CMPUT 175 - Winter 2023

Assignment 2

Opened: Wednesday, 22 February 2023, 12:00 AM
Due: Friday, 17 March 2023, 11:55 PM

Assignment 2 - Goat Race

Due Date: March 17th 2023 at 23:55

Percentage overall grade: 7%
Penalties: No late assignments allowed
Maximum Marks: 100

Goal: Hands-on experience with class building and encapsulation. Understand how classes can be used to represent complex objects, and use methods to interact with the class state. Get familiar with reusing code. Realize the importance of user input validation and Exceptions.

Read the whole document before starting to solve the Assignment problem.

Submit only your work. While you can collaborate and help each other, do not share code. If you collaborate, acknowledge your collaborator in the code.

Follow the Software Quality Requirements.

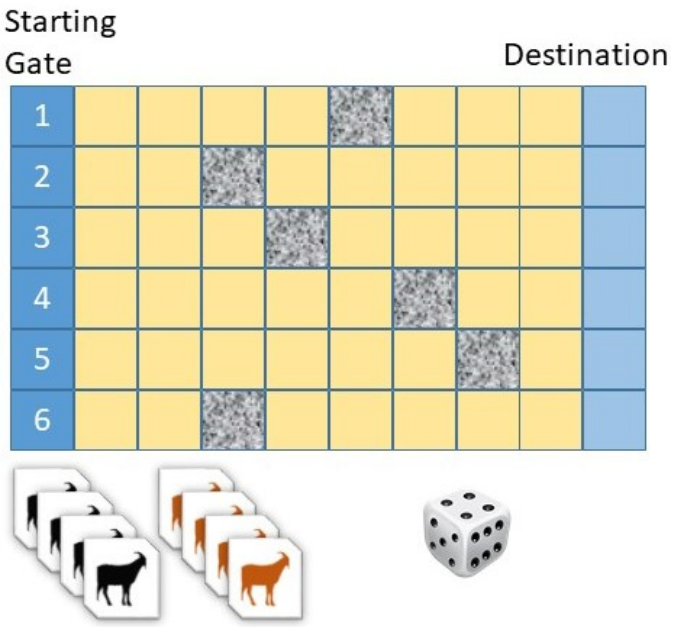
Assignment Specifications

Game description :

Goat Race is a multiplayer board game that can be played by as little as two and up to five players. It is played on a rectangular board that contains 54 squares arranged in a grid that contains six rows by nine columns. To refer to the positions in the board, we refer to columns using letters from A to I, and rows from 1 to 6.

Each row has one **obsta**cle positioned randomly at a given column. The first and last column (i.e. the Gates and the Destination) cannot be blocked by **obsta**cles. Goats can be positioned on any square, alone or stacked with other goats, except for the squares that have an **obsta**cle. These will have to be circumvented moving sideways, along the column before the **obsta**cle.

Each player receives four pieces in the shape of goats, because goats can climb on just anything, including themselves! The first column of the board is the Starting Gate. All goats must first be placed on the starting gate before the race starts. The last column is the Destination, and is where you should take your goats to win the game. The objective of the game is to take three out of your four goats from the Starting Gate to the Destination. The first player to make three out of their four goats reach the destination wins the race.



Setup

To set up the game, each player gets four pieces of the same color in the shape of a goat. A standard 6 faced dice is shared among all the players.

Phase 1: Order of play

To decide the order in which players play, each player throws the dice. The players can chose any order to throw the dice.

Players will play in descending order of their dice (i.e. highest goes first).

In case two players roll the same number, the player that rolled the dice last plays first.

Phase 2: Placement

After all players have rolled the dice and the order of play has been decided, the Placement phase begins.

The placement phase is when the goats will be placed on the board.

During the placement phase, each player, on their turn, places a goat in a row of their choice on the Starting Gate (the first column). After all rows contain one goat, the next player may place a goat on top of any stack that contains ONE goat. The next player must place a goat on top of any stack that contains ONE goat, until all the rows contain two goats. You cannot place a goat on top of a stack that is bigger than the smallest stack (this only applies to this phase).

Example 1:

```
    A   B   C   D   E   F   G   H   I
  +---+---+---+---+---+---+---+---+---+
1 | B |   | X |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+
2 | W |   |   |   | X |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+
3 | B |   |   |   |   |   | X |   |   |
  +---+---+---+---+---+---+---+---+---+
4 |   |   |   |   | X |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+
5 |   |   |   |   |   |   | X |   |   |
  +---+---+---+---+---+---+---+---+---+
6 | W |   | X |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+
```

In the example above, the player may place a goat in row 4 or 5. Rows 1, 2, 3, and 6 already contain one goat, so the player must place a goat on a row that contains no goats.

In the output above, X represents an obstacle, B and W represent A black goat and a white goat respectively.

Example 2:

```
    A   B   C   D   E   F   G   H   I
  +---+---+---+---+---+---+---+---+---+
1 | B |   | X |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+
2 | W |   |   |   | X |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+
3 | B |   |   |   |   |   | X |   |   |
  +---+---+---+---+---+---+---+---+---+
4 | B |   |   | X |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+
5 | W |   |   |   |   |   | X |   |   |
  +---+---+---+---+---+---+---+---+---+
6 | W |   | X |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+
```

In the example above, rows 1 and 4 contain 2 goats, one on top of the other (except the top, the goats are hidden in the stack. Players have to remember where goats are located). Thus, the player may place a goat only in rows 2, 3, 5, or 6 since it can only play on rows that are the same size as the smallest stack. After all goats of all players are placed on the start column, we start the Goat Race phase.

Phase 3: Goat Race

During the Goat Race Phase, each player's turn consists of three actions, from which two are mandatory and one is optional. Actions follow this order:

1. The player MUST first roll the die
2. OPTIONAL, the player may elect to move one of their goats sideways, that is, one row up or down, but keeping the goat in the same column. Only the top goat in a square can be moved.
3. The player **must** move forward (toward the Destination), that is, one column to the right, **any** goat piece that is in the lane (row) that matches the number shown in the face up of the dice. If there are no goats in the lane that matches the number in the face up of the dice, the turn is over. Remember, a goat cannot go to a square where there is an **obsta**cle.

Example :

Consider the following board where X is an **obsta**cle, B, W, O are Black, White and Orange respectively:

```
    A   B   C   D   E   F   G   H   I
  +---+---+---+---+---+---+---+---+---+
1 | B |   | X |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+
2 | O |   |   |   | X |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+
3 |   | O |   | W |   |   | X |   |   |
  +---+---+---+---+---+---+---+---+---+
4 |   |   | W | X |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+
5 | B |   |   |   |   |   | X |   |   |
  +---+---+---+---+---+---+---+---+---+
6 | B |   | X |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+
```

Let's say that the current player has white goats. There are 3 players in the match.

1. The player rolls the dice and gets a 3.

2. The player MAY elect to move ONE of their goats sideways. If they choose to do so, they must peek at the top of each stack and look at their color. If there is a white goat on the top of the stack, the player may move the goat ONE row above or below its current position. After looking at their valid moves, the player sees that they have a white goat in positions C4 and D3. Their valid actions are moving the goat in C4 to C3 or C5, OR moving the goat in D3 to D2. The player decides to move the goat in C4 to C5.

The board would look like this:

```
    A   B   C   D   E   F   G   H   I
  +---+---+---+---+---+---+---+---+---+
1 | B |   | X |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+
2 | O |   |   |   | X |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+
3 |   | O |   | W |   |   | X |   |   |
  +---+---+---+---+---+---+---+---+---+
4 |   |   | W | X |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+
5 | B |   | W |   |   |   | X |   |   |
  +---+---+---+---+---+---+---+---+---+
6 | B |   | X |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+
```

Notice that a goat is still in C4. This can happen if there was already a goat in the stack.

3. The player MUST move a goat (it may be a goat of any color) in row 3 forward. The player has two valid actions: moving the goat in B3 to C3 or moving the goat in D3 to E3. The player decides to move the goat in B3 to C3. This ends the player's turn.

The board now looks like this:

```
    A   B   C   D   E   F   G   H   I
  +---+---+---+---+---+---+---+---+---+
1 | B |   | X |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+
2 | O |   |   |   | X |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+
3 |   |   | O | W |   |   | X |   |   |
  +---+---+---+---+---+---+---+---+---+
4 |   |   | W | X |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+
5 | B |   | W |   |   |   | X |   |   |
  +---+---+---+---+---+---+---+---+---+
6 | B |   | X |   |   |   |   |   |   |
  +---+---+---+---+---+---+---+---+---+
```

If no goats can be moved in the lane that matches the number in the face-up of the dice, the turn is over.

Endgame

The game ends when one of the following conditions is met:

1. All players, for any reason, cannot move any of their goats forward. The game is considered a draw.
2. There are three goats of the same color in the last column. The player whose three goats reach the last column first (Destination) wins the game.

Assignment Specifications

Now that you understand the rules of the game, your objective is to create the following 5 classes that allow playing the goat race. In all of them, **you are supposed to raise Exceptions appropriately** (like you did in Lab 5).

Task 1: Implement the Stack class

The Stack class is a simple stack implementation. It is a generic class, meaning that it can hold any type of object. It must have the following functionalities:

- Push one new item at a time to the stack
- Pop the item on the top of the stack
- Peek at the item in the top of the stack
- Check if the stack is empty
- Check the stack size

Task 2: Implement the Goat class

The Goat class represents a goat in the game. Goats can have one out of five colours: "WHITE", "BLACK", "RED", "ORANGE", "GREEN". They also hold the row and column where they are located. The Goat object should take in the colour as arguments when initialized. The class must have the following functionalities:

- Print a string representing the goat -- the output should be Column + Row. eg. A1, where the column and the row correspond to the current location of the Goat.
- Get the row/column where the goat is located. Return -1 otherwise (i.e. when the location of the Goat hasn't been assigned yet)
- Get the color of the goat
- Be able to set the row/column of the goat
- Validate inputs, throwing exceptions as necessary

Task 3: Implement the Player class

Create a Player class that represents a player in the game. The player has a color (string with 5 possible values: "WHITE", "BLACK", "RED", "ORANGE", "GREEN") and a list of Goat objects. The class should allow adding/removing goats, retrieving the color, and retrieving the number of goats. The class should also have a string representation that shows the color and list of goats.

Example output of **__str__** method:

```
WHITE
Goats:
WHITE A1
WHITE C1
WHITE E1
WHITE B1
```

Task 4: Implement the Board class

The Board class represents the board in the game. The Board class should have at least the following methods:

- **__init__(self, width, height, obstacle_positions)**: This method initializes the board. It takes three arguments, the width and height of the board, and a list that contains tuples indicating the position of obstacles, that is, (row, column).

- **check_row(self, row)**: Checks if the given value for the row is valid (should be a number between 1 and height), and, if this is not the case, it raises an Exception.

- **check_column(self, column)**: Checks if the given value for the column is valid (should be a valid character), and, if this is not the case, it raises an Exception.

- **check_obstacle_positions(self, obstacle_positions)**: Checks that the given list of obstacles has valid rows and columns for each obstacle.

- **get_width(self)**: This method returns the width of the board. It takes no arguments and returns an integer.

- **get_height(self)**: This method returns the height of the board. It takes no arguments and returns an integer.

- **get_board(self)**: This method returns the 2D list representing the board. It takes no arguments and returns a 2D list of Stack objects.

-**__str__(self)**: This method returns a string representation of the board. It takes no arguments and returns a string.

- If a stack has no goats, it should be represented by an empty space
- If a stack has one or more goats, it should be represented by the first letter of the top goat's color
- If there is an obstacle in the position, it should be represented by a capital letter 'X'

Example output of **__str__** method:

```
      A   B   C   D   E   F   G   H   I
   +---+---+---+---+---+---+---+---+---+
 1 |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+
 2 |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+
 3 |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+
 4 |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+
 5 |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+
 6 |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+
```

Task 5: Implement the Game class

The Game class represents the game.

Attributes

- board: The board in the game. This is a Board object.

- players: A list of players in the game. This is a list of Player objects.

- phase: The phase of the game. This is an integer that can be 1, 2, or 3.

- turn: The turn of the game. This is an integer indicating the index of the player whose turn it is in the list of players.

The Game class must have at least the following methods:

- **__init__(self, width, height, obstacle_positions)**: This method initializes the game. It takes three arguments, the width and height of the board, and a list that contains tuples indicating the position of obstacles, that is, (row, column).

- **__str__(self)**: This method returns a string representation of the game. It takes no arguments and returns a string.

- **get_phase(self)**: This method returns the phase of the game. It takes no arguments and returns an integer.

- **get_turn(self)**: This method returns the turn of the game. It takes no arguments and returns an integer.

- **get_current_player(self)**: This method returns the player corresponding to the current turn.

- **get_goats_blocked(self, player)**: This method returns the number of goats that cannot jump for the given player.

- **get_goats_per_player(self)**: This method returns a list with the number of goats of all players.

- **set_phase(self, phase)**: This method sets the phase of the game. It takes one argument, the phase to set, and returns nothing.

- **set_turn(self, turn)**: This method sets the turn of the game. It takes one argument, the turn to set, and returns nothing.

- **add_player(self, player)**: This method adds a player to the list of players in the game. It takes one argument, the player to add, and returns nothing.

- **add_goat(self, row, column)**: This method adds a goat to the stack in the location given, if possible. If it is not possible, an Exception is raised.

- **move_sideways(self, move)**: This method checks if the given sideways move is valid and executes it. If the move is invalid, it raises an Exception.

- **move_forward(self, move, dice_outcome)**: This method checks if the forward move is valid and executes it. The dice_outcome is used to determine whether the move corresponds to the appropriate row or not. If the move is invalid, it raises an Exception.

- **check_row(self, row)**: Checks if the given row value is valid (should be a number between 1 and the height of the board), and, if this is not the case, it raises an Exception.

- **check_valid_move_format(self, move)**: Checks if the move is a list of two tuples, and if each tuple is contained by a valid row and a valid column. If this is not the case, it raises an Exception.

- **check_nonempty_row(self, row)**: This method returns whether the given row has goats that can jump or not.

- **check_starting_goat_placement(self, row)**: This method returns whether the row is valid to make a goat placement or not. In particular, it checks if the stack in the row is not greater than the stack with minimum height.

- **check_winner(self)**: This method returns whether there is a winner or not, that is, if some player got 3 goats to the Destination.

- **check_tie(self)**: This method returns whether there is a tie or not, that is, if no player can move since all their goats are blocked.

Example output of **__str__** method:

```
      A   B   C   D   E   F   G   H   I
   +---+---+---+---+---+---+---+---+---+
 1 |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+
 2 |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+
 3 |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+
 4 |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+
 5 |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+
 6 |   |   |   |   |   |   |   |   |   |
   +---+---+---+---+---+---+---+---+---+

Players: WHITE, BLACK, RED
Phase: 3
Player whose turn it is: BLACK
```

General Guidelines

In addition to making sure that your code runs properly, we will also check that you follow good programming practices. For example, divide the problem into smaller sub-problems, and write functions to solve those sub-problems so that each function has a single purpose; use the most appropriate data structures for your algorithm; use concise but descriptive variable names; define constants instead of hardcoding literal values throughout your code; include meaningful comments to document your code, as well as docstrings for all functions; and **be sure to acknowledge any collaborators/references in a header comment at the top of your Python file**.

Restrictions for this assignment: you cannot use break/continue, and you cannot import any modules apart from the files containing other classes. Importing other modules will result in deductions.

Assignment Deliverables

- ○ You are going to submit 5 Python files, one for each of the described classes:
  - stack.py
  - goat.py
  - player.py
  - board.py
  - game.py