VIETNAM NATIONAL UNIVERSITY
HO CHI MINH CITY UNIVERSITY OF TECHNOLOGY
FACULTY OF COMPUTER SCIENCE AND ENGINEERING

NHẬP MÔN TRÍ TUỆ NHÂN TẠO (CO3061)

Project report

Bloxorz Solver With BFS and Genetic Algorithm

Instructor: Vương Bá Thịnh
Performers: Dương Đức Nghĩa 2011671

Ho Chi Minh City, November 2022

Mục lục

1 Dẫn nhập

1.1 Phạm vi của báo cáo

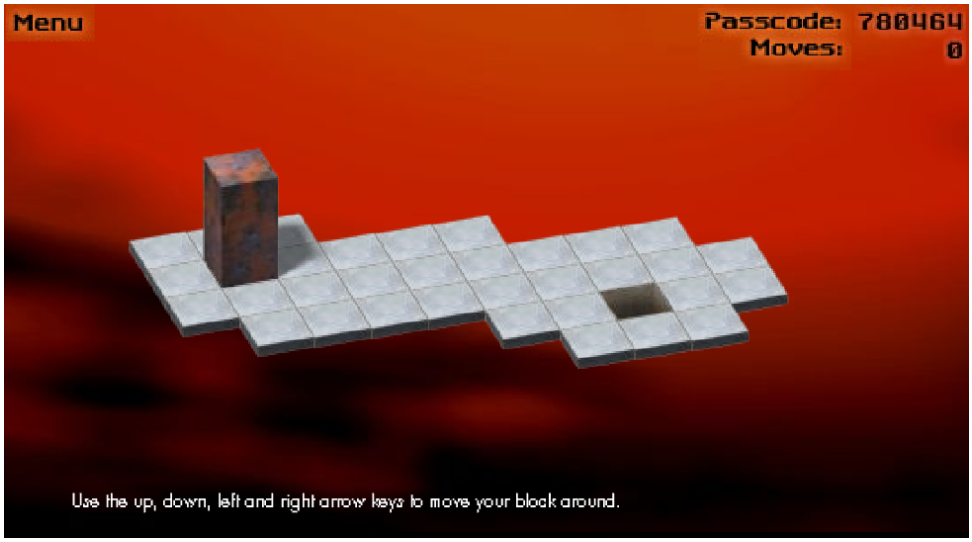
Trong bài báo cáo lần này nhóm tác giả sẽ hiện thực tìm đường đi cho trò chơi Bloxorz bằng 2 giải thuật (1) BFS và (2) Genetic Algorithm

1.2 Những nét chính

Đầu tiên nhóm tác giả sẽ giới thiệu về game Bloxorz, luật chơi, cũng như là nói sơ lược về 2 giải thuật BFS và Genetic. Tiếp theo là phân tích cách để đưa 2 giải thuật này áp dụng để có thể tối ưu hóa thời gian tìm ra đáp án, sau đó nhóm sẽ trình bày cách thực hiện bằng code python, cuối cùng là trình bày kết quả và đánh giá chi tiết.

2 Giới thiệu

2.1 Game Bloxorz



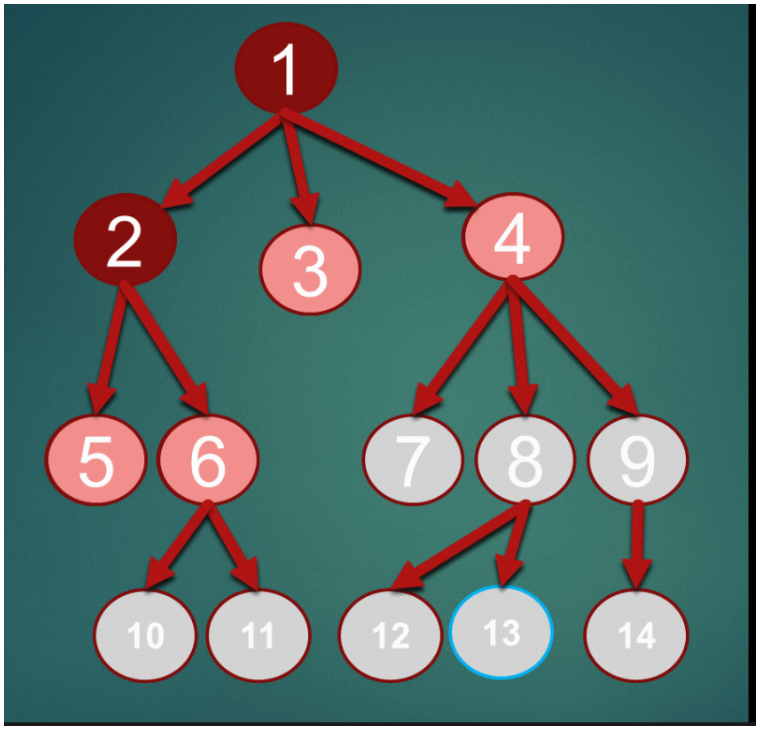
Hình 1: Game Bloxorz

Bàn chơi Bloxorz bao gồm 1 block đứng trên 1 bàn chơi được ghép bằng nhiều ô vuông. Lỗ đen nằm trên bàn chơi này chính là đích đến cuối cùng mà người chơi cần đạt tới. Bằng cách sử dụng 4 phím mũi tên trên bàn phím, người chơi sẽ điều khiển cho khối hình chữ nhật này di chuyển trên bàn chơi sao cho lọt vào lỗ mà không bị rơi ra khỏi bàn. Có tất cả 33 màn chơi, mỗi màn có một địa hình khác nhau. Có tất cả 3 công tắc: công tắc mềm(sort switch), công tắc cứng(hard switch) và công tắc dịch chuyển(teleport switch). Công tắc mềm sẽ được kích hoạt khi bất cứ bộ phận nào của block chạm vào, công tắc cứng và công tắc dịch chuyển yêu cầu khối phải đứng để có thể kích hoạt, nếu kích hoạt công tắc dịch chuyển, block sẽ bị "phân thân" và đi đến các vị trí khác nhau, trường hợp 2 block sau khi bị phân thân mà ở vị trí gần nhau sẽ được dính lại, các công tắc mềm và cứng sẽ kích hoạt các ô khác nhau đóng và mở. Điều lưu ý cuối cùng là không được đứng trên khối màu cam.

2.2 Giải thuật BFS

Trong lý thuyết đồ thị, tìm kiếm theo chiều rộng (BFS) là một thuật toán tìm kiếm trong đồ thị trong đó việc tìm kiếm chỉ bao gồm 2 thao tác: (a) cho trước một đỉnh của đồ thị; (b) thêm các đỉnh kề với đỉnh vừa cho vào danh sách có thể hướng tới tiếp theo. Có thể sử dụng thuật toán tìm kiếm theo chiều rộng cho hai mục đích: tìm kiếm đường đi từ một đỉnh gốc cho trước tới một đỉnh đích, và tìm kiếm đường đi từ đỉnh gốc tới tất cả các đỉnh khác. Trong đồ thị không có trọng số, thuật toán tìm kiếm theo chiều rộng luôn tìm ra đường đi ngắn nhất có thể. Thuật toán BFS bắt đầu từ đỉnh gốc và lần lượt nhìn các đỉnh kề với đỉnh gốc. Sau đó, với mỗi đỉnh trong số đó, thuật toán lại lần lượt nhìn trước các đỉnh kề với nó mà chưa được quan sát trước đó và lặp lại. Xem thêm thuật toán tìm kiếm theo chiều sâu, trong đó cũng sử dụng 2 thao tác trên nhưng có trình tự quan sát các đỉnh khác với thuật toán tìm kiếm theo chiều rộng.

Đây là một thuật toán trong trí tuệ nhân tạo. Cấu trúc dữ liệu được sử dụng là hàng đợi (queue).

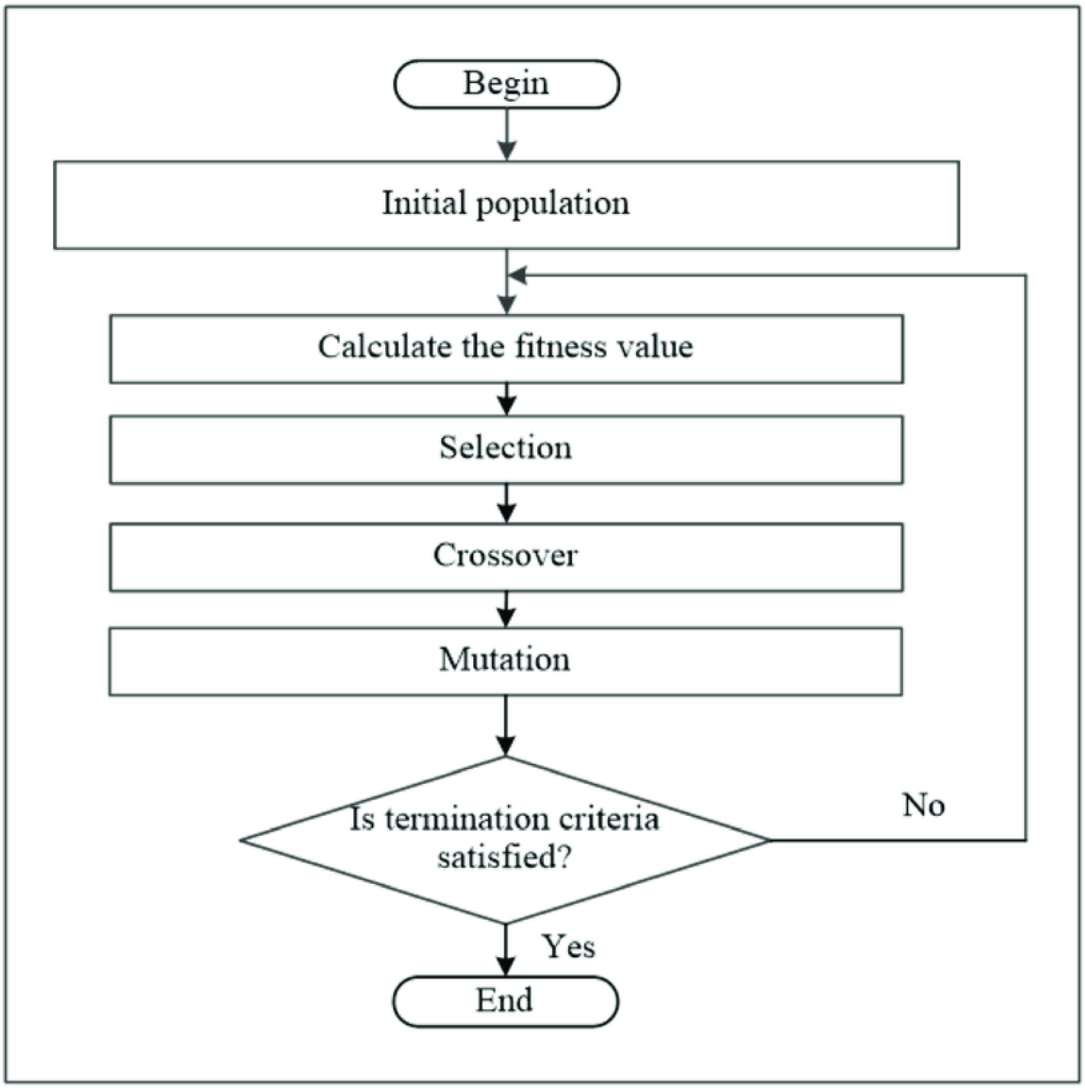


2.3 Genetic Algorithm

Giải thuật di truyền (GA-Genetic Algorithm) là kỹ thuật phỏng theo quá trình thích nghi tiến hóa của các quần thể sinh học dựa trên học thuyết Darwin. GA là phương pháp tìm kiếm tối ưu ngẫu nhiên bằng cách mô phỏng theo sự tiến hóa của con người hay của sinh vật. Tư tưởng của thuật toán di truyền là mô phỏng các hiện tượng tự nhiên, là kế thừa và đấu tranh sinh tồn.

GA thuộc lớp các giải thuật xuất sắc nhưng lại rất khác các giải thuật ngẫu nhiên vì chúng kết hợp các phần tử tìm kiếm trực tiếp và ngẫu nhiên. Khác biệt quan trọng giữa tìm kiếm của GA và các phương pháp tìm kiếm khác là GA duy trì và xử lý một tập các lời giải, gọi là một quần thể (population). Trong GA, việc tìm kiếm giả thuyết thích hợp được bắt đầu với một quần thể, hay một tập hợp có chọn lọc ban đầu của các giả thuyết. Các cá thể của quần thể hiện tại khởi nguồn cho quần thể thế hệ kế tiếp bằng các hoạt động lai ghép và đột biến ngẫu nhiên – được lấy mẫu sau các quá trình tiến hóa sinh học. Ở mỗi bước, các giả thuyết trong quần thể hiện tại được ước lượng liên hệ với đại lượng thích nghi, với các giả thuyết phù hợp nhất được chọn theo xác suất là các hạt giống cho việc sản sinh thế hệ kế tiếp, gọi là cá thể (individual). Cá thể nào phát triển hơn, thích ứng hơn với môi trường sẽ tồn tại và ngược lại sẽ bị đào thải. GA có thể dò tìm thế hệ mới có độ

thích nghi tốt hơn. GA giải quyết các bài toán quy hoạch toán học thông qua các quá trình cơ bản: lai tạo (crossover), đột biến (mutation) và chọn lọc (selection) cho các cá thể trong quần thể. Dùng GA đòi hỏi phải xác định được: khởi tạo quần thể ban đầu, hàm đánh giá các lời giải theo mức độ thích nghi – hàm mục tiêu, các toán tử di truyền tạo hàm sinh sản.



3 Hiện thực

3.1 Hiện thực game

3.1.1 Các module để hiện thực

`Main.py`
Đây là module chính của chương trình
`Read_level_input.py`
Module giúp đọc file, chuyển đổi từ file txt sang dữ liệu đầu vào trong game
`test.py`
Hỗ trợ việc test các level
`draw.py`

Hỗ trợ hiển thị game bằng pygame
`algorithm.py`
Chứa các giải thuật gồm bfs và genetic
`functions.py`
Đây là file dài nhất vì nó chứa tất cả các hàm hỗ trợ
`block.py`
File chứa thông tin về class block
`global_variables.py`
Chứa các biến toàn cục dùng trong các module khác nhau
`genetic_algorithm.py`
Chứa cách hiện thực giải thuật genetic

3.1.2 Xử lí dữ liệu đầu vào

Mô hình hóa dữ liệu

```
(in lvl 10.txt)
10 14 9 1
# # # . . . . . # # # # # #
# G # . . # . . # # # # @ #
# # # . . . . . # # # # . .
. . . . . . . . . # # # . .
. . . . . . . . . . . # # .
. . . . . . . . . . . . # .
. . . . . . . . . . . . # .
. . . . . . . . . . . # # .
. . . . # # # # # . . # # .
. . . . # o . . # # # x # .
3
{   "switch": "teleport_switch",
    "position": [12,1],
    "blocks_process": 2,
    "point": [[9,1],[12,1]],
    "type": "toggle" }
{   "switch": "hard_switch",
    "position": [11,9],
    "blocks_process": 4,
    "point": [[6,1],[7,1],[12,2],[12,3]],
    "type": "toggle" }
{   "switch": "sort_switch",
    "position": [5,9],
    "blocks_process": 2,
    "point": [[3,1],[4,1]],
    "type": "toggle" }
```

Dòng đầu tiên mặc định là 4 số chứa thông tin số hàng, số cột và điểm bắt đầu của block
Tiếp đến là map với '.' thể hiện cho vực, '#' là đường có thể đi, G là đích đến, 'o' là sort_switch, 'x' là hard_switch, và '@' là teleport_switch. Tiếp đến là một số thể hiện số object trong map, và theo sau là object thể hiện ở dạng dictionary.

```
    def read_file(path):
    with open(path) as f:
        #  ====================  get level infomation =============
        first_line = f.readline()
        row, col, start_x, start_y = [int(x) for x in first_line.split()]
        #  ================= get level map =============
        game_map = []
        for i in range(row):
            map_line = f.readline().strip()
            list_map_line = map_line.split()
            game_map.append(list_map_line)
        #   ============== get level objects ==================
        number_of_object = int(f.readline())
        objects = []
        for i in range(number_of_object):
            lines = ""
            for k in range(5):
                line = f.readline().strip()
                lines += line
            obj = json.loads(lines)
            objects.append(obj)
        return row, col, start_x, start_y, game_map, objects
```

Phía trên là code xử lí file đầu vào. Với sự giúp đỡ của hàm json.loads việc dữ liệu vào khá dễ dàng

3.1.3 Xử lí cơ chế trong game

Class block:

```
    def __init__(self, x, y, status, prev, game_map, x_split=None, y_split=None, id=0):
        self.x = x
        self.y = y
        self.status = status
        self.prev = prev
        self.game_map = [x[:] for x in game_map]  # copy
        self.x_split = x_split
        self.y_split = y_split
        self.id = id
```

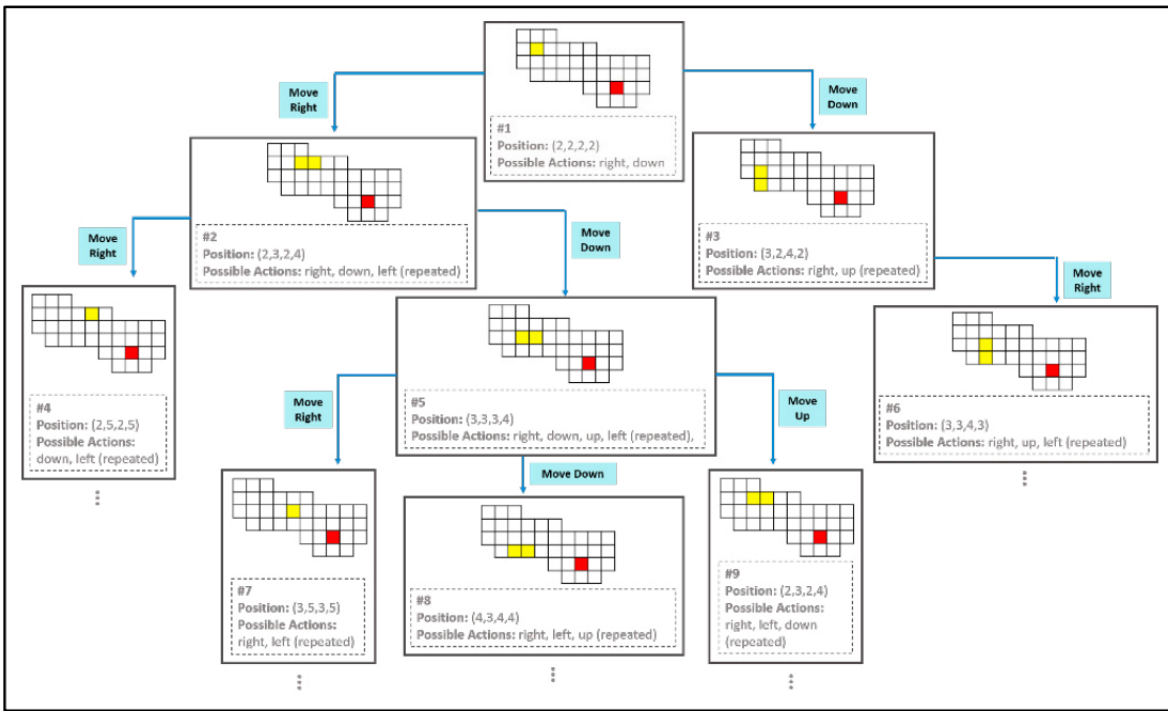
x, y chứa thông tin về vị trí hiện tại của block, nếu block đang nằm thì ưu tiên x, y nhỏ hơn. Status cho biết trạng thái của block gồm : 'STAND', 'LIE_HORIZONTAL', 'LIE_VERTICAL', 'SPLIT'. Prev lưu block parent của nó, game_map lưu map của game, x_split và y_split lưu thông tin về phần còn lại của block khi block ở trạng thái split, id lưu thông tin tiện cho việc debug
Module block.py chứa thông tin cơ bản về class block, có các hàm hỗ trợ block di chuyển. Ví dụ về một hàm trong class này:

```
    def move_up(self):
        global_variables.block_id += 1
        new_block = Block(self.x, self.y, self.status,
        self, self.game_map, id=global_variables.block_id)
        if self.status == "STAND":
            new_block.y -= 2
            new_block.status = "LIE_VERTICAL"
        elif self.status == "LIE_VERTICAL":
            new_block.y -= 1
            new_block.status = "STAND"
        elif self.status == "LIE_HORIZONTAL":
            new_block.y -= 1
```

Ở đây vị trí của block luôn ưu tiên lưu về phía trên và bên trái trong trương hợp block đang nằm
Trong một trạng thái bất kì khi ghi nhận vị trí hiện tại nhóm sẽ kiểm tra xem vị trí đó có hợp lệ không, nếu hợp lệ sẽ tiến hành xử lí xem block có đang đè lên công tắc hay không, nếu có sẽ tiến hành kiểm tra và thực hiện chức năng của công tắc tương ứng.

3.2 Hiện thực giải thuật BFS

Có 4 hướng di chuyển khi khối đang trong trạng thái "hợp nhất" và 8 hướng di chuyển khi khối đang bị tách rời. Trước hết sử dụng một queue để chứa các node/ state cần xử lí, duyệt qua từng state nếu có thể di chuyển theo bất kì hướng nào thì add vào queue và tiếp tục duyệt đến khi phát hiện được block đã vào lỗ hay chưa.
1. Lấy hết thông tin về điểm bắt đầu, đích đến
2. Duyệt state đầu tiên chính là trạng thái bắt đầu của game
3. Lấy phần tử trong queue ra để duyệt 4. Từ trạng thái này đưa block đến các trạng thái hợp lệ tiếp theo và lưu vào queue
5. Tiếp tục đến khi queue rỗng hoặc bất ngờ tìm được lời giải
6. Kết thúc



```
    def BFS(block):
    global_variables.previous = [block]  # save previous states
    queue = [block]
    solution = []
    while queue:
        current = queue.pop(0)
        if check_win(current):
            print("Success!\nFound solution after", current.id, "steps:")
            solution = solution_path(current)
            break

        if current.status != "SPLIT":  # if this is a complete block then it can move 4 directions
            add_move(queue, current.move_up())
            add_move(queue, current.move_right())
            add_move(queue, current.move_down())
            add_move(queue, current.move_left())

        else:
            add_move(queue, current.split_move_up())
            add_move(queue, current.split_move_right())
            add_move(queue, current.split_move_down())
            add_move(queue, current.split_move_left())

            add_move(queue, current.split_move_up_other())
            add_move(queue, current.split_move_right_other())
            add_move(queue, current.split_move_down_other())
            add_move(queue, current.split_move_left_other())
    return solution
```

Tạo một danh sách previous để lưu các trạng thái đã duyệt, một queue để chứa các trạng thái cần xử lí, solution để trả về kết quả tìm được.
Lưu ý trong hàm add_move nhóm sẽ kiểm tra xem trạng thái đó đã có trong list previous chưa nếu có rồi sẽ không add nữa

3.3 Genetic Algorithm

Thực hiện khai báo các biến:
– **population_num**: số dna sẽ tạo ra
– **mutation_rate**: tỉ lệ đột biến
– **no_of_moves**: số bước dự đoán cần để đến đích
– **max_tries**: số lần generation tối đa được tạo ra
Lần lượt tính fitness, kiểm tra, chọn và generate thế hệ tiếp theo cho đến khi tìm được kết quả hoặc vượt quá ngưỡng cho phép

```
    target = [start_point, g]
     population_num = 100
     mutation_rate = 0.05
     no_of_moves = 20
```

```
    max_tries = 10000
    population = Population(target, no_of_moves, mutation_rate, population_num)
    while (not population.finished) and population.generation < max_tries:
        new_block = copy.copy(block)
        population.calculate_fitness(new_block)
        population.evaluate()
        population.print_best_dna()
        population.selection()
        population.generate()
    return population.got
```

Trong hàm tính fitness ta sẽ cộng điểm nếu như block đi được và không đi lại bước cũ đồng thời cộng một số điểm tương ứng bằng khoảng cách từ điểm bắt đầu trừ đi khoảng cách từ điểm hiện tại đến đích nghĩa là block càng gần đích sẽ càng được nhiều điểm, block sẽ bị trừ điểm nếu trong nó có những hướng gây bất lợi cho block.

```
        def calculate_fitness(self, new_block, target):
        distance = sqrt((target[1][0] - new_block.x) ** 2 +
        (target[1][1] - new_block.y) ** 2)
        rate = 1.5
        blocked = 0
        open_space = 0  # Points for going into open spaces
        no_back_forth = 0  # Giving points for not going back and forth e.g DUDU...
        for i in range(len(self.genes) - 1):
            if self.genes[i] == self.U:
                new_block = new_block.move_up()
                if add_move_fitness(new_block):
                    open_space += 1
                    new_distance = sqrt((target[1][0] - new_block.x) ** 2 +
                    (target[1][1] - new_block.y) ** 2)
                    self.fitness += (distance - new_distance) * rate
                else:
                    new_block = new_block.move_down()
                    blocked += 1
            elif self.genes[i] == self.R:
                new_block = new_block.move_right()
                if add_move_fitness(new_block):
                    open_space += 1
                    new_distance = sqrt((target[1][0] - new_block.x) ** 2 +
                    (target[1][1] - new_block.y) ** 2)
                    self.fitness += (distance - new_distance) * rate
                else:
                    new_block = new_block.move_left()
                    blocked += 1
            elif self.genes[i] == self.D:
                new_block = new_block.move_down()
                if add_move_fitness(new_block):
                    open_space += 1
                    new_distance = sqrt((target[1][0] - new_block.x) ** 2 +
```

```
                    (target[1][1] - new_block.y) ** 2)
                    self.fitness += (distance - new_distance) * rate
                else:
                    new_block = new_block.move_up()
                    blocked += 1
            else:
                new_block = new_block.move_left()
                if add_move_fitness(new_block):
                    open_space += 1
                    new_distance = sqrt((target[1][0] - new_block.x) ** 2 +
                    (target[1][1] - new_block.y) ** 2)
                    self.fitness += (distance - new_distance) * rate
                else:
                    new_block = new_block.move_right()
                    blocked += 1
            if self.genes[i] != self.genes[i + 1]:
                if self.genes[i] in [self.U, self.D] and self.genes[i + 1] in [self.L, self.R]:
                    no_back_forth += 1
            if check_win(new_block):
                self.fitness += 10
                self.done = True
                return self.fitness

        self.fitness = open_space + no_back_forth - blocked
        return self.fitness
```

Hàm selection và generation thực hiện giống như một giải thuật ga bình thường nên nhóm xin phép không đưa vào bài báo cáo

4 Kết quả & đánh giá

4.1 BFS

Đối với giải thuật BFS là giải thuật blind search cho ra kết quả khá nhanh đối với nhưng map có độ phức tạp thấp tuy nhiên một số map nó xử lí khá chậm lên đến gần 20 s.

```
    Testing level 1  ..........
Success!
Found solution after 139 steps:
Level  1 :     Found solution in     0.005940676 s
================================================
Testing level 2  ..........
Success!
Found solution after 1001 steps:
Level  2 :     Found solution in     0.011714935 s
================================================
Testing level 3  ..........
Success!
```

```
Found solution after 385 steps:
Level  3 :     Found solution in     0.0 s
================================================
Testing level 4  ..........
Success!
Found solution after 271 steps:
Level  4 :     Found solution in     0.008135557 s
================================================
Testing level 5  ..........
Success!
Found solution after 1296 steps:
Level  5 :     Found solution in     0.010198355 s
================================================
Testing level 6  ..........
Success!
Found solution after 422 steps:
Level  6 :     Found solution in     0.004108429 s
================================================
Testing level 7  ..........
Success!
Found solution after 697 steps:
Level  7 :     Found solution in     0.006087065 s
================================================
Testing level 8  ..........
Success!
Found solution after 3230 steps:
Level  8 :     Found solution in     0.089859486 s
================================================
Testing level 9  ..........
Success!
Found solution after 5055 steps:
Level  9 :     Found solution in     0.106061697 s
================================================
Testing level 10  ..........
Success!
Found solution after 43988 steps:
Level  10 :     Found solution in     7.194708347 s
================================================
Testing level 11  ..........
Success!
Found solution after 772 steps:
Level  11 :     Found solution in     0.009455919 s
================================================
Testing level 12  ..........
Success!
Found solution after 1232 steps:
Level  12 :     Found solution in     0.013143778 s
================================================
Testing level 13  ..........
```

```
Success!
Found solution after 517 steps:
Level  13 :     Found solution in     0.0 s
================================================
Testing level 14  ..........
Success!
Found solution after 1834 steps:
Level  14 :     Found solution in     0.024152756 s
================================================
Testing level 15  ..........
Success!
Found solution after 4550 steps:
Level  15 :     Found solution in     0.079713821 s
================================================
Testing level 16  ..........
Success!
Found solution after 894 steps:
Level  16 :     Found solution in     0.010191917 s
================================================
Testing level 17  ..........
Success!
Found solution after 3906 steps:
Level  17 :     Found solution in     0.109969378 s
================================================
Testing level 18  ..........
Success!
Found solution after 2106 steps:
Level  18 :     Found solution in     0.050165653 s
================================================
Testing level 19  ..........
Success!
Found solution after 885 steps:
Level  19 :     Found solution in     0.010030985 s
================================================
Testing level 20  ..........
Success!
Found solution after 30135 steps:
Level  20 :     Found solution in     3.956735373 s
================================================
Testing level 21  ..........
Success!
Found solution after 926 steps:
Level  21 :     Found solution in     0.005043268 s
================================================
Testing level 22  ..........
Success!
Found solution after 1609 steps:
Level  22 :     Found solution in     0.020178556 s
================================================
```

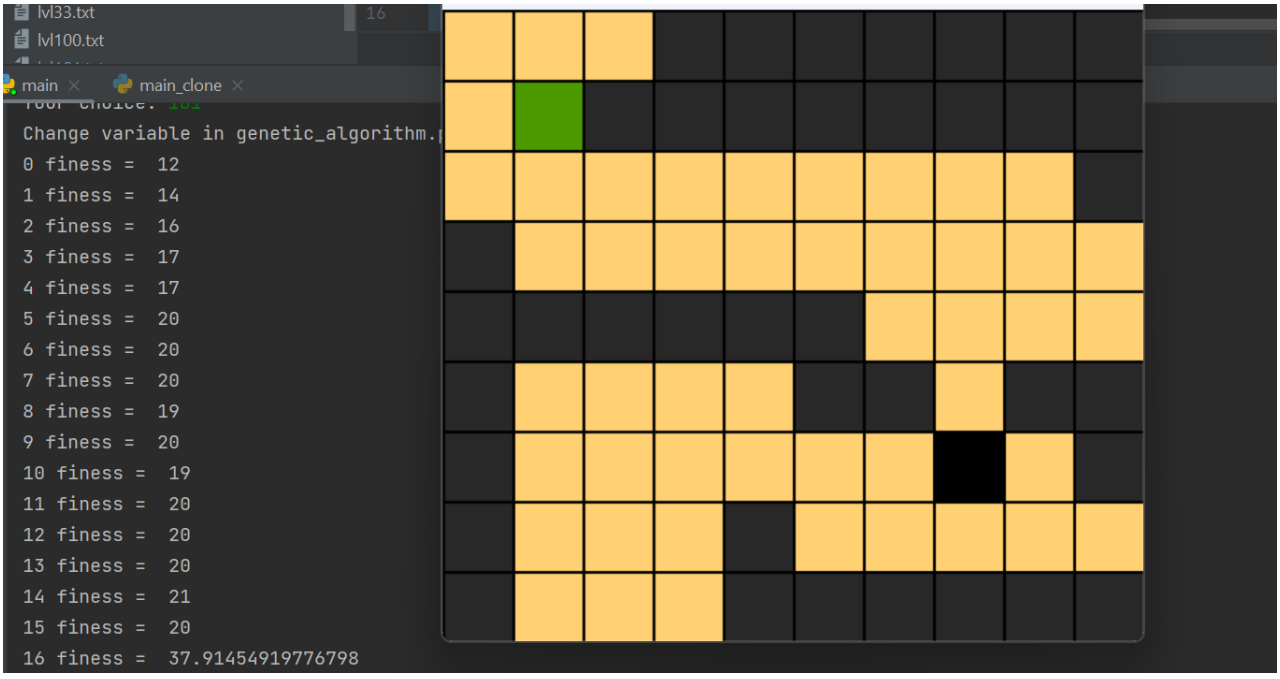
```
Testing level 23  ..........
Success!
Found solution after 9629 steps:
Level  23 :     Found solution in     0.566423178 s
================================================
Testing level 24  ..........
Success!
Found solution after 2502 steps:
Level  24 :     Found solution in     0.040104628 s
================================================
Testing level 25  ..........
Failed to find solution :(
================================================
Testing level 26  ..........
Success!
Found solution after 73316 steps:
Level  26 :     Found solution in     17.470047474 s
================================================
Testing level 27  ..........
Success!
Found solution after 2004 steps:
Level  27 :     Found solution in     0.024972677 s
================================================
Testing level 28  ..........
Success!
Found solution after 51421 steps:
Level  28 :     Found solution in     8.639822483 s
================================================
Testing level 29  ..........
Success!
Found solution after 7236 steps:
Level  29 :     Found solution in     0.29462719 s
================================================
Testing level 30  ..........
Success!
Found solution after 2216 steps:
Level  30 :     Found solution in     0.030214548 s
================================================
Testing level 31  ..........
Failed to find solution :(
================================================
Testing level 32  ..........
Success!
Found solution after 2511 steps:
Level  32 :     Found solution in     0.035142899 s
================================================
Testing level 33  ..........
Success!
Found solution after 2445 steps:
```

```
Level  33 :      Found solution in      0.044860363 s
================================================
31 /33 level success

Process finished with exit code 0
```

4.2 Genetic Algorithm

Đối với giải thuật Genetic thì một số màn tìm ra đáp án rất nhanh

```
    Change variable in genetic_algorithm.py to have the best performance
0 finess =  19
1 finess =  19
2 finess =  18
3 finess =  19
4 finess =  32.182878903867575

Process finished with exit code 0
```



Tuy nhiên một số màn mất khá nhiều thời gian để chạy nguyên nhân có thể do giá trị population, mutation rate không hợp lí, hiện tại nhóm vẫn đang tiếp tục nghiên cứu cải thiện giải thuật cũng như tối ưu thời gian thực thi chương trình

5 Tổng kết

Như vậy trong bài báo cáo lần này nhóm đã đi qua và làm rõ được những nội dung đã nêu ở phần mở đầu, cơ bản chương trình chạy ổn và được hoàn thiện ở mức khá. Tự đánh giá như sau:
Ưu điểm: về phần code, nhóm đã code theo đúng yêu cầu, code rõ ràng, chương trình thực thi được, chạy đúng

yêu cầu, có demo bằng pygame hiển thị từng bước đi
Nhược điểm: Phần genetic vẫn chưa chạy được trên nhiều màn
Vì trong báo cáo lần này có một số sinh viên bỏ môn nên báo cáo này chỉ có một người duy nhất làm, vậy nên không tránh khỏi những lỗi sai. Kính mong quý thầy cô thông cảm

6 File đính kèm

Source Code: Github: `https://github.com/duongnghia222/Bloxorz_game`
Video Demo: Youtube: `https://youtu.be/m8QhqvKYzrI`

Tài liệu

[1] Gameplay https://www.mathplayground.com/logic_bloxorz.html,

[2] BFS applied for bloxorz https://www.sciencedirect.com/science/article/pii/S187705091932160X?ref=pdf_download&fr=RR-2&rr=7690f1058e6b0ef4,