CS101 2024 Fall HW4 Description

Yutnori

Total: 100 Points

Out 25 Nov 2024 00:00
Due 02 Dec 2024 23:59

If you have any questions regarding Homework 4, use the ***Homework 4 Q&A*** board at Elice.

(Korean) 학습도우미 > 게시판 > Homework 4 Q&A
(English) Help Area > Forums > Homework 4 Q&A

Please read the homework description and follow the instructions carefully. Please be aware that this homework is **an individual task**; you can discuss the problem with your friends, but you MUST NOT implement your ideas together. **You will fail the entire course (that is, your CS101 grade will be an F) if you are found to be involved in any attempts of plagiarism, including using AI-sourced code (e.g., ChatGPT).**

Overview

In this homework, you will create a mini *Yutnori* game. *Yutnori* is a traditional Korean board game that is played by two teams or players. Here is how it works.

Game Components

1. **Yut Sticks**: The game uses four wooden sticks called Yut. Each stick has two sides, the marked side and the unmarked side. The marked side has “X” symbols, and the unmarked side has no symbols. The way they land determines the player’s move.

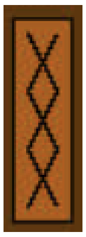


Figure 1a. Marked Side

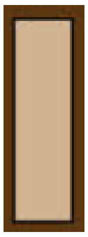


Figure 1b. Unmarked Side

2. **Board**: The game board consists of a large square around the outside and two diagonal paths intersecting in the center.

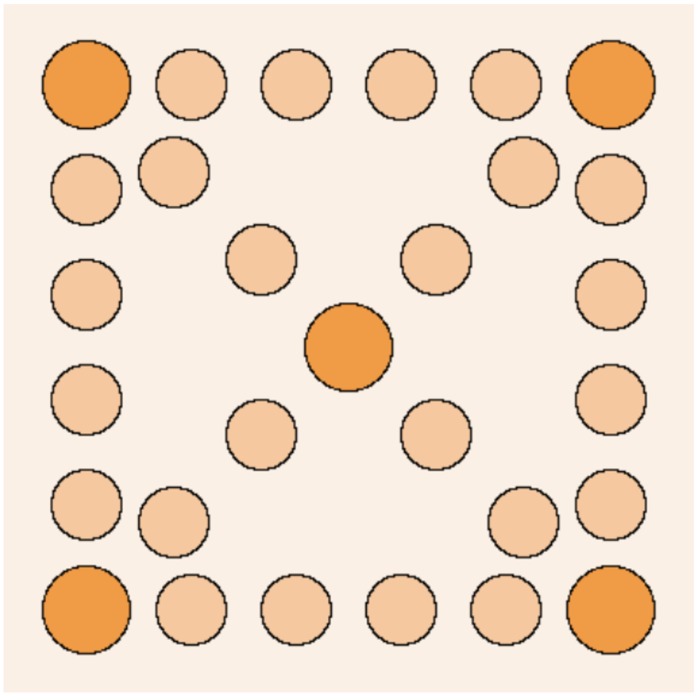


Figure 2. Game Board

3. **Pieces**: Each team or player has a set of markers. In this homework, we assume that there are two players with each having two pieces.

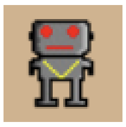


Figure 3a. Player 0's Piece

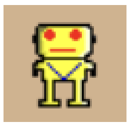


Figure 3b. Player 1's Piece

How to Play

1. **Throwing the Yut Sticks**: At the beginning of a turn, a player tosses the four sticks. How the sticks land (marked side or tail vs. unmarked side or head) determines the number of steps a player can take to move a piece.

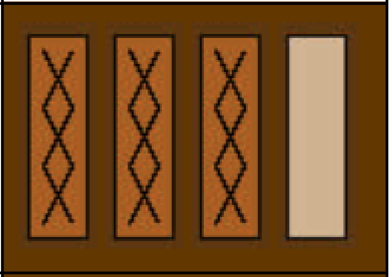
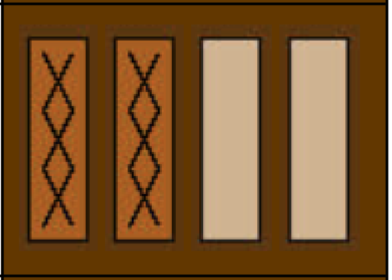
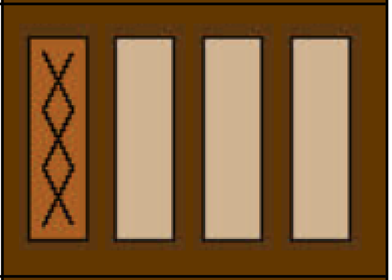
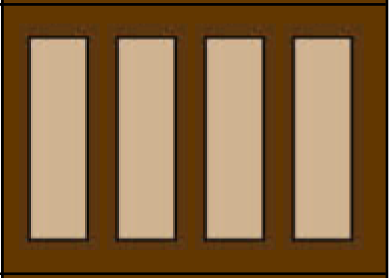
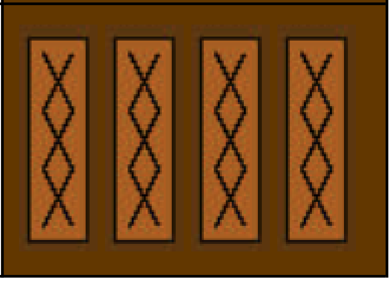
Name	Result	Move	Figure
Do	3 Marked Sides 1 Unmarked Side	Move 1 Step	
Gae	2 Marked Sides 2 Unmarked Sides	Move 2 Steps	
Geol	1 Marked Side 3 Unmarked Sides	Move 3 Steps	
Yut	0 Marked Side 4 Unmarked Sides	Move 4 Steps	
Mo	4 Marked Sides 0 Unmarked Side	Move 5 Steps	

Table 1. Cases of Yut sticks throwing

2. **Moving Pieces**: After each throw, a player moves one of their pieces depending on how the sticks landed (Do, Gae, Geol, Yut, or Mo). Players can enter a new piece onto the board or move an existing one.

3. **Choosing Paths**: The board has different routes, including the shortcuts along the diagonals. The shortcut can only be taken if the piece lands on the four dark points marked X in the board diagram. Once you land on one of the four dark points marked X, you can take the shortcut on your next move. In the game, the player can choose the paths strategically, but in this task, <u>you should always take the shortest path if you land on the darker points marked X on the board.</u> Please note that the piece of a player is not at the starting point of the board but is outside of the board.

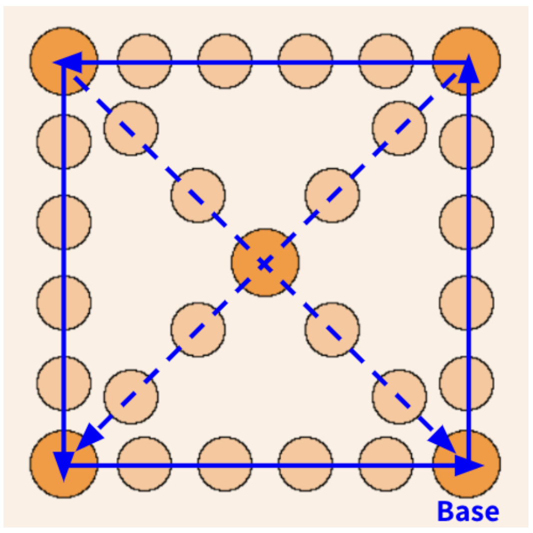


Figure 4a. Overall Direction

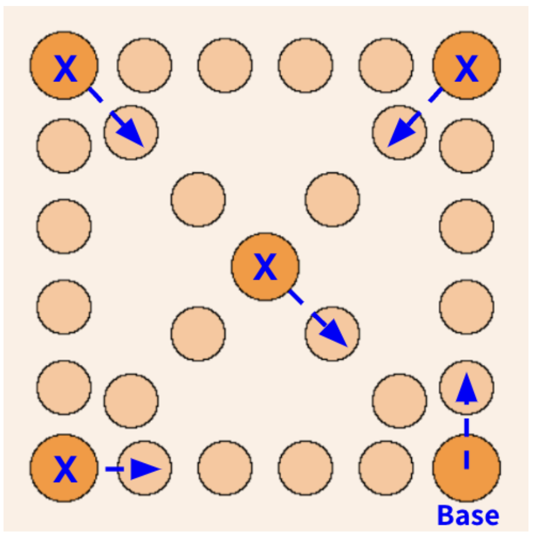


Figure 4b. Choosing Paths

4. **Starting and Ending Position**: In Yutnori, the starting position of the piece is the outside of the board. As in Figure 5, at the beginning of the game, pieces are placed outside, and they enter after throwing the Yut sticks. <u>For example, if a piece placed outside moves by the throwing outcome "Do", it should be placed at the point marked "1" in Figure 5.</u> Moreover, to end the game for a particular piece, it is not sufficient for a piece to return to the point marked "20" in Figure 5. <u>It should go at least one step further from that point to end its journey.</u>

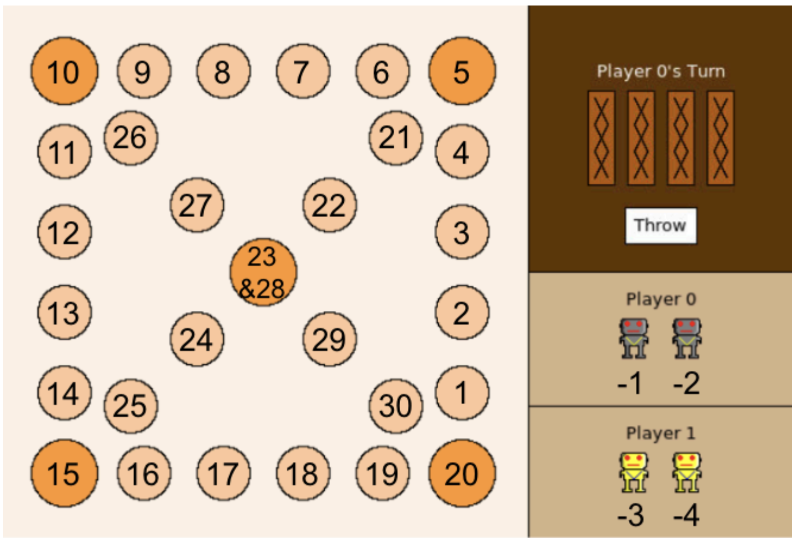


Figure 5: Game Setting

5. **Capturing Opponents**: If a player's piece lands in the same space as an opponent's piece, the opponent's piece is sent back to the starting point. The player who captured the opponent's piece gets another chance to throw the Yut sticks.
6. **Piggyback Pieces**: If two pieces of a player land in the same place, the pieces will be considered as the "piggyback" pieces. The "piggyback" pieces move together as a group rather than individually. This also means that when they are captured by the opponent, both of the pieces should go back to the starting point. Once they are captured, they will be handled individually as before.

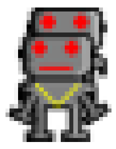


Figure 6a.
Player 0's Piggyback Pieces

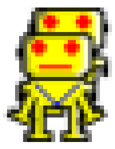


Figure 6b.
Player 1's Piggyback Pieces

7. **Winning the Game**: If all pieces of one player move around the board, the game ends, and the player who came first wins.
8. **Rules not Included**: We will not consider any additional rules beyond those described in the document. For instance, Back-Do or earning an extra throw when Yut or Mo is thrown would not be implemented.

Running the Code

In this task, you can play Yutnori by clicking the button “Throw” or the pieces. When each player’s turn starts, the Yut sticks can be thrown by clicking the “Throw” button. You can click the “Throw” button multiple times of your choice to re-throw the Yut sticks. (This feature is for your debugging. In the real world, each Player only throws once.) Afterward, you can click the piece that you want to move, and the piece moves according to the last outcome of Yut sticks (i.e., Do, Gae, Geol, Yut, Mo).

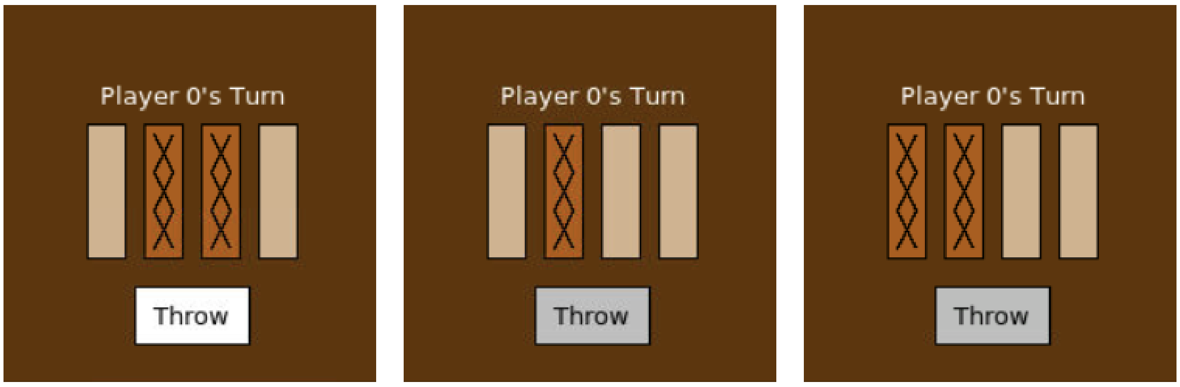


Table 2. Throwing Yut Sticks. You can throw them multiple times.

Important Notes

Please note the following instructions carefully. Failure to follow these will affect the grading of your work negatively. No exceptions will be made and no excuses will be accepted.

- Do NOT alter the reference point or depth of cs1graphics objects.
- Do NOT modify or implement code outside of sections marked as “Implement Here.”
- Do NOT change the parameters of any provided functions.
- Do NOT define new functions or methods in your code.

Grading Criteria

You should implement the code of mini Yutnori with 1 player and 1 piece [Task 1] and 2 players each with 2 pieces [Task 2].

Total Points: 100 Pts

- **Task 1 [50 Pts]**
  - Task 1.1 [5 Pts]
  - Task 1.2 [10 Pts]
  - Task 1.3 [20 Pts]
  - Task 1.4 [10 Pts]
  - Task 1.5 [5 Pts]
- **Task 2 [50 Pts]**
  - Task 2.1 [5 Pts]
  - Task 2.2 [15 Pts]
  - Task 2.3 [5 Pts]
  - Task 2.4 [25 Pts]

[Task 1] Yutnori Alone (50 Pts)

In this task, you will implement a mini Yutnori game with **1 player** and **1 piece**.

Piece
piece_id: int start_index: int current_index: int

Table 3a. Attributes of the `Piece` Class

Player
player_id: int piece: Piece

Table 3b. Attributes of the `Player` Class

Task 1.1: Implement `Piece.__init__(self, piece_id, start_index)` (5 pts)

Implement the constructor of the `Piece` class to initialize a piece object and set the attributes described in Table 3a. Note that each starting position of the piece is outside of the board. Please refer to Figures 7 and Section "How to Play #4" for the indices on the game board.

[Input]

- piece_id (int): id of the piece
- start_index (int): start index of the piece which is one of -1, -2, -3, -4

[Attributes]

- piece_id (int): id of the piece
- start_index (int): start index of the piece which is one of -1, -2, -3, -4
- current_index (int): current index of the piece which is initialized by start_index

Expected Results

```
>>> piece = Piece(piece_id=0, start_index=-1)
>>> print_attr(piece, "piece_id")
piece_id: 0
>>> print_attr(piece, "start_index")
start_index: -1
>>> print_attr(piece, "current_index")
current_index: -1
```

Task 1.2: Implement `Piece.get_point_by_current_index(self)` (10 pts)

Implement the method of the `Piece` class which returns the x and y coordinates of the piece in the canvas. The x and y coordinates should be obtained by the `current_index` attribute of the piece, which is assumed to be in {-4, -3, -2, -1, 1, 2, …, 30}. Please refer to Figures 7 and 8 for the indices and coordinates on the game board.

[Output]

- x, y (tuple of int): x and y coordinates where the piece should be located in the canvas

Expected Results

```
>>> piece = Piece(piece_id=0, start_index=-1)
>>> piece.current_index = 15
>>> x, y = piece.get_point_by_current_index()
>>> print(f"Coordinates of the piece: {(x, y)}")
Coordinates of the piece: (50, 350)
```

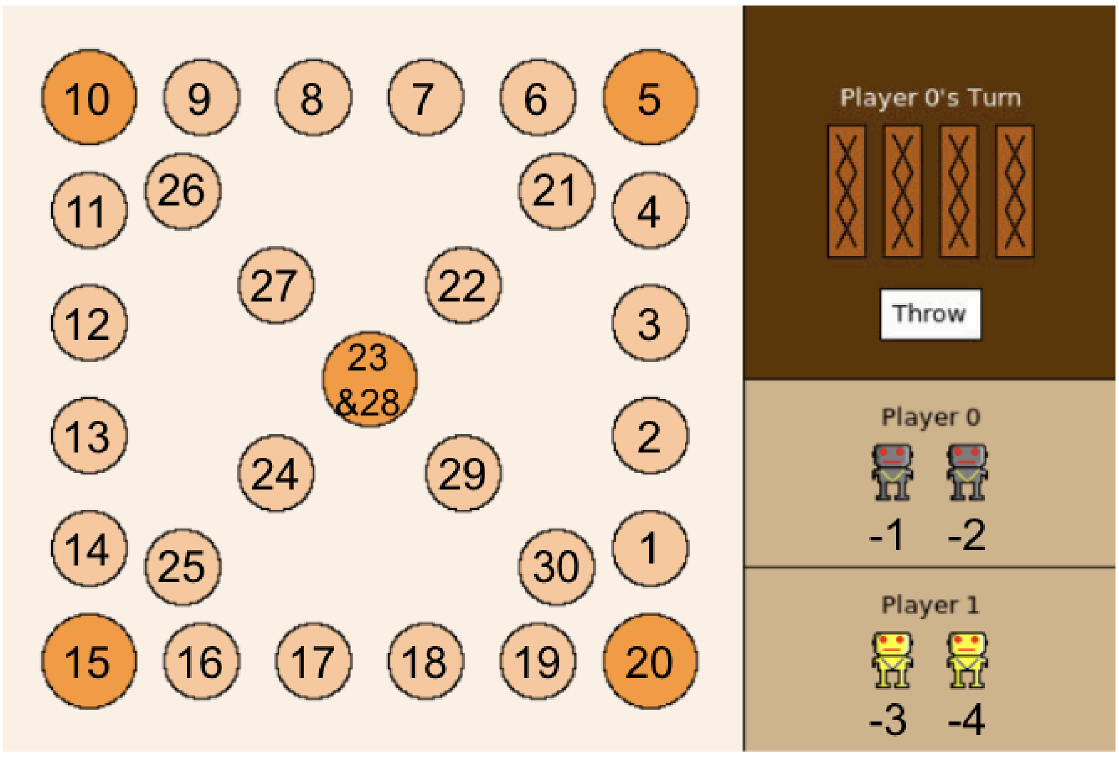


Figure 7. Indices on the Board

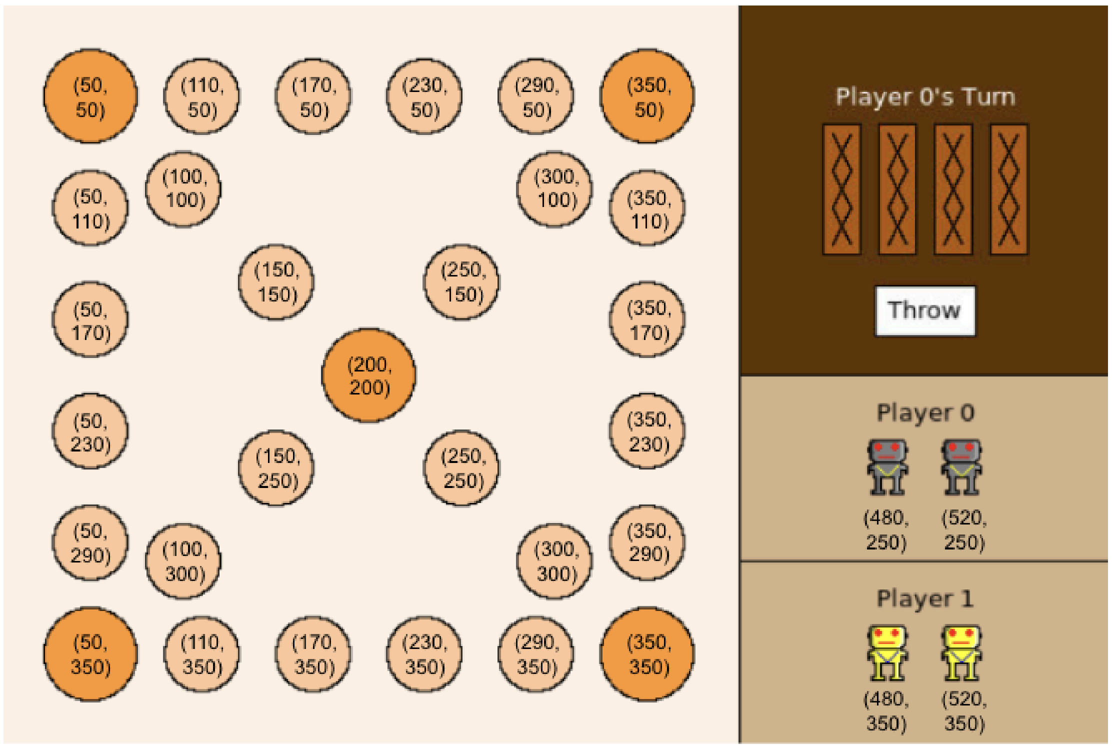


Figure 8. Points on the Board

Task 1.3: Implement `Piece.set_current_index(self, steps_to_move)` (20 pts)

Implement the method of the `Piece` class which takes the number of steps the piece should move in the current turn and updates the piece's `current_index` attribute accordingly. If the destination index is either 23 or 28, `current_index` can match either value. If the piece completes a full lap, then the `current_index` attribute should be set to 0.

[Input]

- steps_to_move (int): number of steps to move by Yut sticks in this turn

Expected Results

```
>>> piece = Piece(piece_id=0, start_index=-1)
>>> piece.current_index = 15
>>> piece.set_current_index(steps_to_move=5)
>>> print_attr(piece, "current_index")
current_index: 20
```

Task 1.4: Implement `Player.move_piece_by_yut_sticks(self, yut_sticks)` (10 pts)

Implement the method of the `Player` class which takes `yut_sticks`, a list of `YutStick` objects representing the outcome of each throw, and moves the piece of the player to the specified location. Calculate the number of steps to move the piece, then move the piece of the player by using the `set_current_index` method of the `Piece` class.

[Input]

- yut_sticks (list of str): list of 4 strings each of which is either "marked" or "unmarked"

Expected Results

```
>>> player = Player(player_id=0, start_index=-1)
>>> gae = ["marked", "unmarked", "marked", "unmarked"]
>>> player.move_piece_by_yut_sticks(yut_sticks=gae)
>>> print_attr(player.piece, "current_index")
current_index: 2
```

Task 1.5: Implement `Player.is_winner(self)` (5 pts)

Implement the method of the `Player` which determines whether the player currently wins the game. The method should return `True` if the player becomes the winner, `False` otherwise. <u>A player is the winner if every piece of the player currently completes a full round on the Yut board.</u>

[Output]

- is_player_winner (bool): whether the player is the current winner

Expected Results

```
>>> player = Player(player_id=0, start_index=-1)
>>> player.piece.current_index = 20
>>> is_winner = player.is_winner()
>>> print(f"Is Player {player.player_id} the winner?:
{is_winner}")
Is Player 0 the winner?: False
```

```
>>> do = ["marked", "unmarked", "marked", "marked"]
>>> player.move_piece_by_yut_sticks(yut_sticks=do)
>>> is_winner = player.is_winner()
>>> print(f"Is Player {player.player_id} the winner?:
{is_winner}")
Is Player 0 the winner?: True
```

[Task 2] Mini Yutnori (50 Pts)

In this task, you will develop a mini Yutnori game with **2 players** and **2 pieces per player**. Extend your Task 1 code to accomplish this. **Before you start Task 2, please copy and paste your code from Task 1.2 and Task 1.3.**

Piece
piece_id: int start_index: int current_index: int piggyback: bool

Table 4a. Attributes of the `Piece` Class

Player
player_id: int pieces: list of Piece

Table 4b. Attributes of the `Player` Class

Task 2.1: Implement `Piece.__init__(self, piece_id, start_index)` (5 pts)

Implement the constructor of the `Piece` class to create a piece object and set the attributes described in Table 4a.

[Input]

- piece_id (int): id of the piece which is one of 0, 1
- start_index (int): start index of the piece which is one of -1, -2, -3, -4

[Attribute]

- piece_id (int): id of the piece
- start_index (int): start index of the piece which is one of -1, -2, -3, -4
- current_index (int): current index of the piece which is initialized by start_index
- piggyback (bool): whether the piece is one of the "piggyback" pieces

Expected Results

```
>>> piece = Piece(piece_id=0, start_index=-1)
>>> print_attr(piece, "piece_id")
piece_id: 0
>>> print_attr(piece, "start_index")
start_index: -1
>>> print_attr(piece, "current_index")
current_index: -1
>>> print_attr(piece, "piggyback")
piggyback: False
```

Task 2.2: Implement `Player.move_piece_by_yut_sticks(self, piece_id, yut_sticks)` (15 pts)

Implement the method of the `Player` class which takes `piece_id` and `yut_sticks` to move the piece by the given piece id and the throwing outcome of Yut sticks. Calculate the number of steps, then move the piece of the player to the specified location by the `set_current_index` method of the `Piece` class. If the moving piece lands on the other piece of the current player, set the `piggyback` attributes of both two pieces of the player to `True`.

Notes

- Clicking the "piggyback" pieces in the same position executes this method once for each piece, resulting in a total of two executions.
- This method should only consider the case where the piece with the given piece_id moves. In other words, you should not move the other piece of the same player.

[Input]

- piece_id (int): id of the piece to be moved
- yut_sticks (list of str): list of 4 strings each of which is either “marked” or “unmarked”

Expected Results

```
>>> player = Player(player_id=0)
>>> player.pieces[0].current_index = -1
>>> player.pieces[1].current_index = 2
>>> gae = ["marked", "unmarked", "marked", "unmarked"]
>>> player.move_piece_by_yut_sticks(piece_id=0, yut_sticks=gae)
>>> print_attr(player.pieces[0], "current_index")
current_index: 2
>>> print_attr(player.pieces[0], "piggyback")
piggyback: True
>>> print_attr(player.pieces[1], "piggyback")
piggyback: True
```

Task 2.3: Implement `Player.is_winner(self)` (5 pts)

Implement the method of the `Player` which determines whether the player currently wins the game. The method should return `True` if the player becomes the winner, `False` otherwise. A player is the winner if every piece of the player currently completes a full round on the Yut board.

[Output]

- is_player_winner (bool): whether the player is the current winner

Expected Results

```
>>> player = Player(player_id=0)
>>> player.pieces[0].current_index = 0
>>> player.pieces[1].current_index = 20
>>> is_winner = player.is_winner()
>>> print(f"Is Player {player.player_id} the winner?:
{is_winner}")
Is Player 0 the winner?: False
>>> do = ["marked", "unmarked", "marked", "marked"]
>>> player.move_piece_by_yut_sticks(piece_id=1, yut_sticks=do)
>>> is_winner = player.is_winner()
>>> print(f"Is Player {player.player_id} the winner?:
{is_winner}")
Is Player 0 the winner?: True
```

Task 2.4: Implement `capture_and_get_next_players(current_player, current_opponent)` (25 pts)

Implement a function that takes the current player and the opponent player objects as input, checks if any pieces of the opponent are captured, and determines the next player accordingly. This function is called after the current player moves their pieces, and does the following:

1. If any current player's pieces capture the opponent's pieces, update the captured pieces by setting their `piggyback` attributes to `False`, and `current_index` attributes to their starting position, `start_index`.
2. This function returns a tuple of two Player objects: the next player and their opponent for the upcoming turn. If any pieces were captured, the turn remains the same. Both returned player objects must be the exact same objects as the ones provided as inputs, not newly initialized instances.

[Input]

- current_player (Player): current player's object
- opponent_player (Player): the other player's object

[Output]

- next_player, next_opponent (tuple of Player): the next player and its opponent player for the next iteration

Expected Results

```
>>> player = Player(player_id=0)
>>> opponent = Player(player_id=1)
>>> player.pieces[0].current_index = 0
>>> player.pieces[1].current_index = 2
>>> opponent.pieces[0].current_index = 0
>>> opponent.pieces[1].current_index = 4
>>> gae = ["marked", "unmarked", "marked", "unmarked"]
>>> player.move_piece_by_yut_sticks(piece_id=1, yut_sticks=gae)
>>> next_player, next_opponent =
capture_and_get_next_players(player, opponent)
>>> print(f"Captured piece goes to index
{opponent.pieces[1].current_index}.")
Captured piece goes to index -4.
>>> print(f"Next player is Player {next_player.player_id}.")
Next player is Player 0.
>>> print(f"Next opponent is Player {next_opponent.player_id}.")
Next opponent is Player 1.
```