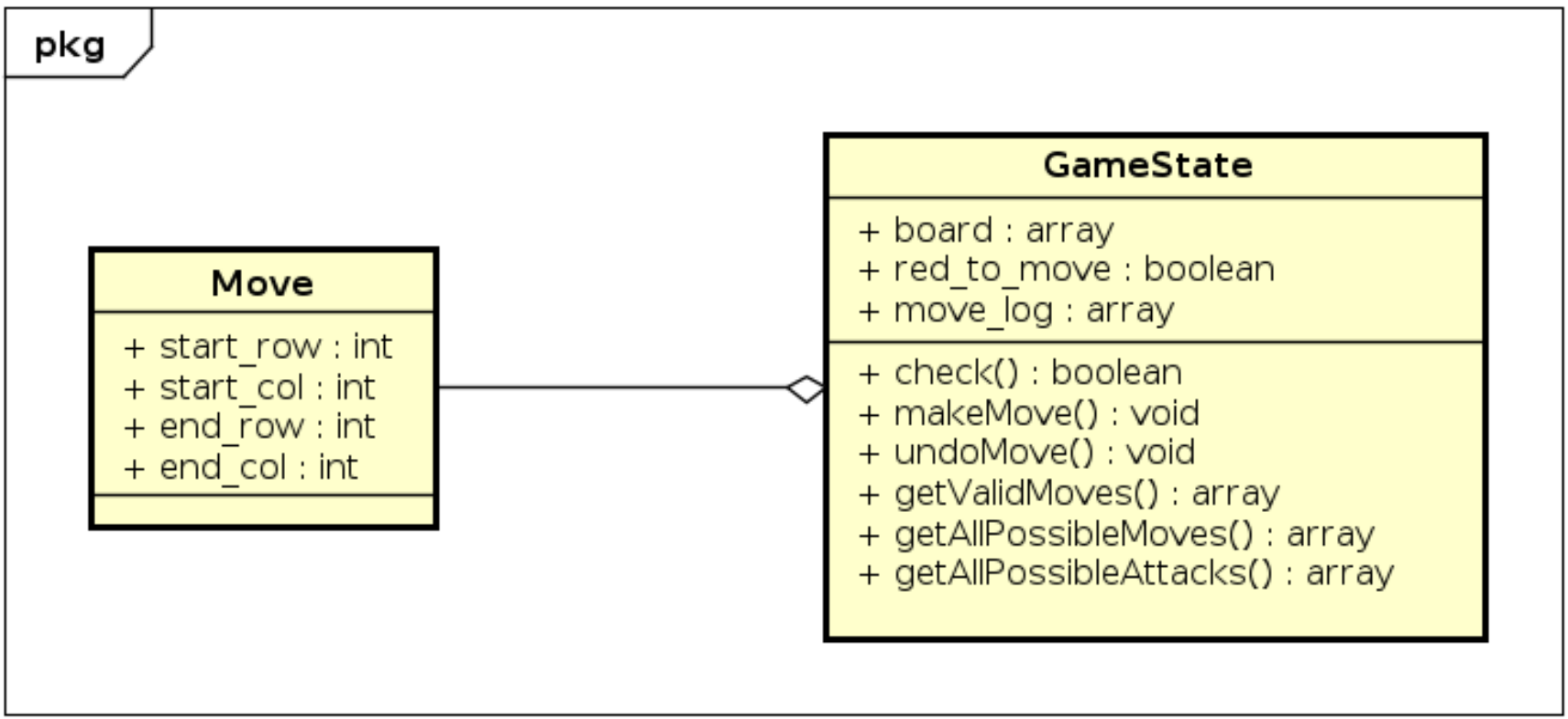
1. Engine: mã hóa trạng thái bàn cờ và các nước đi
2. UI: tạo giao diện cho trò chơi
3. AI: triển khai các thuật toán tìm kiếm có đối thủ vào trò chơi



Engine

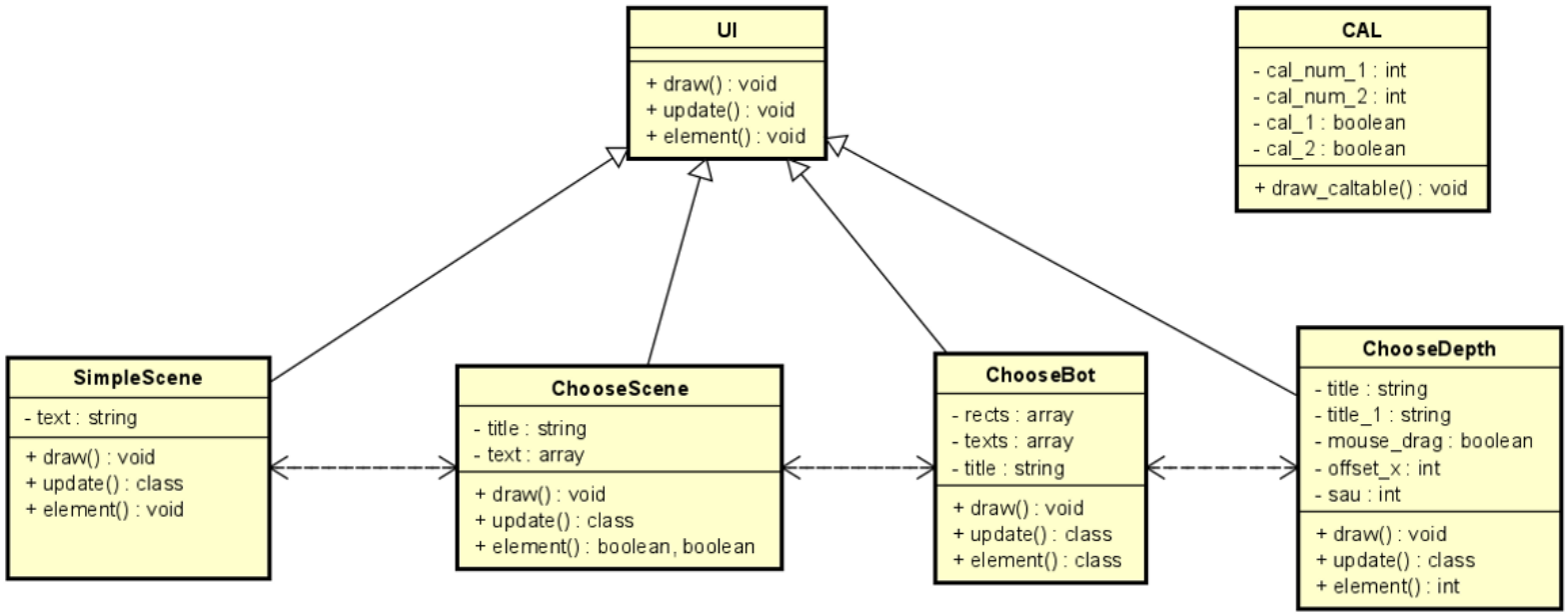
Bao gồm 2 class:

1. GameState: mã hóa trạng thái bàn cờ (lượt đi, các nước đã đi, các nước đi hợp lệ tiếp theo)
2. Move: mã hóa nước đi



Hình 4: Các class trong module engine

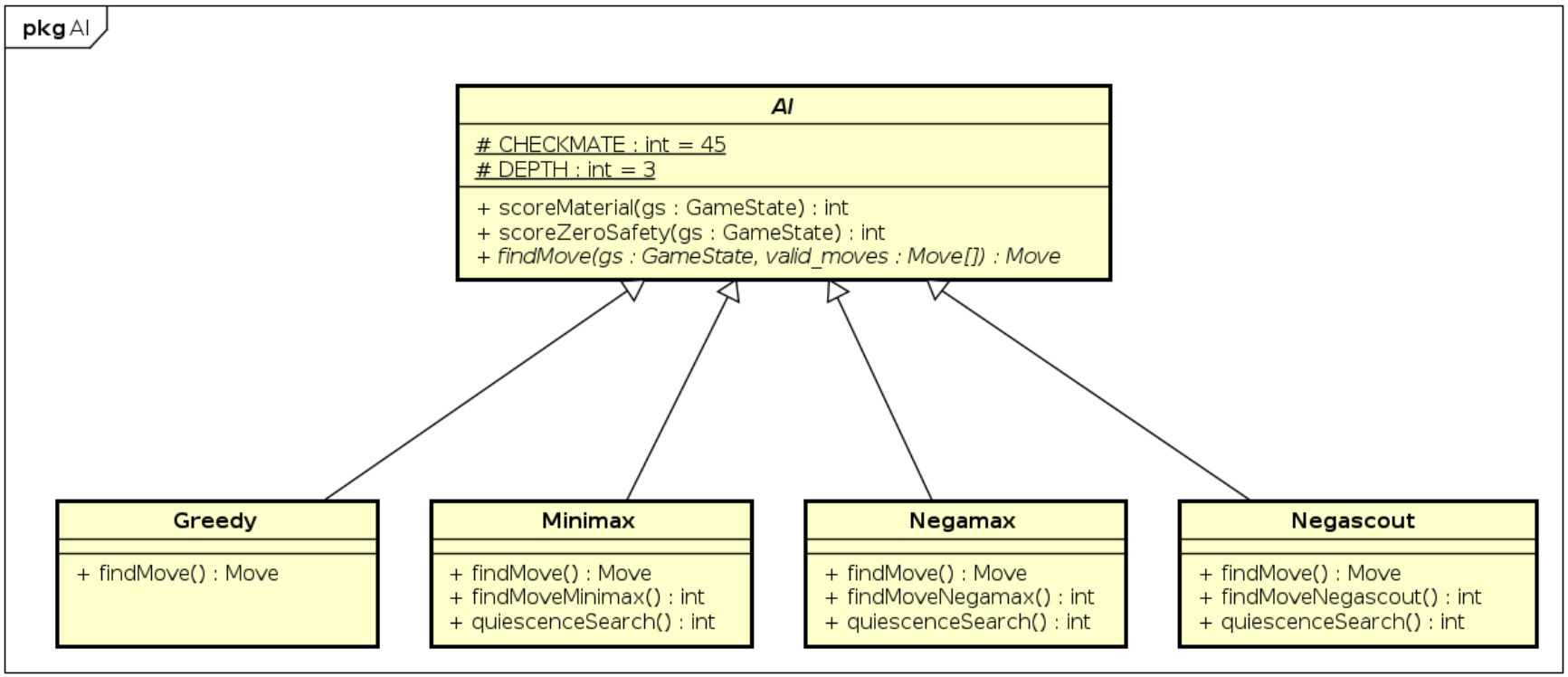
UI



Hình 5: Các class trong module UI

AI

Bao gồm 5 class:



Hình 6: Các class trong module AI

Mục lục

Hàm ước lượng

Hàm ước lượng được đánh giá bằng tổng số điểm của các quân còn trên bàn cờ, riêng quân 0 đóng vai trò quan trọng nên điểm của quân 0 được tính là 45.

```
def scoreMaterial(board):
  score = 0
  for row in board:
      for square in row:
          if square is red:
              if value(square) == 0:
                  score += 45
              else:
                  score += value(square)
          elif square is blue:
              if value(square) == 0:
                  score -= 45
              else:
                  score -= value(square)
  return score
```

Hình 7: Mã giả hàm ước lượng

Thuật toán tham lam

```
def greedy(state, valid_moves):
    max_score = -inf
    for move in valid_moves:
        make_move(move)
        score = evaluate(state)
        undo_move(move)
        if score > max_score:
            max_score = score
            next_move = move
    return next_move
```

Hình 8: Mã giả thuật toán tham lam

Thuật toán minimax

```
def minimax(node, depth, maximizingPlayer):
    if depth == 0 or node is leaf:
        return evaluate(node)

    if maximizingPlayer:
        bestValue = -inf
        for child in children(node):
            value = minimax(child, depth - 1, False)
            bestValue = max(bestValue, value)
        return bestValue
    else:
        bestValue = inf
        for child in children(node):
            value = minimax(child, depth - 1, True)
            bestValue = min(bestValue, value)
        return bestValue
```

Hình 9: Mã giả thuật toán minimax

Thuật toán minimax cắt tỉa alpha-beta

```
def minimax(node, depth, alpha, beta, maximizingPlayer):
    if depth == 0 or node is a leaf:
        return evaluate(node)

    if maximizingPlayer:
        bestValue = -inf
        for child in children(node):
            value = minimax(child, depth - 1, alpha, beta, False)
            bestValue = max(bestValue, value)
            alpha = max(alpha, bestValue)
            if alpha ⩾ beta:
                break
        return bestValue
    else:
        bestValue = -inf
        for child in children(node):
            value = minimax(child, depth - 1, alpha, beta, True)
            bestValue = min(bestValue, value)
            beta = min(beta, bestValue)
            if alpha ⩾ beta:
                break
        return bestValue
```

Hình 10: Mã giả thuật toán minimax kết hợp cắt tỉa alpha-beta

Thuật toán negamax

```
def negamax(node, depth, player):
  if depth == 0 or node is a leaf:
    return player * evaluate(node)

  bestValue = -inf
  for child in children(node):
    value = -negamax(child, depth-1, -player)
    bestValue = max(bestValue, value)
  return bestValue
```

Hình 11: Mã giả thuật toán negamax

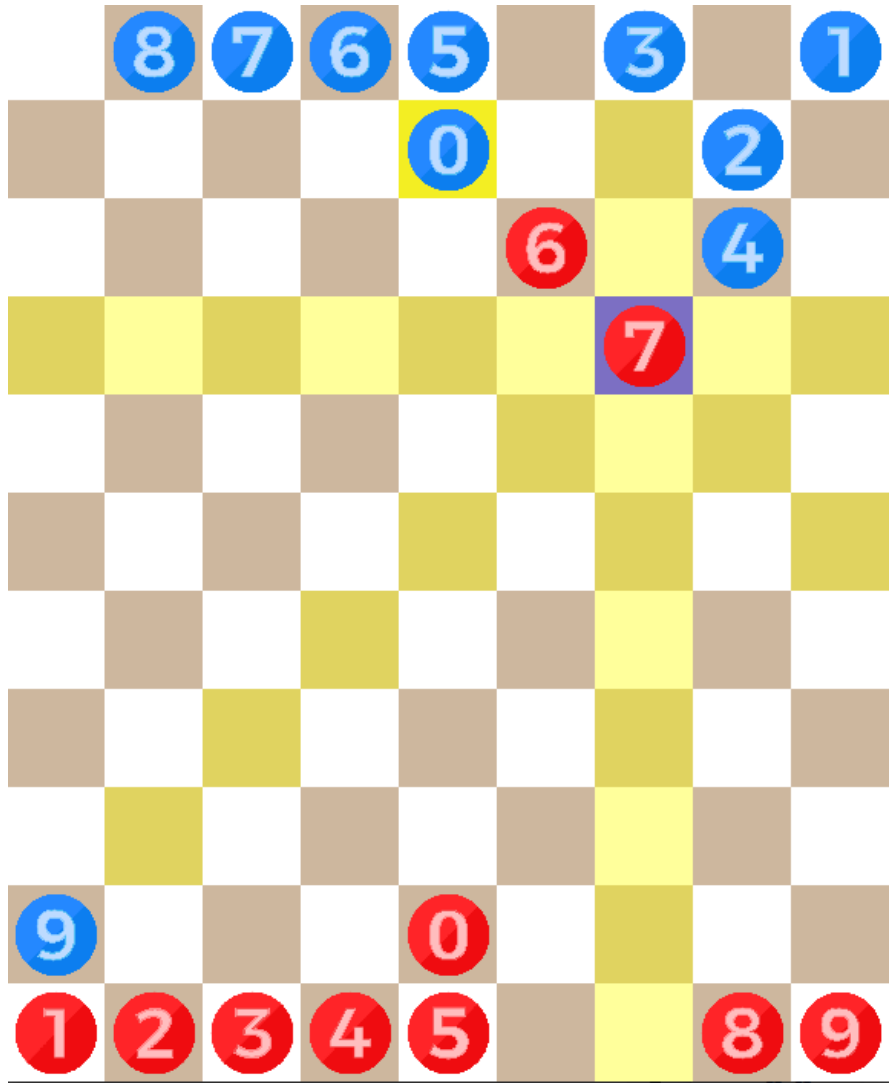
Thuật toán negamax cắt tỉa alpha-beta

```
def negamax(node, depth, player, alpha, beta):
  if depth == 0 or node is a leaf:
    return player * evaluate(node)

  bestValue = -inf
  for child in children(node):
    bestValue = max(bestValue, -negamax(child, depth-1,
                -player, -beta, -alpha))
    alpha = max(alpha, bestValue)
    if alpha ⩾ beta:
      break
  return bestValue
```

Hình 12: Mã giả thuật toán negamax kết hợp cắt tỉa alpha-beta

Vấn đề



Hình 13: Không dự đoán để tránh được chiếu hết

Thay đổi hàm ước lượng

```
def scoreZeroSafety(board):
  score = 0
  red_direction = ((-1, -1), (-1, 1), (-1, 0), (0, -1), (0, 1))
  blue_direction = ((0, -1), (0, 1), (1, -1), (1, 0), (1, 1))
  for d in red_direction:
      for i in range(1, 4, 1):
          if board[d[0] * i][d[1] * i][0] == 'b':
              score -= 4 - i
      for d in blue_direction:
          for i in range(1, 4, 1):
              if board[d[0] * i][d[1] * i][0] == 'r':
                  score += 4 - i
  return score
```

Hình 14: Thay đổi hàm ước lượng

Thay đổi hàm ước lượng

```
def scoreMaterial(board):
    score = 0
    for row in board:
        for square in row:
            if square is red:
                if value(square) == 0:
                    score += 45
                else:
                    score += value(square)
            elif square is blue:
                if value(square) == 0:
                    score -= 45
                else:
                    score -= value(square)
    score += scoreZeroSafety(board)
    return score
```

Hình 15: Thay đổi hàm ước lượng

Quiescence search

```
def quiescence(alpha, beta):
    best_score = evaluate()
    if best_score ⩾ beta:
        return beta
    if alpha < best_score:
        alpha = best_score

    for move in all_captures():
        make_capture(move)
        score = -quiescence(-beta, -alpha)
        undo_capture(move)

        if score ⩾ beta:
            return beta
        if score > alpha:
            alpha = score
    return alpha
```

Hình 16: Mã giả quiescene search

Thuật toán negascout

```
def negascout(node, depth, alpha, beta):
    if depth == 0 or node is a terminal node:
        return evaluate(node)
    for child in node:
        if child == node[0]:
            score = -negascout(child, depth-1, -beta, -alpha)
        else:
            score = -negascout(child, depth-1, -alpha-1, -alpha)
            if alpha < score < beta:
                score = -negascout(child, depth-1, -beta, -score)
        alpha = max(alpha, score)
        if alpha ⩾ beta:
            break
    return alpha
```

Hình 17: Mã giả thuật toán negascout

Phân chia công việc

Họ tên	Công việc
Phan Minh Anh Tuấn	Mã hóa trò chơi. Triển khai Negascout, Quiescence search
Nguyễn Thị Hoài Linh	Triển khai thuật toán Greedy, Minimax, Negamax. Cải thiện hàm ước lượng
Đồng Gang Thép	Giao diện đồ họa
Vũ Tuấn Kiệt	Tìm hiểu Negamax, Negascout. Triển khai Negascout
Nguyễn Hoàng Long	Bổ sung vào giao diện: Tính điểm người chơi, hiển thị các quân có thể tấn công được

The End