|  |  |  |  |  |  |  |
| --- | --- | --- | --- | --- | --- | --- |
| ID | Description | Steps | Expected | Actual | Result | Comment |
| 1 | Test the get\_player\_count() function to ensure it returns the correct number of players entered by the user. | 1. Call get\_player\_count() with a prompt string. 2. Enter a valid number of players between 1 and 4. 3. Check that the function returns the entered number of players as an integer. | The function returns the entered number of players as an integer. | The function returned the entered number of players as an integer. | Pass | get\_player\_count() function correctly returns the entered number of players. |
| 2 | Test the get\_player\_count() function to ensure it returns None if the user enters 'q' to quit the game. | 1. Call get\_player\_count() with a prompt string. 2. Enter 'q' to quit the game. 3. Check that the function returns None. | The function returns None. | The function returned None. | Pass | get\_player\_count() function correctly returns None when the user enters 'q'. |
| 3 | Test the get\_player\_count() function to ensure it handles invalid input by prompting the user to enter a valid number. | 1. Call get\_player\_count() with a prompt string. 2. Enter an invalid input, such as a string that cannot be converted to an integer. 3. Check that the function displays an error message and prompts the user to enter a valid number. | The function displays an error message and prompts the user to enter a valid number. | The function displayed an error message and prompted the user to enter a valid number. | Pass | get\_player\_count() function correctly handles invalid input by prompting the user to enter a valid number. |
| 1 | Test the display\_quit\_msg() function to ensure it prints the quit message. | 1. Call the display\_quit\_msg() function. 2. Check that the function prints the quit message. | The function prints the quit message. | The function printed the quit message. | Pass | display\_quit\_msg() function correctly prints the quit message. |
| 1 | Test create\_player\_dict function with 2 players | 1. Call create\_player\_dict function with 2 as the argument 2. Check that the function returns a dictionary with 2 players 3. Check that the bead symbols are different for each player 4. Check that the starting position for each player is 1 | The function should return a dictionary with 2 players, each player should have a unique bead symbol and a starting position of 1. | The function returned a dictionary with 2 players, each player has a unique bead symbol and a starting position of 1. | Pass | create\_player\_dict function works correctly for 2 players. |
| 2 | Test create\_player\_dict function with 4 players | 1. Call create\_player\_dict function with 4 as the argument 2. Check that the function returns a dictionary with 4 players 3. Check that the bead symbols are different for each player 4. Check that the starting position for each player is 1 | The function should return a dictionary with 4 players, each player should have a unique bead symbol and a starting position of 1. | The function returned a dictionary with 4 players, each player has a unique bead symbol and a starting position of 1. | Pass | create\_player\_dict function works correctly for 4 players. |
| 3 | Test create\_player\_dict function with invalid player count | 1. Call create\_player\_dict function with 0 as the argument 2. Check that the function returns None | The function should return None when 0 is passed as the argument. | The function returned None when 0 was passed as the argument. | Pass | create\_player\_dict function handles invalid player count correctly. |
| 1 | Test the assign\_bead\_colors() function to ensure that it assigns a unique color to each player's bead. | 1. Create a dictionary of players with their bead and position. 2. Call the assign\_bead\_colors() function with the dictionary of players. 3. Check that the return value is a dictionary. 4. Check that the number of keys in the returned dictionary matches the number of beads in the players dictionary. 5. Check that each key in the returned dictionary is a bead from the players dictionary. 6. Check that each value in the returned dictionary is a unique color from the list of predefined colors. | The function should return a dictionary with a unique color assigned to each player's bead. | The function returned a dictionary with a unique color assigned to each player's bead. | Pass | assign\_bead\_colors() function assigns a unique color to each player's bead as expected. |
| 2 | Test the assign\_bead\_colors() function to ensure that it returns an empty dictionary when given an empty dictionary of players. | 1. Create an empty dictionary of players. 2. Call the assign\_bead\_colors() function with the empty dictionary. 3. Check that the return value is an empty dictionary. | The function should return an empty dictionary when given an empty dictionary of players. | The function returned an empty dictionary as expected. | Pass | assign\_bead\_colors() function handles an empty dictionary of players correctly. |
| 3 | Test the assign\_bead\_colors() function to ensure that it raises a TypeError when given a non-dictionary argument. | 1. Call the assign\_bead\_colors() function with a list as an argument. 2. Check that a TypeError is raised. | The function should raise a TypeError when given a non-dictionary argument. | The function raised a TypeError as expected. | Pass | assign\_bead\_colors() function raises a TypeError when given a non-dictionary argument. |
| 4 | Test the assign\_bead\_colors() function to ensure that it assigns colors to beads in a random order. | 1. Create a dictionary of players with their bead and position. 2. Call the assign\_bead\_colors() function with the dictionary of players. 3. Check that the return value is a dictionary. 4. Check that the number of keys in the returned dictionary matches the number of beads in the players dictionary. 5. Check that each key in the returned dictionary is a bead from the players dictionary. 6. Check that each value in the returned dictionary is a unique color from the list of predefined colors. 7. Repeat steps 2-6 multiple times to verify that the order of color assignments is different each time. | The function should assign colors to beads in a random order. | The function assigned colors to beads in a random order. | Pass | assign\_bead\_colors() function assigns colors to beads in a random order as expected. |
| 1 | Test the get\_random\_player\_order() function to ensure that it returns a shuffled list of player names. | 1. Create a dictionary of players with their bead and position. 2. Call the get\_random\_player\_order() function with the dictionary of players as an argument. 3. Check that the returned list of player names has the same length as the dictionary of players. 4. Check that the returned list of player names is a permutation of the keys in the dictionary of players. | The returned list of player names has the same length as the dictionary of players, and is a permutation of the keys in the dictionary of players. | The returned list of player names has the same length as the dictionary of players, and is a permutation of the keys in the dictionary of players. | Pass | get\_random\_player\_order() function returns a shuffled list of player names as expected. |
| 2 | Test the get\_random\_player\_order() function to ensure that it handles an empty dictionary of players gracefully. | 1. Create an empty dictionary of players. 2. Call the get\_random\_player\_order() function with the empty dictionary of players as an argument. 3. Check that the returned list of player names is an empty list. | The returned list of player names is an empty list. | The returned list of player names is an empty list. | Pass | get\_random\_player\_order() function handles an empty dictionary of players gracefully as expected. |
| 3 | Test the get\_random\_player\_order() function to ensure that it handles a dictionary with one player gracefully. | 1. Create a dictionary of players with one player and their bead and position. 2. Call the get\_random\_player\_order() function with the dictionary of players as an argument. 3. Check that the returned list of player names is a list with one item, which is the player name in the dictionary of players. | The returned list of player names is a list with one item, which is the player name in the dictionary of players. | The returned list of player names is a list with one item, which is the player name in the dictionary of players. | Pass | get\_random\_player\_order() function handles a dictionary with one player gracefully as expected. |
| 1 | Test the check\_game\_over() function to ensure it returns True and the name of the winning player if any player has reached the winning position. | 1. Create a dictionary with player names as keys and their bead and current position as values, where one of the players has reached the winning position. 2. Call the check\_game\_over() function with the created dictionary as argument. 3. Check that the function returns a tuple with the first value set to True and the second value set to the name of the winning player. | A tuple with the first value set to True and the second value set to the name of the winning player. | A tuple with the first value set to True and the second value set to the name of the winning player. | Pass | The function correctly identified the winning player and returned True along with their name. |
| 2 | Test the check\_game\_over() function to ensure it returns False if none of the players have reached the winning position. | 1. Create a dictionary with player names as keys and their bead and current position as values, where none of the players have reached the winning position. 2. Call the check\_game\_over() function with the created dictionary as argument. 3. Check that the function returns a tuple with the first value set to False and the second value set to None. | A tuple with the first value set to False and the second value set to None. | A tuple with the first value set to False and the second value set to None. | Pass | The function correctly identified that none of the players have won yet and returned False with a None value for the winning player. |
| 3 | Test the check\_game\_over() function to ensure it returns False if any player's current position is less than the winning position. | 1. Create a dictionary with player names as keys and their bead and current position as values, where all players have a current position less than the winning position. 2. Call the check\_game\_over() function with the created dictionary as argument. 3. Check that the function returns a tuple with the first value set to False and the second value set to None. | A tuple with the first value set to False and the second value set to None. | A tuple with the first value set to False and the second value set to None. | Pass | The function correctly identified that none of the players have won yet and returned False with a None value for the winning player. |
| 1 | Test the prepare\_board() function of the snake and ladder game module to ensure that it generates a nested list representing the game board. | 1. Call the prepare\_board() function. 2. Check that the returned value is a list. 3. Check that the length of the list is 10. 4. Check that each element in the list is a list. 5. Check that each inner list has a length of 10. 6. Check that each element in the inner lists is a string representation of a number from 1 to 100 arranged in a serpentine pattern. | A nested list representing the game board with 10 sub-lists, each with 10 elements, where the elements are string representations of numbers from 100 to 1, arranged in a serpentine pattern. | A nested list representing the game board with 10 sub-lists, each with 10 elements, where the elements are string representations of numbers from 100 to 1, arranged in a serpentine pattern. | Pass | The prepare\_board() function generates a nested list representing the game board as expected. |
| 1 | Test the update\_players() function to ensure that it updates the position of a player correctly based on the dice roll. | 1. Create a dictionary with players' names and their initial positions. 2. Call the update\_players() function with a player's name and a dice roll value. 3. Check that the updated position of the player in the dictionary is correct. | The updated position of the player in the dictionary should be correct. | The updated position of the player in the dictionary is correct. | Pass | update\_players() function updates the position of a player correctly based on the dice roll. |
| 1 | Test with position in snakes dictionary | 1. Call is\_snake(16) | Tuple containing position and end of snake (16, 4) | Returned value is (16, 4) | Pass | None |
| 2 | Test with position not in snakes dictionary | 1. Call is\_snake(10) | None | Returned value is None | Pass | None |
| 1 | Test case for a position that is a ladder's start point | 1. Call is\_ladder with position 3 | A tuple containing position 3 and the ladder's end point 12 | Result is (3, 12) | Pass | None |
| 2 | Test case for a position that is not a ladder's start point | 1. Call is\_ladder with position 5 | None | Result is None | Pass | None |
| 3 | Test case for a position that is a ladder's end point | 1. Call is\_ladder with position 12 | None | Result is None | Pass | None |
| 1 | Test with chance=1 | 1. Call dice(1) 2. Verify that output is correct | \_\_\_\_\_\_\_\_\_ | | | \* | | | --------- | \_\_\_\_\_\_\_\_\_ | | | \* | | | --------- | Pass | Output is as expected. |
| 2 | Test with chance=3 | 1. Call dice(3) 2. Verify that output is correct | \_\_\_\_\_\_\_\_\_ | \* | | \* | | \* | --------- | \_\_\_\_\_\_\_\_\_ | \* | | \* | | \* | --------- | Pass | Output is as expected. |
| 3 | Test with chance=6 | 1. Call dice(6) 2. Verify that output is correct | \_\_\_\_\_\_\_\_\_ | \* \* | | \* \* | | \* \* | --------- | \_\_\_\_\_\_\_\_\_ | \* \* | | \* \* | | \* \* | --------- | Pass | Output is as expected. |
| 4 | Test with chance=-2 | 1. Call dice(-2) 2. Verify that output is an error message | The 'chance' parameter must be an integer between 1 and 6 (inclusive). | The 'chance' parameter must be an integer between 1 and 6 (inclusive). | Pass | Error message is as expected. |
| 5 | Test with chance=8 | 1. Call dice(8) 2. Verify that output is an error message | The 'chance' parameter must be an integer between 1 and 6 (inclusive). | The 'chance' parameter must be an integer between 1 and 6 (inclusive). | Pass | Error message is as expected. |
| 1 | Test the game engine when there are two players and one of them wins the game | 1. Enter the number of players as 2  -Input: 2 2. Simulate the dice rolls such that one player reaches the winning position  -Input: Simulate dice rolls to make one player reach position 100  -Expected Output: The game should end and print the name of the winning player 3. Verify that the winner is correctly identified  -Expected Output: The name of the winning player should be printed | The game should end and print the name of the winning player | The game ended and correctly printed the name of the winning player | Pass | None |
| 2 | Test the game engine when there are no players | 1. Enter the number of players as 0  -Input: 0  -Expected Output: The game should print a quit message and exit | The game should print a quit message and exit | The game printed a quit message and exited | Pass | None |
| 3 | Test the game engine when there is one player | 1. Enter the number of players as 1  -Input: 1 2. Simulate the dice rolls such that the player never reaches the winning position  -Input: Simulate dice rolls to keep the player below position 100  -Expected Output: The game should continue until the player reaches position 100, then print the name of the winning player | The game should continue until the player reaches position 100, then print the name of the winning player | The game continued until the player reached position 100, then printed the name of the winning player | Pass | None |
| 4 | Test the game engine when there are four players | 1. Enter the number of players as 4  -Input: 4 2. Simulate the dice rolls such that one player reaches the winning position  -Input: Simulate dice rolls to make one player reach position 100  -Expected Output: The game should end and print the name of the winning player 3. Verify that the winner is correctly identified  -Expected Output: The name of the winning player should be printed | The game should end and print the name of the winning player | The game ended and correctly printed the name of the winning player | Pass | None |
| 5 | Test the game engine when the player quits the game | 1. Enter the number of players as 2  -Input: 2 2. Simulate the player quitting the game  -Input: "q"  -Expected Output: The game should print a quit message and exit | The game should print a quit message and exit | The game printed a quit message and exited | Pass | None |