## Notes

* 4-8 snakes on the board at once
* Each group will battle in 2 games
* Winner from each game moves forward to next round
* Groups announced at 5.30
* 4-8 teams will be final division
* Bounty snakes – defeat to win prize
* Code review corner - registration desk
* Food trucks in museum courtyard
* Do not bring food back to conference center
* Don’t be an asshole

Sneaky Snake Battlesnake Contestant

# Callbacks:

The server will POST to our server, and we will respond with one of these callbacks. All game moves should be done during /move.

## /start

HTTP POST /start will be called when a new game is started. Three attributes are included, a unique game\_id, the map height, and the map width. The response should include the snake’s body color, its head\_url, the snake’s name, and its taunt, and should have HTTP status code 200 OK.

Table 1 - /start Request Attributes

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** |  |
| **game\_id** | **UUID** |  |
| **height** | **integer** |  |
| **width** | **integer** |  |

Table 2 - /start Response Attributes

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** |  |
| **color** | **string** | **A valid css color. HSL, RGB, hex, or named color.** |
| **head\_url** | **URL** | ***optional* – URL of an image to use as the head of your snake.** |
| **name** | **string** | **Your snake’s name** |
| **taunt** | **string** | ***optional* – Message to display in the game client** |
|  |  |  |

Example request

{  
 "width": 20,  
 "height": 20,  
 "game\_id": "b1dadee8-a112-4e0e-afa2-2845cd1f21aa"  
}

Example response

{  
 "color": "#FF0000",  
 "head\_url": "http://placecage.com/c/100/100",  
 "name": "Cage Snake",  
 "taunt": "OH GOD NOT THE BEES"  
 }

## /move

HTTP POST /move – This callback is requested any time it is the client’s opportunity to move. This callback should be used for all game logic, and we only have 200ms to respond.

Table 1 - /move Request Attributes

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** |  |
| **food** | **Array<Point>** | **Array of all food currently on the board** |
| **game\_id** | **UUID** |  |
| **height** | **integer** |  |
| **width** | **integer** |  |
| **turn** | **integer** | **The current turn** |
| **snakes** | **Array<Snake>** | **Array of all living snakes in the game** |
| **you** | **UUID** | **A reference to your snake’s id, the snake object can be found in** snakes. |

Table 2 - /move Response Attributes

|  |  |  |
| --- | --- | --- |
| **Attribute** | **Type** |  |
| **move** | **string** | **“up” | “left” | “down” | “right”** |
| **taunt** | **string** | ***optional*** |

Example request

{

"you": "25229082-f0d7-4315-8c52-6b0ff23fb1fb",

"width": 2,

"turn": 0,

"snakes": [

{

"taunt": "git gud",

"name": "my-snake",

"id": "25229082-f0d7-4315-8c52-6b0ff23fb1fb",

"health\_points": 93,

"coords": [

[

0,

0

],

[

0,

0

],

[

0,

0

]

]

},

{

"taunt": "gotta go fast",

"name": "other-snake",

"id": "0fd33b05-37dd-419e-b44f-af9936a0a00c",

"health\_points": 50,

"coords": [

[

1,

0

],

[

1,

0

],

[

1,

0

]

]

}

],

"height": 2,

"game\_id": "870d6d79-93bf-4941-8d9e-944bee131167",

"food": [

[

1,

1

]

],

"dead\_snakes": [

{

"taunt": "gotta go fast",

"name": "other-snake",

"id": "c4e48602-197e-40b2-80af-8f89ba005ee9",

"health\_points": 50,

"coords": [

[

5,

0

],

[

5,

0

],

[

5,

0

]

]

}

]

}

Example response

{

"move": "up",

"taunt": "gotta go fast"

}

OUTPUT:

# Operation:

## Features

What can Sneaky Snake do?

* Move up / down / left / right
* Avoid bigger snakes
  + Weigh cells around big snakes heads heavily
* Ignore smaller snakes
  + Weigh cells around smaller snakes heads lightly
* Potentially destroy smaller snakes
  + Reward paths that intersect smaller snakes heads path
* Dumb simulator of next potential move for opponent snakes
  + Possibly one or two space radius around snakes, looking for food or heads of smaller snakes
  + Based on the last move they made (ie a snake can’t move backwards towards food)
  + Potentially pathfinding their future path
* Memory of old snake movements
  + Ex: if they move immediately towards food, make a target of destroying them or eating food first
* Head towards food if closest snake to food, otherwise evaluate if opposing snake is traveling towards or away, and if bigger (both including food and not)
* Remember our body length and position to avoid collisions
* Remember our health, weigh decisions based on how much is left
  + Avoid aggressive decisions if health is low
  + Potentially go into “open-area” health mechanism, where we circle one area waiting for food
* Weigh snake locations, snake potential moves, and food locations to find optimal path
  + Do in layers
    - Weigh Snake locations
    - Weigh Simulated snake locations
    - Weigh food items
* Create a priority queue of goals we’d like to go to (foods & smaller snake head locations)
* Simulate what the board would look like if we took our path. If there is no clear path to our tail, then we skip that goal
* Snake tail avoiding
* After 180ms if we don’t have a decisions, go to simplified decision algorithm.
  + Instead, first do a simplified algorithm, then do Djikstras with food, then with snakes, etc. We refine the choice of movement with every next algorithm
* Timer interrupt to submit best next step at given time
* Using python-igraph for Djikstra’s algorithm
* Person controlled interface
* Graphical output for pathfinding weights
* Output each best decision from each algorithm

## Functions

### getSnakes()

Creates a list of snake objects, with their lengths and full body locations.

### getFoods()

Creates a list of food objects and returns a list with their distances from our snake’s head.