

Project 2

Please read this file carefully to understand the **framework** and **grading policy**.

Also, the result of your project should include your **source code** and **the report**!

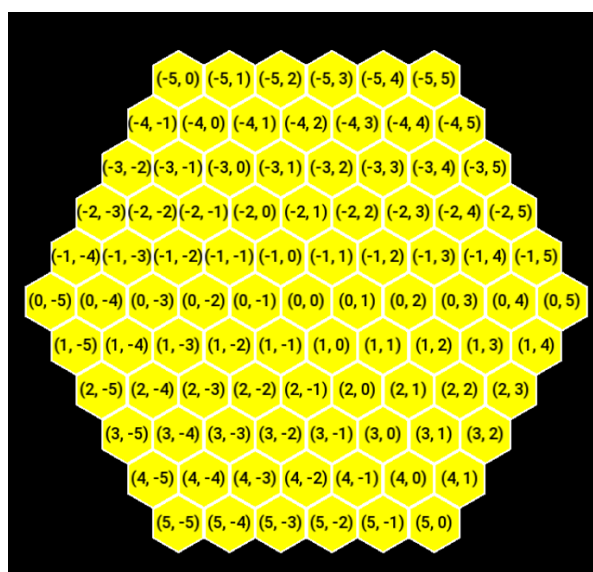
Framework

We provide **C++** and **Python** framework for Project 2; you can choose the one that you're more familiar with to implement your Minimax and MCTS code

*(Each of your AI's turn has only **time limit of 30 seconds** !!!)*

Coordinate

Both C++ and Python framework use the same coordinate system. Its general form is **(row, col)**. What you should notice is that row and col is the abstract coordinate for hexagon, but not the actual location in the window.



C++ Framework

This section includes the **explanation** of C++ framework. The thing that you should know is how hexagon information is stored in the data structure.

Also, I've already provided a random function version of **AI_move** as an **example** for you to understand how the framework works. (*In move.cpp*)

HexInfo

In `hex_info.h`, there's a structure called `HexInfo`, which stored the actual location and its label for each hexagon.

```
7
8  struct HexInfo {
9      float x;
10     float y;
11     int label;
12 };
13
```

- x: the actual coordinate of x in the window
- y: the actual coordinate of y in the window
- label: the representation of the number (The label will be **1~5** if not placed yet, **0** if placed by black, and **255** if placed by white.)

Then, I map the row and col of each hexagon to its actual x, y coordinate and label, using map function. So, the information of hexagon is stored in this form:

`<(row, col), {x, y, label}>`

hexagon_board

Also, there's a global map container named `hexagon_board`, which stored the state of all hexagons on the board.

```
14  extern std::map<std::pair<int, int>, HexInfo> hexagon_board;
```

(e.g. if one of the hexagon is row = 0, col = 0, x = 325.0, y = 325.0, label = 1, then it will be stored by this way:

```
hexagon = std::make_pair(0, 0);  
hexagon_board[hexagon] = {325, 325, 1};)
```

(You can use a for loop to find the desire hexagons that you want to place. I've show how to do it in **AI_move** function written in move.cpp.)

Yor Job: Finishing the framework

Please finish the following two functions in C++ framework:

(These two functions must be written in `move.cpp` file ! They are located at `C++_framework_<your-OS>/move.`)

a) yourFirstAIMove

This function decides the first move in the very beginning of the game, if your AI is Black player.

Because it only requires putting one place on the board, it only needs to return `pair<int, int>`, where the first thing and the second in the pair is **row** and **col** of the hexagon that is labeled '2'.

b) yourAIMove

This function should decide the places to put in your AI's turn, no matter which color of the player you are.

Because it always needs to decide more than one place to put on the board, it must return `vector<pair<int, int> >`, where each element in it is a pair that stores **row** and **col**. And every element in the vector must have same label!

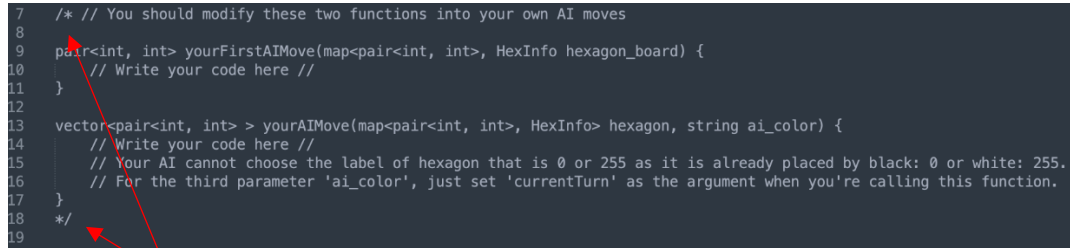
e.g. If you decide to choose hexagons labeled '5', then you must return **5 (row, col)** that the hexagon it associated with is labeled '5' to put on the board, such as `vector<(-4, 5), (-3, 5), (4,1), (-1, -4), (-5, 2)>`.

(You should notice that yourAIMove **cannot** return the (row, col) of the hexagons that have already been placed, and **cannot** return the (row, col) of the hexagons that have different labels either. The mistake preventing mechanism will trigger if these situations happen.)

```

7  /* // You should modify these two functions into your own AI moves
8
9  pair<int, int> yourFirstAIMove(map<pair<int, int>, HexInfo hexagon_board) {
10     // Write your code here //
11 }
12
13 vector<pair<int, int> > yourAIMove(map<pair<int, int>, HexInfo> hexagon, string ai_color) {
14     // Write your code here //
15     // Your AI cannot choose the label of hexagon that is 0 or 255 as it is already placed by black: 0 or white: 255.
16     // For the third parameter 'ai_color', just set 'currentTurn' as the argument when you're calling this function.
17 }
18 */
19

```



Please delete ‘/*’ and ‘*/’ when you finish your code.

Then, go to `player.cpp`, which is located at `C++_framework_<your-OS>/players`.

Type `Ctrl + f` for Windows/ `command + f` for MacOS , and type `FirstAI_move` and `AI_move` to search all of these two functions.

Replace all the `FirstAI_move` with `yourFirstAIMove`; replace all the `AI_move` with `yourAIMove`.

Also, remember to add the `prototype` of `yourFirstAIMove` and `yourAIMove` into `move.h` file, which is also located at `C++_framework_<your-OS>/move`.

*(By the way, you can write any of the **<helper functions>** or **<helper .cpp file with .h file>**, just make sure the code can work and write it in the report to let me know where it is.)*

Python Framework

This documentation provides an overview of the Python framework for Project 2, which supports the development of AI algorithms using the Minimax and Monte Carlo Tree Search (MCTS) techniques. This framework is set up for a game played on a hexagonal grid, with specific focus on the logical management of game state rather than graphical representations.

Framework Overview

The Python framework is designed to facilitate AI gameplay within a hexagonal coordinate system, abstracting the actual window coordinates to focus on the

game logic. Each AI move must be computed within a 30-second time limit to ensure game flow and challenge.

Coordinate System

- **Description**: The game uses a logical coordinate system for the hexagonal grid, expressed as (row, col).
- **Purpose**: Facilitates the placement and movement strategies by abstracting game logic from graphical details.

Game Board Initialization

- **Global Variables**:

- `hexagon_board``: A dictionary mapping (row, col) tuples to dictionaries containing hex properties such as coordinates and state.
- `selected_counts``: Tracks how many hexes have been selected by each label.
- `initial_counts``: Stores initial counts for each hex label at the start of the game.

Key Functions

`draw_hexagon` (surface, x, y, size, border_color, fill_color, number=None)

- **Parameters**:

- `surface``: The pygame surface to draw on.
 - `x`, y``: The center coordinates for the hexagon.
 - `size``: The radius of the hexagon.
 - `border_color`, fill_color``: Colors for the hexagon's border and fill.
 - `number``: Optional label or identifier for the hexagon.
- **Purpose**: Draws a single hexagon on the specified surface with provided visual

characteristics.

``draw_hex_shape_grid` (surface, center_row, center_col, size)``

- **Purpose:** Sets up the initial board layout by drawing hexagons in a grid pattern.
- **Details:** Uses the ``draw_hexagon`` function to place hexagons based on a centered grid system.

``process_selections_by_round` (x, y, current_round)``

- **Purpose:** Handles user interactions for selecting hexagons during the game.
- **Returns:** A list of hexagons that have been selected based on user input.

``select_hexes_by_random` (hexes_by_label, current_round)``

- **Description:** Implements a basic AI by randomly selecting hexagons based on their availability and labeling.
- **Usage:** This function is a placeholder for more complex AI implementations and serves as an example of how hexagons can be selected.

Implementing Your AI

To successfully integrate and execute your AI strategies using either Minimax or Monte Carlo Tree Search (MCTS) algorithms within our Python game framework, follow these steps:

1. Modify AI Selection Function:

- Update the ``select_hexes_by_random`` function in your Python script. Replace the random selection logic with your own AI logic, applying Minimax or MCTS algorithms to make strategic decisions.

```

302 def select_hexes_by_random(hexes_by_label, current_round):
303     """Selects hexes randomly based on the current round and label availability."""
304     selected_hexes = []
305     if current_round == 1:
306         # 第一回合特别处理：只选择一个标签为2的六边形
307         if 2 in hexes_by_label and any(not hex_info['selected'] for _, hex_info in hexes_by_label[2]):
308             available_hexes = [(pos, hex_info) for pos, hex_info in hexes_by_label[2] if not hex_info['selected']]
309             if available_hexes:
310                 selected_hexes.append(random.choice(available_hexes))
311         else:
312             # Calculate remaining unselected hexes for each label and choose only those labels with remaining hexes
313             available_labels = {
314                 label: hexes
315                 for label, hexes in hexes_by_label.items()
316                 if any(not hex_info['selected'] for _, hex_info in hexes)
317             }
318             if available_labels:
319                 selected_label = random.choice(list(available_labels.keys()))
320                 available_hexes = [(pos, hex_info) for pos, hex_info in available_labels[selected_label] if not hex_info['selected']]
321                 n = selected_label # 根据六边形的标签来决定需要选择的六边形数量
322
323                 # Randomly select n hexes, select all remaining hexes if fewer than n are available
324                 if len(available_hexes) > n:
325                     selected_hexes.extend(random.sample(available_hexes, n))
326                 else:
327                     selected_hexes.extend(available_hexes)
328
329     return selected_hexes
330

```

```

395     # Handle random selections for players set as "Random"
396     if not turn_ended and ((current_turn == 'black' and black_player_type == "random") or \
397                            (current_turn == 'white' and white_player_type == "random")):
398         # Filter unbooked hexes from the hexagon_board and categorize them by label
399         hexes_by_label = {}
400         for pos, info in hexagon_board.items():
401             if not info.get('booked', False):
402                 label = info['label']
403                 if label in hexes_by_label:
404                     hexes_by_label[label].append((pos, info))
405                 else:
406                     hexes_by_label[label] = [(pos, info)]
407
408         # Randomly select a specified number of hexes from those filtered by label
409         selected_hexes = select_hexes_by_random(hexes_by_label, current_round)
410
411         # Process the selected hexes
412         for pos, hex_info in selected_hexes:
413             hex_info['selected'] = True
414             hex_info['booked'] = True
415             fill_color = BLACK if current_turn == 'black' else WHITE
416             draw_hexagon(screen, hex_info['x'], hex_info['y'], HEX_SIZE, (128, 128, 128), fill_color, hex_info['label'])
417

```

2. Enforce Time Constraints:

- Your AI must complete its decision-making process within a 30-second limit per move. Ensure your implementation includes a timing mechanism to track the decision time. If your AI cannot determine the optimal moves within the 30-second window, it should default to a random selection to comply with the time constraints. This ensures the game progresses smoothly without delays.

3. Interaction with Game State:

- Utilize the `hexagon_board` dictionary for accessing and modifying the game state. This structure is crucial for your AI to evaluate the current status of the game and make informed decisions. The `hexagon_board` contains all the necessary information about each hexagon's position, state, and label, facilitating strategic gameplay.

By following these guidelines, your AI will be well-integrated into the gaming

framework, capable of competing under the specified rules and time constraints.

Testing and Validation

Thoroughly test your AI under various game scenarios to ensure it adheres to the rules and performs within the expected time constraints. Utilize the logging or printing of game states to debug and refine your strategy.

Summary

This framework provides a solid base for developing hexagon grid-based AI strategies, focusing on game logic and efficient processing. By following the structure and guidelines provided, developers can implement effective AI solutions tailored to the complexities of hexagonal grid gameplay.

Grading Policy [Total 120 pts]

The following is how we score your project in detail:

(a) Source Code [40 pts]

You need to provide both source code of **Minimax algorithm** and **MCTS** so we can check if it is work or not and use it to compete with your classmates.

- i. Minimax searching -with alpha-beta pruning [20 pts]
- ii. MCTS (Monte Carlo Tree Search) [20 pts]

(b) Report [50 pts]

In your report, you need to answer the following questions:

- i. [30 pts total] Explain how you implement Minimax algorithm [15 pts] and MCTS [15 pts] for this game in detail?
- ii. [10 pts total] Explain the most difficult thing you faced when implementing the code. Why? [5 pts] And how you solved it? [5 pts]
- iii. [10 pts total] Compare Minimax algorithm and MCTS, what is the difference between them? [5 pts] Which one do you think is better for **“Strands”**? [5pts]
(We’ll take the one you think is better for **Strands** to compete with your

classmates. For example, if you think your minimax algorithm is better, we'll use it to compete with your opponents)

(c) The ranking in the competition [30 pts]

The competition for the game will be held after the deadline of project 2 (Maybe after final exam). The detail of the contest will be announced soon.

You will need to attend the competition to earn this part of the score!