ĐẠI HỌC QUỐC GIA THÀNH PHỐ HỒ CHÍ MINH
TRƯỜNG ĐẠI HỌC BÁCH KHOA
KHOA KHOA HỌC - KỸ THUẬT MÁY TÍNH

MÔN HỌC: TRÍ TUỆ NHÂN TẠO

Giải game BLOXORZ Bằng giải thuật Depth First Search, Breadth First Search và Hill Climbing

---

Giáo viên hướng dẫn: ThS. Vương Bá Thịnh

Thành viên:
Lê Văn Rin - 1413235
Nguyễn Tường Vi - 1414636
Lê Văn Chi - 1510289
Vũ Văn Huynh - 1511328
Phùng Thị Ánh Nguyệt - 1412592

TP. HỒ CHÍ MINH, tháng 4 năm 2018

Mục lục

1 Giới thiệu

2 Quá trình tìm hiểu

2.1 Luật game

Luật game được mô tả như sau [1]:

- Mục đích của game là để khối trụ rơi vào lỗ thoát vào cuối mỗi màn chơi. Có 33 màn để hoàn thành.
- Sử dụng các phím lên, xuống, trái, phải để di chuyển khối trụ. Cẩn thận không để khối trụ rơi ra khỏi cảnh không thì sẽ bị chơi lại.
- Có hai loại thiết bị chuyển mạch chính:
  - Công tắc □ : Bất kỳ phần nào của khối trụ nhấn xuống đều được kích hoạt.
  - Công tắc **X** : Đòi hỏi áp lực nhiều hơn, nên khối trụ phải đứng dọc lên nó mới có thể kích hoạt.
- Khi công tắc được kích hoạt, mỗi công tắc có thể hoạt động khác nhau, một số sẽ chuyển đổi những cây cầu từ đóng - mở hoặc ngược lại. Hoặc một số sẽ đóng - mở cầu vĩnh viễn.
- Những lát gạch **màu cam** rất mong manh. Nếu khối trụ đứng dọc lên nó, gạch sẽ bị vỡ và khối trụ sẽ bị rơi xuống, đồng nghĩa với việc bị chơi lại.
- Cuối cùng, có một loại công tắc thứ ba (◯), nó dịch chuyển khối trụ và tách khối trụ thành hai khối lập phương cùng lúc. Chúng sẽ được kiểm soát riêng lẽ bằng phím Space và gộp lại một khối bình thường khi chúng đứng cạnh nhau.
- Lưu ý, khối nhỏ lập phương chỉ có thể kích hoạt công tắc □, không thể kích hoạt công tắc **X**, và chúng cũng không thể đi qua lỗ thoát để kết thúc màn chơi. Chỉ có khối trụ hoàn chỉnh mới làm được điều này.

2.2 Giải thuật

2.2.1 Depth First Search [3]

- Giải thuật được mô tả như sau:

1. Tập Open chứa trạng thái gốc s chờ được xét, lưu trữ theo cấu trúc ngăn xếp (stack)
2. Kiểm tra tập OPEN có rỗng không.
   (a) Nếu tập OPEN không rỗng, lấy một đỉnh ra khỏi tập OPEN làm đỉnh đang xét p (Sang bước 3). Nếu p là đỉnh g cần tìm, kết thúc tìm kiếm.
   (b) Nếu tập OPEN rỗng, tiến đến bước 4.
3. Đưa đỉnh p vào tập CLOSED, sau đó xác định các đỉnh kề với đỉnh p vừa xét. Nếu các đỉnh kề không thuộc tập CLOSED, đưa chúng vào đầu tập OPEN. Quay lại bước 2.
4. Kết luận không tìm ra đỉnh đích cần tìm.

2.2.2 Breadth First Search [3]

- Từ đỉnh xuất phát duyệt tất cả các đỉnh kề với đỉnh này sau đó làm vậy với các đỉnh vừa được duyệt. Quá trình kết thúc khi tìm được trạng thái mục tiêu hoặc đã duyệt hết đồ thị mà không tìm thấy.
- Ta có chi tiết của giải thuật:
  1. Đưa trạng thái khởi đầu vào hàng đợi OPEN
  2. Lặp cho đến khi đạt trạng thái mục tiêu hoặc không còn trạng thái nào trong OPEN:
     (a) Lấy ra trạng thái trong hàng đợi OPEN
     (b) Lưu nó vào tập đã duyệt CLOSED
     (c) Sinh ra các trạng thái tiếp theo của nó.
     (d) Với mỗi trạng thái tiếp theo , nếu là trạng thái mới ghi nhớ trạng thái cha sinh ra nó , kiểm tra xem nó đã được duyệt qua trong tập CLOSED chưa và nếu nó chưa có trong CLOSED thì đưa nó vào hàng đợi OPEN (Quay lại bước 2)
  3. Trả về trạng thái mục tiêu và danh sách trạng thái đã duyệt

2.2.3 Hill Climbing [4]

- Giải thuật leo đồi (*hillclimbing*) dựa theo ý tưởng của việc một người phải leo từ dưới mặt đất lên đến đỉnh cao nhất trong các ngọn đồi mà không nhìn thấy trước được đỉnh đó ở đâu.Khi đó kinh nghiệm được áp dụng là làm thế nào để, sau mỗi bước di chuyển,vị trí mới cao hơn vị trí cũ.Như vậy hi vọng người đó dần sẽ lên đến được đỉnh.Tức là,từ

mỗi vị trí đang đứng,chỉ thử một hướng đi dẫn đến một vị trí cao hơn, thay vì thử tất cả các hướng đi có thể.

- Giải thuật này sử dụng một hàm kinh nghiệm (*heuristicfunction*) hay hàm lượng giá (*evaluation*) để ước lượng xem mỗi trạng thái gần với trạng thái mục tiêu như thế nào.Bước đi nào dẫn đến một trạng thái gần với trạng thái mục tiêu hơn so với trạng thái hiện tại sẽ được chọn để thực hiện.Hàm lượng giá là một cách để đưa tri thức cụ thể cho nhiệm vụ của bài toán vào trong quá trình điều khiển.

- Ta có giải thuật leo đồi đơn giản:
  1. Lượng giá trạng thái khởi đầu
  2. Lặp cho đến khi đạt trạng thái mục tiêu hoặc không còn tác vụ để thử:
     - Chọn một tác vụ để thử
     - Lượng giá trạng thái mới do tác vụ sinh ra.
     - Nếu đó là trạng thái mục tiêu thì kết thúc;
       Nếu trạng thái mới tốt hơn trạng thái hiện tại thì chuyển sang trạng thái mới
- Một biến thể của giải thuật leo đồi đơn giản là leo đồi dốc nhất(*steepest*−*ascenthillclimbing*).Ý tưởng là thay vì chuyển đến bất kì trạng thái nào tốt hơn trạng thái hiện tại giải thuật chuyển đến trạng thái tốt nhất trong số các trạng thái tốt hơn đó.
- Giải thuật leo đồi dốc nhất:
  1. Lượng giá trạng thái khởi đầu
  2. Lặp cho đến khi đạt trạng thái mục tiêu hoặc không chuyển đến được một trạng thái mới:
     - Với mỗi tác vụ lượng giá trạng thái do tác vụ sinh ra;nếu đó là trạng thái mục tiêu thì kết thúc.
     - Chọn trạng thái tốt nhất trong sô các trạng thái do các tác vụ sinh ra.Nếu nó tốt hơn trạng thái hiện tại thì chuyển sang trạng thái đó
- Nhược điểm

1. Tối ưu cục bộ: trạng thái đạt đến khi giải thuật kết thúc có thể chỉ là trạng thái tốt nhất so với các trạng thái gần xung quanh, chứ không phải tốt nhất toàn cục.
   Khắc phục: Quay lui các trạng thái trước và sử dụng các lựa chọn khác, vì một trạng thái có thể có nhiều trạng thái có giá trị tốt hơn
2. Rơi vào đồng bằng: Giải thuật có thể dẫn đến trạng thái có cùng giá trị như tất cả các trạng thái tiếp theo và sẽ kết thúc tại đó mà chưa tìm được trạng thái mục tiêu.
   Khắc phục : Thực hiện nhiều tác vụ để chuyển trạng thái liên tiếp theo cùng một hướng, thực hiện bước nhảy dài để có thể vượt qua vùng đồng bằng
3. Rơi vào rìa đồi: Tất cả các trạng thái tiếp theo đều xấu hơn trạng thái hiện tại,giải thuật kết thúc.
   khắc phục: Chuyển trạng thái liên tiếp theo các hướng khác nhau, leo xuống đồi theo một hướng rồi leo lên lại theo hướng khác.

(a) Tối ưu cục bộ (b) Đồng bằng (c) Rìa đồi

Hình 1: Các trường hợp Hill Climbing thất bại

3 Quá trình hiện thực

3.1 Chuẩn hóa Input/Output

Sau khi tìm hiểu luật game và các giải thuật chúng tôi quyết định dùng định dạng chuẩn JSON để chuẩn hóa Input cho bài toán và Output sẽ là tập hợp tất cả các vị trí đường đi của khối trụ từ điểm bắt đầu đến điểm đích.

Lý do nhóm chọn chuẩn JSON là vì tính tùy biến cao và dễ dàng cấu hình để thể hiện logic của game.

Cụ thể Input được đặc tả bằng các tag:value JSON như sau:

- **"level"** : number - số thự tự của màn chơi (IntType)
- **"size"** : [x, y] - là chiều cao và chiều rộng của Maps (IntType)

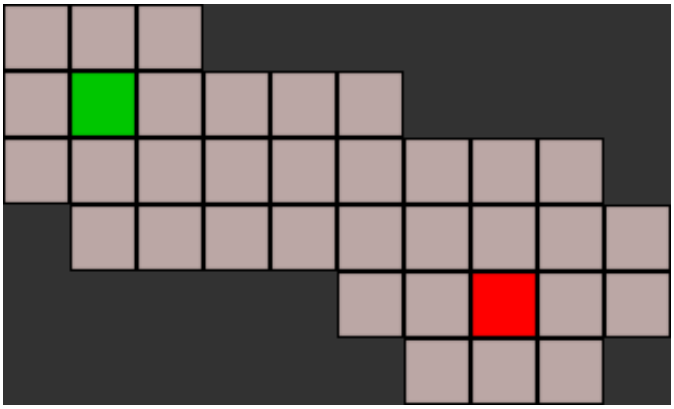
- **"swX"** : - Định danh Công tắc **X**
  - **"count"**: number - số lượng công tắc **X** có trên Maps (IntType)
  - **"symbol"** : [x0, x1,..., xN] - x là ký tự đại diện cho 1 công tắc **X** trên Maps (StringType)
  - **"x0"** :
    + **"type"** : 0 hoặc 1. Nếu là 0 thì công tắc ở chế độ có thể chuyển đổi trạng thái liên tục, 1 thì chỉ chuyển đổi trạng thái 1 lần vĩnh viễn (IntType)
    + **"active"**: true hoặc false. Nếu là true thì công tắc đang được kích hoạt, false là chưa kích hoạt (BoolType)
    + **"location"**: [x, y] - x, y là vị trí của công tắc nằm trên Maps (IntType)
    + **"bridge"**: name - ký tự đại diện cho cầu mà công tắc sẽ kích hoạt (StringType)
  - **"itemN"** - ...
- **"swQ"** : - Định danh công tắc □, cấu hình tương tự "swX".
- **"swO"** : - Định danh công tắc ◯ **(Tạm thời nhóm chưa hiện thực được công tắc tách đôi khối trụ, có thể bỏ qua tag này)**
- **"bridge"** : - Định danh những cầu nối.
  - **"count"** : number - số lượng cầu có trên Maps (IntType)
  - **"symbol"**: [c0, c1,..., cN] - c là ký tự đại diện cho mỗi cầu nối (StringType)
  - **"c0"**:
    + **"times_active"**: number - số lần cầu được kích hoạt, mặc định 0 (IntType)
    + **"active"**: false hoặc true - Thể hiện trạng thái cầu đang được kích hoạt hay không (BoolType)
    + **"location"**: [[x1, y1][x2, y2]] - vị trí của cầu nối, nếu cầu chỉ có 1 viên gạch thì để [[x1, y1]] (IntType)
  - **"start"** : [x, y] - Vị trí bắt đầu game (IntType)
  - **"end"** : [i, j] - Vị trí kết thúc game (IntType)
  - **"box"** :
    + **"state"** : 1 hoặc 2. Nếu là 1 là khối lập phương, 2 là khối trụ. (Mặc định là 2)

  + **"symbol"** : ký tự đại diện cho khối trụ (mặc định là "#")
  + **"location"** : [[x, y]] vị trí bắt đầu của khối trụ (mặc định là [[1, 1]])
- **"tiles"** : chất liệu gạch {"orange": 2,"rock": 1,"space": 0} (mặc định)
- **"maps"** : [] - mảng 2 chiều mô tả bề mặt của Maps, gồm các giá trị 0, 1 hoặc 2.

Ví dụ:
Với một JSON input đơn giản sau:

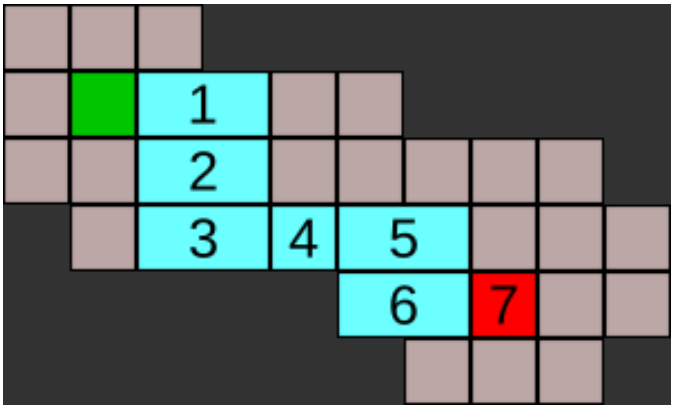
```
{"level": 0,
"size": [6,10],
"swX": "count": 0,"symbol": [],
"swQ": "count": 0,"symbol": [],
"swO": "count": 0,"symbol": [],
"bridge": "count": 0,"symbol": [],
"start": [1,1],"end": [4,7],
"box": "state": 2,"symbol": "#","location": [[1,1]],
"tiles": "orange": 2,"rock": 1,"space": 0,
"maps": [
[1,1,1,0,0,0,0,0,0,0],
[1,1,1,1,1,1,0,0,0,0],
[1,1,1,1,1,1,1,1,1,0],
[0,1,1,1,1,1,1,1,1,1],
[0,0,0,0,0,1,1,0,1,1],
[0,0,0,0,0,0,1,1,1,0]]}
```

Sẽ biểu diễn cho Maps:



Hình 2: Hình vẽ trực quang của Input [5]

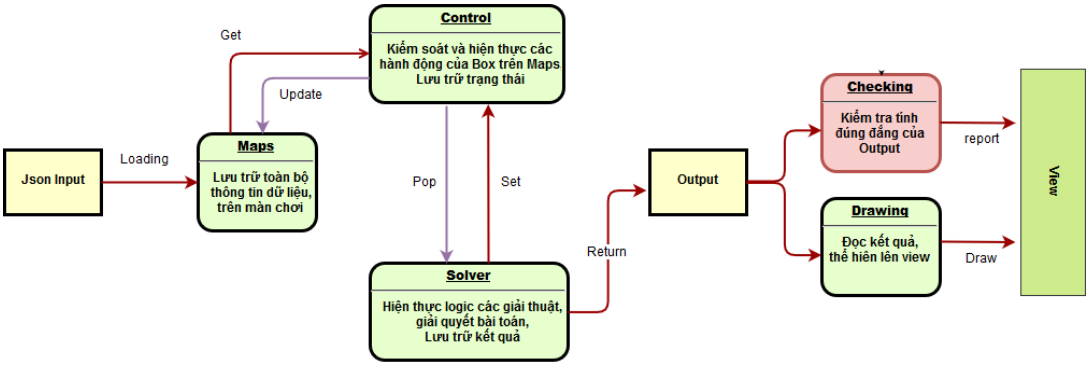
Output : [[1, 1]] - [[1, 2], [1, 3]] - [[2, 2], [2, 3]] - [[3, 2], [3, 3]] - [[3, 4]] - [[3, 5], [3, 6]] - [[4, 5], [4, 6]] - [[4, 7]]



Hình 3: Hình vẽ trực quang của Output [5]

3.2 Mô hình hóa bài toán

Sau khi nghiên cứu luật game và đã chuẩn hóa được input, chúng tôi tiến hành mô hình bài toán để triển khai coding.



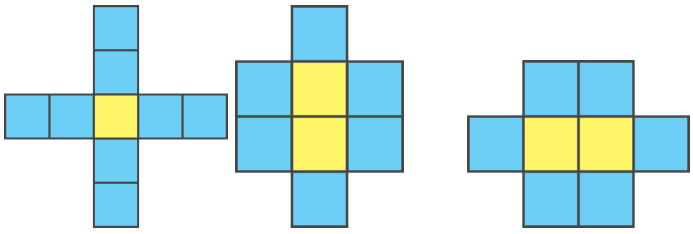
Hình 4: Mô hình giải quyết bài toán

3.3 Phân tích nghiệp vụ

3.3.1 Trạng thái của khối trụ

Khối trụ di chuyển trên maps, và chuyển trạng thái với 4 thao tác chính: UP, DOWN, RIGHT, LEFT.

Với mỗi trường hợp khác nhau, thì khối trụ sẽ chuyển trạng thái vị trí theo 3 quy luật dưới đây:



(a) Trường hợp 1 (b) Trường hợp 2 (c) Trường hợp 3

Hình 5: Các trạng thái của khối trụ

- Trường hợp 1: Khối trụ đứng thẳng trên 1 tọa độ [i, j]
  - UP: tọa độ mới [i - 1, j] và [i - 2, j]
  - DOWN: tọa độ mới [i + 1, j] và [i + 2, j]
  - RIGHT: tọa độ mới [i, j + 1] và [i, j + 2]
  - LEFT: tọa độ mới [i, j - 1] và [i, j - 2]
- Trường hợp 2: Khối trụ nằm dọc trên 2 tọa độ [i, j] [i + 1, j]
  - UP: tọa độ mới [i - 1, j]
  - DOWN: tọa độ mới [i + 2, j]
  - RIGHT: tọa độ mới [i, j + 1] và [i + 1, j + 1]
  - LEFT: tọa độ mới [i, j - 1] và [i + 1, j - 1]
- Trường hợp 3: Khối trụ nằm ngang trên 2 tọa độ [i, j] [i, j + 1]
  - UP: tọa độ mới [i - 1, j] và [i - 1, j + 1]
  - DOWN: tọa độ mới [i + 1, j] và [i + 1, j + 1]
  - RIGHT: tọa độ mới [i, j + 2]
  - LEFT: tọa độ mới [i, j - 1]

Khối trụ sẽ **mất hiệu lực** khi rơi vào các trường hợp:

- Tọa độ của khối vượt quá giới giạn kích thước của maps.
- Tọa độ của khối rơi vào vị trí biên hoặc rỗng của maps (ngoại trừ lỗ thoát vị trí đích).
- Khi khối trụ nằm đứng (Trường hợp 1) tại vị trí có **lát gạch màu cam**.

Trạng thái đích là khi **tọa độ của khối trụ trùng với tọa độ đích**.

3.3.2 Giải quyết bài toán bằng giải thuật

Vì trạng thái map có thể bị thay đổi sau mỗi bước đi của khối trụ nếu như nó tác động lên những switch, nên với mỗi trạng thái được lưu lại ta cần 2 giá trị (cur_location, cur_maps) để đảm bảo tính đúng đắng cho solution khi được tìm ra, bởi nếu không lưu trữ lại trạng thái map, thì giải thuật sẽ chạy trên cùng 1 trạng thái map từ đầu đến cuối mà không thể quay lùi nếu một bước nào đó không đúng, dẫn đến sai solution.

- cur_location: vị trí của khối trụ hiện tại
- cur_maps: trạng thái map hiện tại

Depth First Search

1. Trạng thái bắt đầu là (cur_location, cur_maps) tại vị trí bắt đầu của khối trụ, cho vào tập OPEN lưu trữ theo cấu trúc ngắn xếp (stack).
2. Lặp cho đến khi nào tập OPEN rỗng:
   (a) Lấy ra trạng thái trong ngăn xếp OPEN
   (b) Lưu nó vào tập đã duyệt CLOSED
   (c) Thực hiện các phép chuyển trạng thái UP, DOWN, RIGHT, LEFT. Sinh ra các trạng thái mới của nó nếu có (Sang bước (d)), nếu không có thì (Quay lại bước 2).
   (d) Với mỗi trạng thái mới, kiểm tra xem nó đã có trong tập CLOSED chưa, nếu chưa có thì đưa nó vào ngắn xếp OPEN. **Nếu nó là trạng thái mục tiêu thì kết thúc tìm kiếm**, nếu không (Quay lại bước 2)
3. Kết luận không tìm thấy trạng thái mục tiêu.

Breadth First search

1. Trạng thái bắt đầu là (cur_location, cur_maps) tại vị trí bắt đầu của khối trụ, cho vào tập OPEN lưu trữ theo cấu trúc hàng đợi (queue).
2. Lặp cho đến khi nào tập OPEN rỗng:
   (a) Lấy ra trạng thái trong hàng đợi OPEN
   (b) Lưu nó vào tập đã duyệt CLOSED
   (c) Thực hiện các phép chuyển trạng thái UP, DOWN, RIGHT, LEFT. Sinh ra các trạng thái mới của nó nếu có (Sang bước (d)), nếu không có thì (Quay lại bước 2).

(d) Với mỗi trạng thái mới, kiểm tra xem nó đã có trong tập CLOSED chưa, nếu chưa có thì đưa nó vào hàng đợi OPEN. **Nếu nó là trạng thái mục tiêu thì kết thúc tìm kiếm**, nếu không (Quay lại bước 2)

3. Kết luận không tìm thấy trạng thái mục tiêu.

Hill Climbing

Hàm lượng giá mà chúng tôi chọn cho giải thuật leo đồi là tính khoảng cách tại vị trí hiện tại của khối trụ đến vị trí của trạng thái mục tiêu.

- Vị trí của khối trụ: box(i, j)
- Vị trí mục tiêu: goal(x, y)
- Khoảng cách: Distance(box, goal)

Nếu Distance(box, goal) **càng nhỏ** thì trạng thái đó **càng tốt**. Trạng thái tốt nhất là khi Distance(box, goal) = 0, là lúc vị trí của box trùng với vị trí goal.

Nhóm hiện thực giải thuật leo đồi dốc nhất để giải quyết bài toán:

1. Lượng giá toàn bộ vị trí trên map so với vị trí mục tiêu. Lưu vào một map mới (Eval_Maps).
2. Lặp cho đến khi nào đạt được trạng thái mục tiêu hoặc không chuyển đến được một trạng thái mới:
   (a) Trạng thái hiện tại được xét.
   (b) Thực hiện các phép chuyển trạng thái UP, DOWN, LEFT, RIGHT. Với mỗi trạng thái mới:
      - Lấy giá trị lượng giá của trạng thái mới eNew trong Eval_Maps, so sánh với giá trị lượng giá của trạng thái hiện tại eCurr, nếu eNew <= eCurr, thì cho trạng thái mới vào tập ACCEPTED.

   (c) Nếu tập ACCEPTED khác rỗng, thì tìm MIN của giá trị lượng giá tập ACCEPTED, lấy ra trạng thái MIN đó (Cho vào tập đã chọn), nếu trạng thái đó là trạng thái mục tiêu - **kết thúc tìm kiếm**, nếu không thì chọn làm trạng thái tiếp theo. Lấy toàn bộ trạng thái trong tập ACCEPTED và sắp xếp lại từ bé - lớn cho vào tập ALL_ACCEPTED.(Quay lại bước 2)
   (d) Nếu tập ACCEPTED rỗng. (Rơi vào problems của giải thuật leo đồi)

- Nếu tập ALL_ACCEPTED rỗng sang bước 3, nếu không, lấy ra trạng thái đỉnh của tập ALL_ACCEPTED. (*)
- Nếu trạng thái đã có trong tập đã chọn, lặp lại bước (*), nếu chưa có, chọn trạng thái đó làm trạng thái tiếp theo (Quay lại bước 2).

3. Kết luận không tìm thấy trạng thái mục tiêu.

4 Chạy chương trình

4.1 Cài đặt môi trường và thư viện hỗ trợ

Yêu cầu môi trường : Python 3.5 trở lên, Pip Package version mới nhất.

Để cập nhật pip package mới nhất:

```
$ pip install --upgrade pip
```

Cài đặt thư viện cần thiết: **simplejson**, **pygame**, **PyOpenGL**, **numpy**

Vào thư mục **bloxorz** chạy lệnh

```
$ pip install -r requirement.txt
```

4.2 Thực thi chương trình

Vào thư mục **bloxorz**, source code mà nhóm đã cung cấp, thực thi các lệnh sau:

```
$ cd ./bloxorz
$ python run.py path-to-level options view
```

Trong đó:

- **path-to-level**: là đường dẫn đến file json config của màn chơi đã có trong thư mục **./bloxorz/level**
- **options**: là giải thuật muốn kiểm tra, gồm các giá trị: **dfs, bfs, hill** hoặc **handle** (chơi bằng tay)
- **view**: là chế độ xem

- 1: là xem toàn bộ step by step mà giải thuật chạy đến lúc kết thúc (Trên cả Consoles và 3D View)
- 2: là chỉ xem kết quả tìm kiếm của giải thuật (Trên Consoles)
- 3: là chỉ xem kết quả tìm kiếm của giải thuật (3D View)

Ví dụ:

Muốn kiểm tra màn chơi có trong thư mục **./level/1.json**, bằng giải thuật DFS và chế độ xem 3D, gõ lệnh như sau

```
$ python run.py ./level/1.json dfs 3
```

Muốn chơi bằng tay:

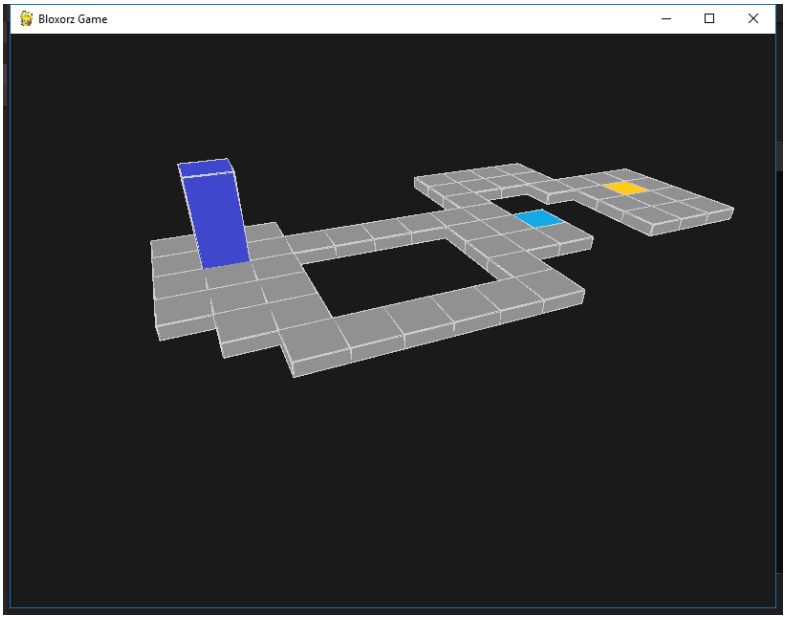
```
$ python run.py ./level/1.json handle
```

Ngoài ra, nếu không muốn thực hiện nhập thông số trên terminal, có thể vào file run.py và tùy chỉnh thông số của hàm main() tại line 174:

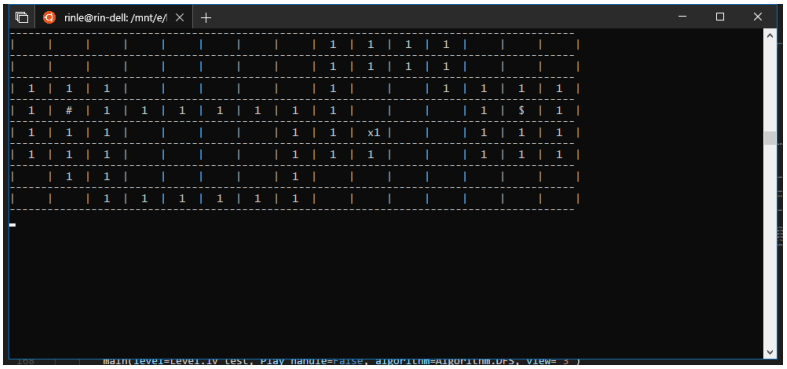
main(level=Level.lv1, Play_handle=False, algorithm=Algorithm.DFS, view='3')

Sau đó thực hiện lệnh:

```
$ python run.py
```



(a) Màn hình 3D View



(b) Màn hình Consoles

Hình 6: Màn hình Game

5 Đánh giá hiệu năng

Chạy chương trình trên 15 map, ta có thời gian và số bước đi của mỗi giải thuật như sau:

- **time**: thời gian tìm ra solution của giải thuật, tính bằng s
- **move**: số bước đi của solution để tới đích
- **N**: None, giải thuật không tìm được đường đi tới đích

- Lưu ý: Thời gian chạy giải thuật là khác nhau trên mỗi thiết bị, tùy thuộc vào trình compiler, hệ điều hành (Linux, Window, MasOS) và cấu hình phần cứng, số liệu bên dưới được chạy trên HĐH Window 7, RAM 4G, Core i5 2.5GHz.

	Level 1		Level 2		Level 3		Level 4		Level 5	
	Time	Move	Time	Move	Time	Move	Time	Move	Time	Move
HILL	0.354	7	0.328	12	0.307	12	0.877	16	0.928	N
DFS	0.34	32	1.562	70	1.872	70	1.325	51	1	41
BFS	0.316	7	2.393	12	2.189	12	3.871	16	1.431	18

	Level 6		Level 7		Level 8		Level 9		Level 10	
	Time	Move	Time	Move	Time	Move	Time	Move	Time	Move
HILL	0.037	N	1.28	N	1.422	N	0.655	N	1.053	N
DFS	1.159	51	1.487	63	1.474	63	2.208	47	1.407	69
BFS	0.799	29	4.505	24	4.167	24	4.025	36	2.464	27

	Level 11		Level 12		Level 13		Level 14		Level 15	
	Time	Move	Time	Move	Time	Move	Time	Move	Time	Move
HILL	1.164	12	0.165	N	0.253	8	0.248	8	0.517	11
DFS	1.806	86	0.259	26	0.253	26	0.37	32	0.842	64
BFS	2.812	12	1.301	15	0.404	8	0.359	8	1.603	9

Đánh giá kết quả:

- **HILL**:
  - Trong trường hợp map thuận lợi(không bị rơi vào đồng bằng, rơi vào rìa đồi, tối ưu cục bộ), đa số HILL tìm ra đường đi nhanh nhất với số bước đi ngắn nhất.
  - Trong trường hợp bị rơi vào các exception, giải thuật HILL đa số không tìm được đích hoặc dừng lại ở vị trí chưa phải đích.
  - Đối với các map có switch, nếu các switch nằm ở vị trí mà theo hàm lượng giá thì vị trí đó được chọn thì giải thuật tìm được solution, ngược lại không tìm được solution.
  - Vấn đề chọn hàm lượng giá đúng đắn cho HILL với game bloxorz là khá khó khăn, vì một số map có switch nếu phải cần đi qua những switch mới đến đích được, khi đó hàm lượng giá phải xác định được switch nào ưu tiên hơn, lúc đó mới tối ưu được bài toán.
- **DFS**:

  - Dựa vào bảng kết quả , DFS với thời gian trung bình trong 3 giải thuật và có số bước đi lớn nhất
  - Đối với các map có switch:
    * Khi switch nằm trên vị trí thuận lợi trên đường đi, giải thuật tìm được solution.
    * Khi giải thuật chưa đi qua switch, nhưng vị trí switch bị vây bởi các vị trí đã ở trong tập CLOSED thì không thể đến được vị trí switch nữa nên giải thuật dừng.
    * Khi đã đi qua switch(cầu mở), nhưng trong một bước đi nào đó lại tiếp tục đi qua switch lần 2(cầu đóng), trường hợp này giải thuật cũng không đi được tới đích.

- **BFS**:
  - Dựa vào bảng kết quả , BFS với thời gian trung bình lớn nhất trong 3 giải thuật và có số bước đi bằng hoặc nhỏ hơn HILL
  - Đối với các map có switch:
    * Khi switch nằm trên vị trí thuận lợi trên đường đi, giải thuật tìm được solution.
    * Khi giải thuật chưa đi qua switch, nhưng vị trí switch bị vây bởi các vị trí đã ở trong tập CLOSED thì không thể đến được vị trí switch nữa nên giải thuật dừng.
    * Khi đã đi qua switch(cầu mở), nhưng trong một bước đi nào đó lại tiếp tục đi qua switch lần 2(cầu đóng), trường hợp này giải thuật cũng không đi được tới đích.

Kết luận:

- Khi map không có switch, giải thuật HILL là tốt nhất với thời gian và số bước đi
- Khi map có switch, giải thuật BFS là tốt nhất về số bước đi, DFS tốt nhất về mặt thời gian tìm solution
- Khi chọn được hàm lượng giá tốt và xử lý tất cả exception, HILL là giải thuật tối ưu nhất trong 3 giải thuật ở cả hai trường hợp có switch và không switch.

Tài liệu

[1] *Luật Game Bloxorz*
http://www.papou.byethost9.com /game/bloxorz.html?i=1#mozTocId707172

[2] *Draw Cube By Pygame and PyOpenGL*
https://pythonprogramming.net/opengl-rotating-cube-example-pyopengl-tutorial/

[3] *Sách Phân tích và thiết kế giải thuật*
Dương Tuấn Anh (2016)

[4] *Sách Nhập môn trí tuệ nhân tạo. Tr.25 mục 3.2*
Cao Hoàng Trụ - xuất bản năm 2014

[5] *Hình vẽ mô tả input/output*
https://www.codewars.com/kata/bloxorz-solver/python