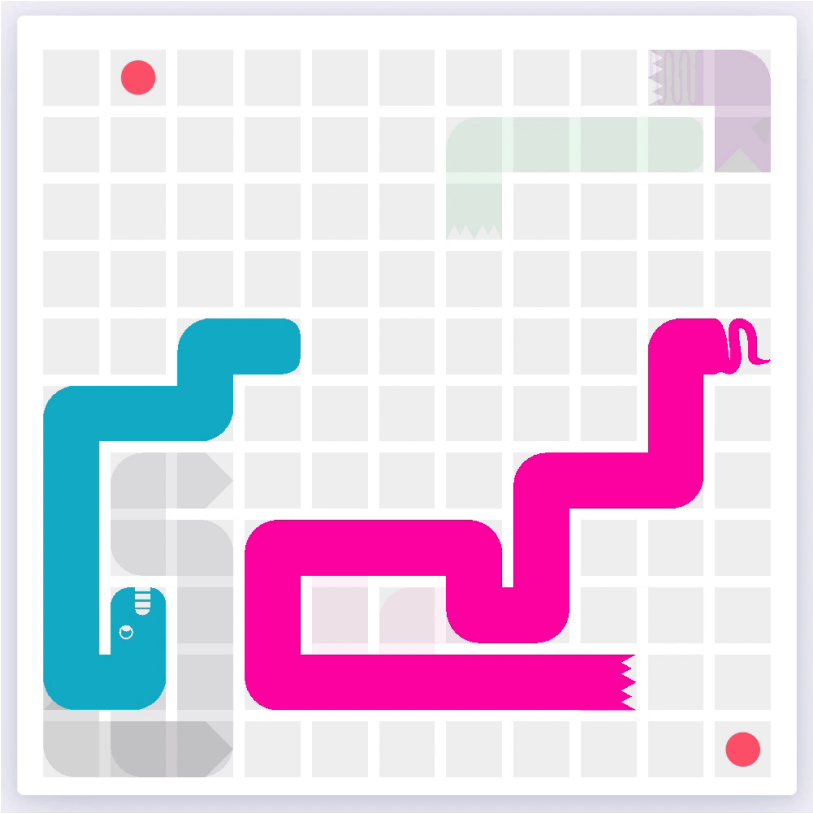
Lab 2 – Battlesnake Competition: Minimax Algorithm
100 points

Overview:
In this programming assignment, we will implement the Minimax adversarial search algorithm to control an AI to play a game of Battlesnake. The programming assignment should be completed in groups of 3-5 students (for a total of 16 teams). To complete the lab you must submit via Canvas a zip file containing your source code and a write-up describing your methods and results. Since this is a group assignment, only one group member needs to submit the final deliverable on Canvas.

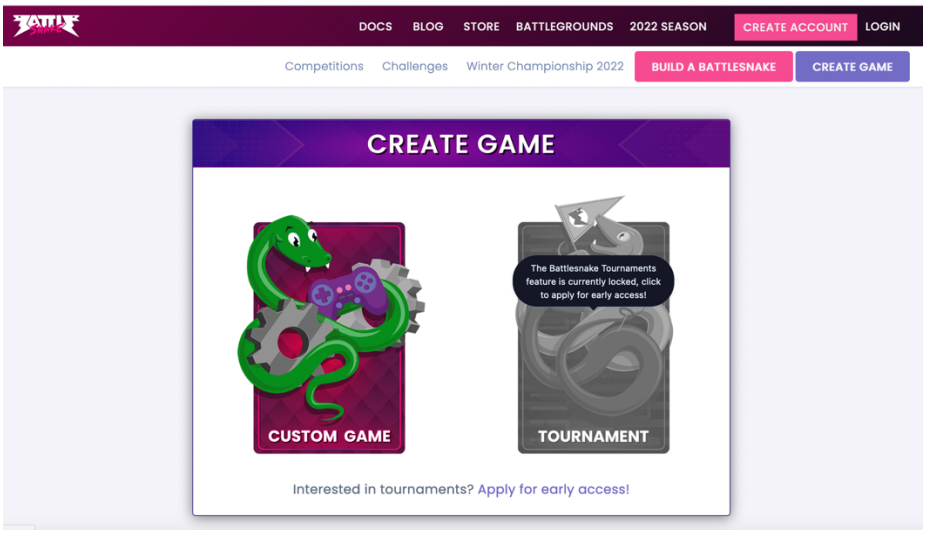


Part I: Tutorial

1. In the Battlesnake Competition, teams will build and deploy a Python AI script that implements the Battlesnake API (https://docs.battlesnake.com/api). When a game is created, the game engine will make HTTP requests to your Battlesnake server, sending game board information and asking for your next move. Your Battlesnake's behavior is determined by how you program it to act at each game state.
2. The game rules are as follows (refer https://docs.battlesnake.com/rules for the complete rules):
    - **Winning the Game:** Be the last snake alive on the game board
    - **Boundaries:** The game is played on a rectangular game board of variable size. Any snake attempting to move outside the boundaries will be eliminated from the game.
    - **Collisions:**
        - **Self-collisions:** If a snake collides with its own body, it will be eliminated

    - **Body collisions:** If a snake collides with the body of another Battlesnake, it will be eliminated
    - **Head-to-head collisions:** When two snakes meet head-to-head, the larger snake will survive and the shorter will be eliminated. If both are the same size, then the outcome will be a draw.
  - **Food:** At the beginning of each game some amount of food will be placed around the board. On each subsequent turn additional food may be added randomly. A snake that enters the same square as a piece of food will immediately consume it, filling its health to maximum and growing its length by one.
  - **Health:** Each snake starts with a health of 100 points. At the end of every turn after making a move, a snake's health will decrease by 1 point. If a snake's health reaches 0, it will be eliminated. The only way to restore health is by consuming food.
3. Read through the official guide (https://docs.battlesnake.com/guides/customizations) and quickstart (https://docs.battlesnake.com/quickstart) on Battlesnake's website to get familiarized with the game API, rules, and terminology. Note that only the **Standard** game mode will be relevant for this assignment.

Part II: Playing on the Official Server

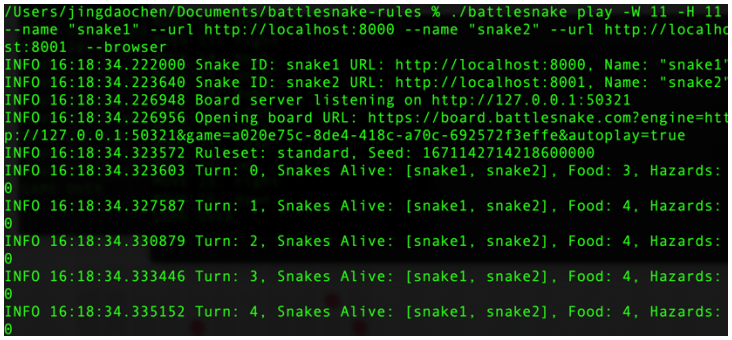


First, we will run the starter Python code using cloud hosting and play on the official server.

1. Download the Battlesnake Python Starter Project from Github. (https://github.com/BattlesnakeOfficial/starter-snake-python)
2. Identify a cloud provider that can host your Battlesnake code. https://codesandbox.io/ is the best option since it offers free online IDE and web hosting.
    a. replit.com is the recommended option on the Battlesnake website but replit.com no longer offers free hosting as of 01/01/2024.
    b. Other cloud hosting options can be found here https://docs.battlesnake.com/guides/hosting-suggestions
3. Create an account on the cloud provider and upload your Python starter project there. After running your Battlesnake within the remote host, your Battlesnake server will start and you should see the live output from your Battlesnake server together with a public URL.

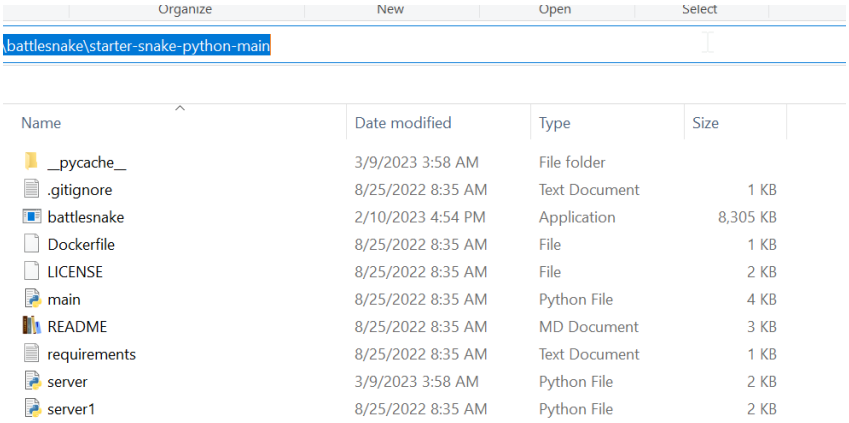
4. Create an account on https://play.battlesnake.com/. Go to "Build a Battlesnake" and paste in the Battlesnake server URL you have obtained from your cloud hosting site.
5. Go to "Create Game" and add your starter snake as one of the players. You may also add in one of the default AI snakes as your opponent. After clicking on "Start Game", you will be able to view the Battlesnake game on your web browser.
6. The starter Battlesnake only moves in a random direction. This means that for this first game, your Battlesnake is most likely going to collide with a wall or turn back in on its own body. To help your Battlesnake survive, you will need to implement the following functionality:
    (a) Avoid colliding with walls
    (b) Avoid colliding with yourself
    (c) Avoid colliding with other snakes
    (d) Try to move towards food

Part III: Playing on localhost (optional)



To facilitate experimentation and testing, it may be more convenient to run the game engine and code for competing Battlesnakes on the same computer. We will run two competing AI Battlesnakes in Python as well as the official Battlesnake rules engine.

1. Go to: https://github.com/BattlesnakeOfficial/starter-snake-python to obtain the starter Python code. Download and extract the zip file to a local folder.
2. Open the command line (terminal) in the target folder. In Windows, this can be easily done by typing *cmd* in the address bar https://stackoverflow.com/a/40146208 .
3. Install dependencies using pip: `pip install -r requirements.txt`
4. Go to https://github.com/BattlesnakeOfficial/rules/releases and select the compiled *battlesnake* executable that is suitable for your operating system. Download and extract the zip file to the target folder.
5. (optional) To compile the Battlesnake engine yourself, you will have to install the latest version of Go (https://go.dev/dl/). Then, download and compile the Go command line tool for the official Battlesnake rules and game logic from https://github.com/BattlesnakeOfficial/rules.
6. The target folder should look similar to the following:

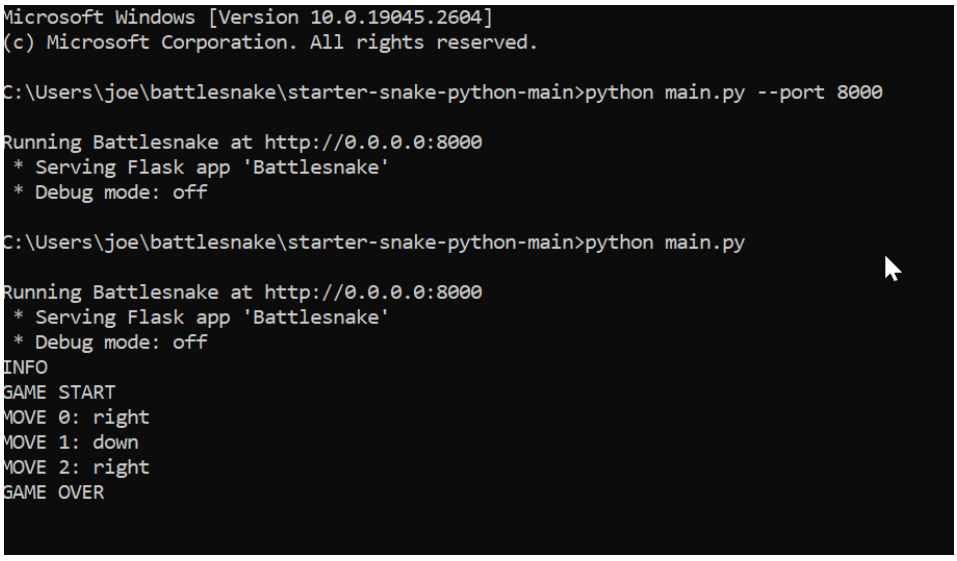


7. Replace the default starter Python code in the target folder with *main.py*, *simple.py* and *server.py* which can be downloaded from the HW 2 page in Canvas.
8. Open 2 additional terminal windows to have 3 terminal windows in total. Run the following commands to create 2 localhost servers with different port numbers. In the first terminal, run the *main.py* script which should be the main body of your minimax code. In the second terminal, run the *simple.py* script which provides a simple AI opponent.

```
python main.py --port 8000
python simple.py --port 8001
```

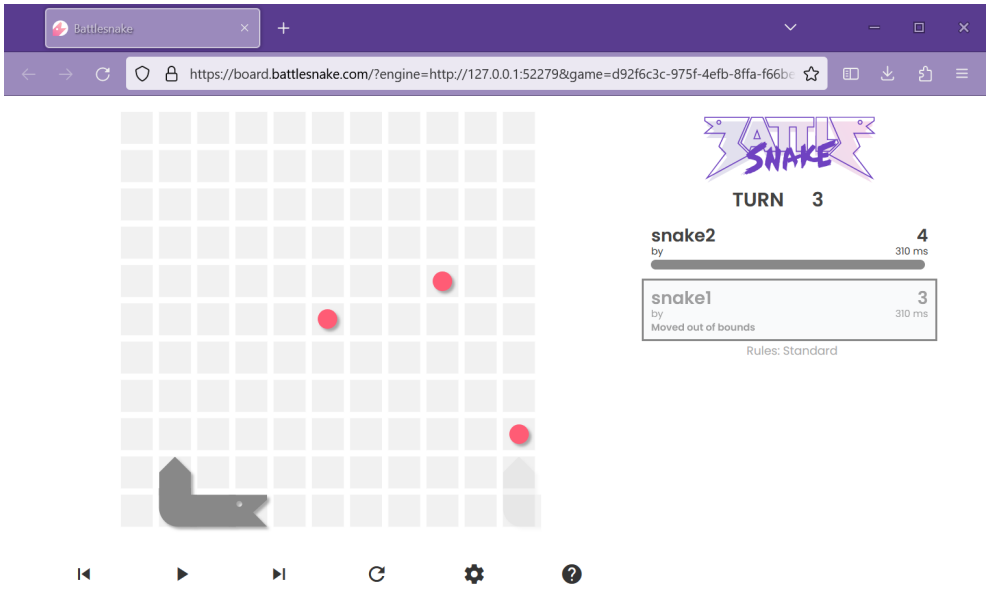
9. In the third terminal, run the following command to initiate a Battlesnake game. The game results will be animated through your web browser.

```
./battlesnake play -W 11 -H 11 --name "snake1" --url
http://127.0.0.1:8000 --name "snake2" --url http://127.0.0.1:8001  --
browser
```



```
Microsoft Windows [Version 10.0.19045.2604]
(c) Microsoft Corporation. All rights reserved.

C:\Users\joe\battlesnake\starter-snake-python-main>python main.py --port 8001

Running Battlesnake at http://0.0.0.0:8001
 * Serving Flask app 'Battlesnake'
 * Debug mode: off
INFO
GAME START
MOVE 0: down
MOVE 1: right
MOVE 2: right
MOVE 3: up
GAME OVER
```

```
Microsoft Windows [Version 10.0.19045.2604]
(c) Microsoft Corporation. All rights reserved.

C:\Users\joe\battlesnake\starter-snake-python-main>battlesnake play -W 11 -H 11 --name "snake1" --url http://localhost:8000 --n
ame "snake2" --url http://localhost:8001 --browser
INFO 04:15:03.481360 Snake ID: df7a7fb7-7bbf-4560-a696-9f644a92ac71 URL: http://localhost:8000, Name: "snake1"
INFO 04:15:03.796886 Snake ID: 36b052da-2226-483f-9370-ac829917d274 URL: http://localhost:8001, Name: "snake2"
INFO 04:15:04.419859 Board server listening on http://127.0.0.1:52279
INFO 04:15:04.419859 Opening board URL: https://board.battlesnake.com?engine=http://127.0.0.1:52279&game=d92f6c3c-975f-4efb-8ff
a-f66be26e93e1&autoplay=true
INFO 04:15:04.611033 Ruleset: standard, Seed: 1678356903144054600
INFO 04:15:04.611116 Turn: 0, Snakes Alive: [snake1, snake2], Food: 3, Hazards: 0
INFO 04:15:04.921129 Turn: 1, Snakes Alive: [snake1, snake2], Food: 3, Hazards: 0
INFO 04:15:05.231868 Turn: 2, Snakes Alive: [snake1, snake2], Food: 3, Hazards: 0
INFO 04:15:05.543412 Turn: 3, Snakes Alive: [snake2], Food: 3, Hazards: 0
INFO 04:15:06.475858 Game completed after 4 turns. snake2 was the winner.

C:\Users\joe\battlesnake\starter-snake-python-main>
```



Part IV: Minimax (80 pts)

To write a proper AI program that can win in a Battlesnake competition, we will implement the Minimax adversarial search algorithm. The Minimax algorithm will search for the best possible move given that

your agent will attempt to maximize its score while the opponent will attempt to minimize your agent's score.

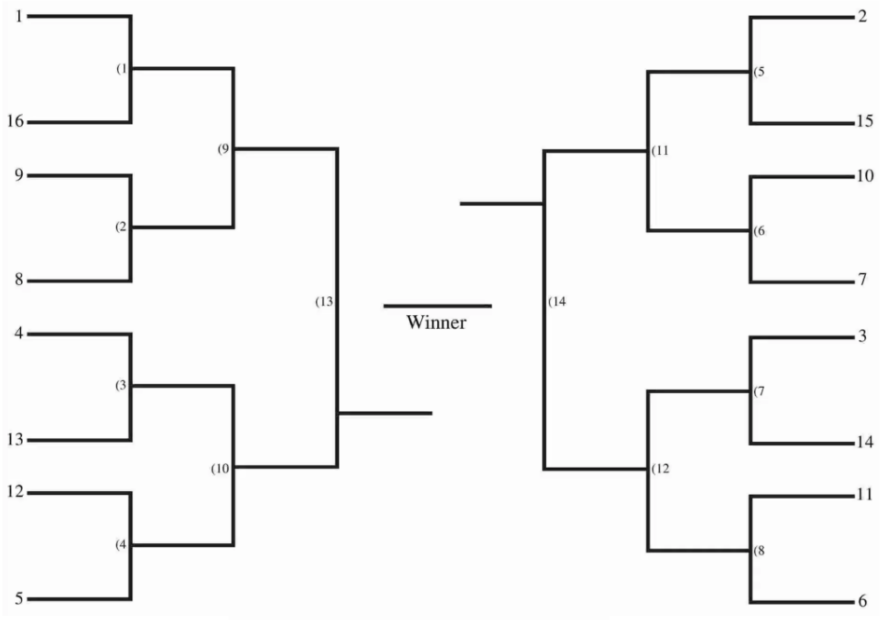
1. Implement the Minimax algorithm according to the pseudocode below.
2. Implement a heuristic function that appropriately scores the current state:
    a. The score should be a large positive number (+∞) if the opponent's snake dies
    b. The score should be a large negative number (-∞) if your snake dies
    c. The score can also be a function of various attributes of the game state such as the current number of move options available, length of each snake, number of squares controlled, snake health, distance to nearest food etc.
3. You may test the effectiveness of your Minimax Battlesnake by pitting it against the given simple AI opponent (simple.py) or any of the opponents on the official Battlesnake website.

```
function minimax(gameState, depth, maximizingPlayer ):
    if depth = 0 or gameState is terminal:
        return the heuristic value of current state
    if maximizingPlayer then
        value := −∞
        bestMove := None
        for each move_option do
            newState = gameState.apply(move_option)
            value, bestMove := max(value, minimax(newState, depth−1, False))
        return (value, best_move)
    else # minimizing player
        value := +∞
        bestMove := None
        for each child of node do
            newState = gameState.apply(move_option)
            value, bestMove := min(value, minimax(newState, depth-1, True))
        return (value, best_move)
```

Part V: Competition

1. The Battlesnake competition will be held in class (date will be specified on Canvas). The competition date will be right after the submission deadline for this assignment.
2. A trial run of the Battlesnake competition will be conducted one week before the competition date. Teams will be randomly paired against one another. This will be an opportunity for teams to fix remaining bugs and strategize for the actual competition.
3. The final competition will be in a Single Elimination format with 1v1 battles. Each battle consists of multiple games with different initial seeds; the first snake to win 2 games advances to the next round (ignoring draws). The Round-of-16 and Quarter Finals will be conducted offline; the Semi-Finals and Final will be conducted live in class.
4. Teams that submit non-working code or code that violates the rules or fair play will be automatically disqualified. Their opponent will receive a bye to advance to the next round.
5. Winning teams will receive extra credit points according to the following bracket:
    a. Winner: +50 points
    b. Runner-up: +40 points
    c. Semi-finalist: +30 points
    d. Quarter-finalist: +20 points

6. Consolation prize: Teams that are eliminated in the Round-of-16 but did implement any improvement to the minimax algorithm (e.g. shortest path, alpha beta pruning, probabilistic reasoning) will qualify for +10 extra credits points.



Part VI: Report (20 pts)

Each group is required to submit a report in PDF format on Canvas containing the following items:

- List of group members
- Contribution of each group member (e.g. technical discussion, experimentation, coding, parameter tuning, simulations, evaluation, testing, report writing)
- Strategies that were attempted (both strategies that worked and did not work)
- External tools / libraries / resources that were used
- Explanation of final algorithm implemented, with accompanying diagrams where applicable
- Strengths and weaknesses of final algorithm implemented
- Challenges encountered and how to overcome them

Tips and Tricks

1. When debugging your Battlesnake code against an AI opponent, it may be helpful to seed the pseudo-random number generator so that the results are reproducible (using the `--seed` command line argument). For example,

```
./battlesnake play -W 11 -H 11 --url http://127.0.0.1:8000 --seed 0
```

guarantees that the same board will be generated each time whereas

```
python simple.py --port 8000 –seed 0
```

guarantees that the simple AI opponent will execute the same moves each time given a certain board.

2. Go through the following sites to get more specific details about how the Battlesnake game logic is implemented:

- https://docs.battlesnake.com/guides/playing/rules
- https://github.com/BattlesnakeOfficial/rules/blob/main/standard.go

3. A key part of the Minimax algorithm lies in creating a new state dictionary that represents the "future" state of the game board after a specific snake has made a move, thereby updating the snake position and snake health in the state dictionary. The "future" state needs to be a "deep" copy of the "current" state so that each node in the minimax game tree essentially has its own copy of the game state. Creating a deep copy of the game state every iteration can be computationally expensive so you may experiment with ways to make this more efficient (e.g. modify only a single state at the time but backtrack / reverse those modifications when moving back up the game tree).

4. Even though the Minimax algorithm models the game as two players taking turns alternatingly, in reality, the Battlesnake game engine executes moves for both snakes simultaneously. This means that your code will have to carefully consider edge cases such as when a snake moves to an opponent's tail or to its own tail or when both snakes move to the same location.

5. Consider using some of the AI techniques we have learned in this class such as state-space search, local search, heuristics, alpha-beta pruning, Monte-Carlo tree search etc. to optimize the performance of your Battlesnake. Make sure to use efficient data structures and optimized algorithms to minimize computation time.

6. Other helpful resources:

- Official Battlesnake tools and documentation
  - https://github.com/BattlesnakeOfficial/
- A curated list of open-source Battlesnake projects
  - https://github.com/xtagon/awesome-battlesnake
- Paper on Battlesnake algorithms
  - https://dl.gi.de/bitstream/handle/20.500.12116/29001/SKILL2019-09.pdf

Frequently-asked Questions

Q: Can I refer to or make use of example code from the Internet?
A: Yes, you may use any Python library functions, Github, StackOverflow, or open-source community-developed Battlesnake projects that are available on the Internet (e.g. functions for geometry calculations, game update logic, path planning). However, this may only be in the form of function imports from your main Python script and all the code in *main.py* (i.e. the implementation of the Minimax algorithm) has to be your own work. You must cite and acknowledge all external resources that you use in your written PDF report that you submit on Canvas by explaining how you used those resources and providing links to the original sites.

Q: What programming languages / frameworks are allowed?
A: To make the execution time fair for all teams, the final Battlesnake code submitted should be written in Python, although your code for simulation and benchmarking can be written in other languages. The code should be written so that it can be executed on a basic laptop computer. During the competition,

submitted code from all teams will be executed on the instructor's laptop. Standard Python numerical computing libraries such as NumPy and SciPy are allowed but Python libraries that exploit hardware/GPU acceleration/parallel computing such as Tensorflow and PyTorch are not allowed. Please check with the instructor if you are unsure whether a specific library is allowed.

Q: What are the ruleset and game settings that will be used for the Battlesnake competition?
A: The Battlesnake competition will use the Standard ruleset with a 11 x 11 board and a 500ms timeout.