ECS 170 Programming Assignment 1
Code Infrastructure & Requirements

Zheng Fang

UC Davis

Jan 25, 2017

Outline

1. Heuristic Search Review
2. Code Infrastructure
3. Assignment Requirements & Grading Policy

1 Heuristic Search Review

2 Code Infrastructure

3 Assignment Requirements & Grading Policy

Heuristic Search

Heuristic Search

Heuristic search refers to the search techniques that utilize the information about the preference of states with respect to achieve a goal state.

- i.e. heuristic search knows whether one non-goal state is "more promising" than another

Heuristic Search

Heuristic Search

Heuristic search refers to the search techniques that utilize the information about the preference of states with respect to achieve a goal state.

- i.e. heuristic search knows whether one non-goal state is "more promising" than another

Heuristic

- A "heuristic" refers to a "heuristic function" that maps a state to a value
- $h(n) =$ estimated cost of the cheapest path from node n to a goal node.

Heuristic

- A "heuristic" refers to a "heuristic function" that maps a state to a value
- $h(n) =$ estimated cost of the cheapest path from node n to a goal node.

Admissible Heuristic

Admissible Heuristic

A heuristic $h(n)$ is admissible if and only if $h(n)$ *never overestimate* the cost to reach the goal.

- i.e. $h(n)$ is by nature optimistic and provide a lower bound for the cost to reach the goal.

- A* is optimal if $h(n)$ is an admissible heuristic (using TREE-SEARCH? Why?)
- Read P97-99 on AIMA 2e

Admissible Heuristic

Admissible Heuristic

A heuristic $h(n)$ is admissible if and only if $h(n)$ *never overestimate* the cost to reach the goal.

- i.e. $h(n)$ is by nature optimistic and provide a lower bound for the cost to reach the goal.

- A* is optimal if $h(n)$ is an admissible heuristic (using TREE-SEARCH? Why?)
- Read P97-99 on AIMA 2e

Admissible Heuristic

Admissible Heuristic

A heuristic $h(n)$ is admissible if and only if $h(n)$ *never overestimate* the cost to reach the goal.

- i.e. $h(n)$ is by nature optimistic and provide a lower bound for the cost to reach the goal.

- A* is optimal if $h(n)$ is an admissible heuristic (using TREE-SEARCH? Why?)
- Read P97-99 on AIMA 2e

Admissible Heuristic

Admissible Heuristic

A heuristic $h(n)$ is admissible if and only if $h(n)$ *never overestimate* the cost to reach the goal.

- i.e. $h(n)$ is by nature optimistic and provide a lower bound for the cost to reach the goal.

- A* is optimal if $h(n)$ is an admissible heuristic (using TREE-SEARCH? Why?)
- Read P97-99 on AIMA 2e

1 Heuristic Search Review

2 Code Infrastructure

3 Assignment Requirements & Grading Policy

Overview

All relevant modules:

- Main : primary program entry *
- AIModule : Base class of all AI implmentations
- StupidAI : Sample class of AI implementaion ***
- DijkstraAI : Sample class of AI implementaion, Demo use only, no source code.
- TerrainMap : A class representing the world, start and end points, and visited squares. ***
- PerlinTerrainGenerator : A class that generates random terrains using Perlin noise functions. *

Overview

All relevant modules:

- Main : primary program entry *
- AIModule : Base class of all AI implmentations
- StupidAI : Sample class of AI implementaion ***
- DijkstraAI : Sample class of AI implementaion, Demo use only, no source code.
- TerrainMap : A class representing the world, start and end points, and visited squares. ***
- PerlinTerrainGenerator : A class that generates random terrains using Perlin noise functions. *

Overview

All relevant modules:

- Main : primary program entry *
- AIModule : Base class of all AI implmentations
- StupidAI : Sample class of AI implementaion ***
- DijkstraAI : Sample class of AI implementaion, Demo use only, no source code.
- TerrainMap : A class representing the world, start and end points, and visited squares. ***
- PerlinTerrainGenerator : A class that generates random terrains using Perlin noise functions. *

Overview

All relevant modules:

- Main : primary program entry *
- AIModule : Base class of all AI implmentations
- StupidAI : Sample class of AI implementaion ***
- DijkstraAI : Sample class of AI implementaion, Demo use only, no source code.
- TerrainMap : A class representing the world, start and end points, and visited squares. ***
- PerlinTerrainGenerator : A class that generates random terrains using Perlin noise functions. *

Overview

All relevant modules:

- Main : primary program entry *
- AIModule : Base class of all AI implmentations
- StupidAI : Sample class of AI implementaion ***
- DijkstraAI : Sample class of AI implementaion, Demo use only, no source code.
- TerrainMap : A class representing the world, start and end points, and visited squares. ***
- PerlinTerrainGenerator : A class that generates random terrains using Perlin noise functions. *

Overview

All relevant modules:

- Main : primary program entry *
- AIModule : Base class of all AI implmentations
- StupidAI : Sample class of AI implementaion ***
- DijkstraAI : Sample class of AI implementaion, Demo use only, no source code.
- TerrainMap : A class representing the world, start and end points, and visited squares. ***
- PerlinTerrainGenerator : A class that generates random terrains using Perlin noise functions. *

Skim through the code

First play with the sample and read the code a little bit:

- test DijkstraAI
- read Main.java, see the possible command line arguments
- find getCost(final Point p1, final Point p2) function in TerrainMap.java
- read the two getCost() functions
- other code mentioned above

Skim through the code

First play with the sample and read the code a little bit:

- test DijkstraAI
- read Main.java, see the possible command line arguments
- find getCost(final Point p1, final Point p2) function in TerrainMap.java
- read the two getCost() functions
- other code mentioned above

Skim through the code

First play with the sample and read the code a little bit:

- test DijkstraAI
- read Main.java, see the possible command line arguments
- find getCost(final Point p1, final Point p2) function in TerrainMap.java
- read the two getCost() functions
- other code mentioned above

Skim through the code

First play with the sample and read the code a little bit:

- test DijkstraAI
- read Main.java, see the possible command line arguments
- find getCost(final Point p1, final Point p2) function in TerrainMap.java
- read the two getCost() functions
- other code mentioned above

Skim through the code

First play with the sample and read the code a little bit:

- test DijkstraAI
- read Main.java, see the possible command line arguments
- find getCost(final Point p1, final Point p2) function in TerrainMap.java
- read the two getCost() functions
- other code mentioned above

Part 1

Creating Heuristics for 4 cost-action combination each.

Part 2

- Implement the heuristic and A* algorithm, only for 8-option action and two cost functions.
- copy the StupidAI.java and rename it.
- create getHeuristic() as a member function of your implemented AI module.
- implement A* algorithm in createPath() function using your getHeuristic() function
- you might need to read TerrainMap.java for reference

Part 2

- Implement the heuristic and A* algorithm, only for 8-option action and two cost functions.
- copy the StupidAI.java and rename it.
- create getHeuristic() as a member function of your implemented AI module.
- implement A* algorithm in createPath() function using your getHeuristic() function
- you might need to read TerrainMap.java for reference

Part 2

- Implement the heuristic and A* algorithm, only for 8-option action and two cost functions.
- copy the StupidAI.java and rename it.
- create getHeuristic() as a member function of your implemented AI module.
- implement A* algorithm in createPath() function using your getHeuristic() function
- you might need to read TerrainMap.java for reference

Part 2

- Implement the heuristic and A* algorithm, only for 8-option action and two cost functions.
- copy the StupidAI.java and rename it.
- create getHeuristic() as a member function of your implemented AI module.
- implement A* algorithm in createPath() function using your getHeuristic() function
- you might need to read TerrainMap.java for reference

Part 2

- Implement the heuristic and A* algorithm, only for 8-option action and two cost functions.
- copy the StupidAI.java and rename it.
- create getHeuristic() as a member function of your implemented AI module.
- implement A* algorithm in createPath() function using your getHeuristic() function
- you might need to read TerrainMap.java for reference

Part 3

- Try out your code for 10 scenarios (2 cost functions * 5 random seeds)
- read Main.java to see how to input arguments (random seeds) (default map size is 500x500)
- record each command you use and the result (the cost of the shortest path and the number of nodes expanded)

Part 3

- Try out your code for 10 scenarios (2 cost functions * 5 random seeds)
- read Main.java to see how to input arguments (random seeds) (default map size is 500x500)
- record each command you use and the result (the cost of the shortest path and the number of nodes expanded)

Part 3

- Try out your code for 10 scenarios (2 cost functions * 5 random seeds)
- read Main.java to see how to input arguments (random seeds) (default map size is 500x500)
- record each command you use and the result (the cost of the shortest path and the number of nodes expanded)

Part 4

- Implemented a variant of standard A* and modify heuristic to cope with the much larger grid in MTAFT.XYZ
- consider using memory-bounded heuristic search
- don’t be so clever to memorize the path in your code :)

Part 4

- Implemented a variant of standard A* and modify heuristic to cope with the much larger grid in MTAFT.XYZ
- consider using memory-bounded heuristic search
- don’t be so clever to memorize the path in your code :)

Part 4

- Implemented a variant of standard A* and modify heuristic to cope with the much larger grid in MTAFT.XYZ
- consider using memory-bounded heuristic search
- don’t be so clever to memorize the path in your code :)

1 Heuristic Search Review

2 Code Infrastructure

3 Assignment Requirements & Grading Policy

Part 1

- CREATE heuristic, DOCUMENT the exact form of the heuristic and PROVE it is admissible
- 6 points for each cost-action combination

Part 1

- CREATE heuristic, DOCUMENT the exact form of the heuristic and PROVE it is admissible
- 6 points for each cost-action combination

Part 2

- Implement A* algorithm. 20 points
- Implement your AI Module for two cost functions. 5 points each
- You shall lose points with wrong filenames.

Part 2

- Implement A* algorithm. 20 points
- Implement your AI Module for two cost functions. 5 points each
- You shall lose points with wrong filenames.

Part 2

- Implement A* algorithm. 20 points
- Implement your AI Module for two cost functions. 5 points each
- You shall lose points with wrong filenames.

Part 3

- For two cost function and corresponding heuristic, experiment with seeds 1,2,3,4,5
- hand in the record for each case (command, cost of shortest path, number of nodes expanded)
- For each case:
  - 5 points*(shortest path cost)/(your path cost)
  - if your get 5 points above, you shall have bonus marks depends on rank.

Part 3

- For two cost function and corresponding heuristic, experiment with seeds 1,2,3,4,5
- hand in the record for each case (command, cost of shortest path, number of nodes expanded)
- For each case:
  - 5 points*(shortest path cost)/(your path cost)
  - if your get 5 points above, you shall have bonus marks depends on rank.

Part 3

- For two cost function and corresponding heuristic, experiment with seeds 1,2,3,4,5
- hand in the record for each case (command, cost of shortest path, number of nodes expanded)
- For each case:
  - 5 points*(shortest path cost)/(your path cost)
  - if your get 5 points above, you shall have bonus marks depends on rank.

Part 3

- For two cost function and corresponding heuristic, experiment with seeds 1,2,3,4,5
- hand in the record for each case (command, cost of shortest path, number of nodes expanded)
- For each case:
  - 5 points*(shortest path cost)/(your path cost)
  - if your get 5 points above, you shall have bonus marks depends on rank.

Part 3

- For two cost function and corresponding heuristic, experiment with seeds 1,2,3,4,5
- hand in the record for each case (command, cost of shortest path, number of nodes expanded)
- For each case:
  - 5 points*(shortest path cost)/(your path cost)
  - if your get 5 points above, you shall have bonus marks depends on rank.

Part 4

- Describe your modified A* algorithm and admissible heuristic clearly and concisely ? points
- hand in the right files. 10 points
- record and hand in the cost, number of nodes expanded, time for each cost function. 10 points
- if you have any doubts about the validity of your approach then first contact Prof.Davidson.

Part 4

- Describe your modified A* algorithm and admissible heuristic clearly and concisely ? points
- hand in the right files. 10 points
- record and hand in the cost, number of nodes expanded, time for each cost function. 10 points
- if you have any doubts about the validity of your approach then first contact Prof.Davidson.

Part 4

- Describe your modified A* algorithm and admissible heuristic clearly and concisely ? points
- hand in the right files. 10 points
- record and hand in the cost, number of nodes expanded, time for each cost function. 10 points
- if you have any doubts about the validity of your approach then first contact Prof.Davidson.

Part 4

- Describe your modified A* algorithm and admissible heuristic clearly and concisely ? points
- hand in the right files. 10 points
- record and hand in the cost, number of nodes expanded, time for each cost function. 10 points
- if you have any doubts about the validity of your approach then first contact Prof.Davidson.

Thank you!

- Good luck with coding!