Poker Dice

Figure 1: Poker Dice

Introduction

Poker dice[1] is a simple game that involves rolling **five dice** and using the resulting values to make poker dice hands (see description of poker dice hands below). Each die has the values 1, 2, 3, 4, 5, and 6 on its faces, and the player **rolls all five dice at the start of her turn**. The player then has the option to **re-roll any or all of the dice up to two more times**, trying to make the best poker hand possible. After the third roll, the player's final dice values are used to determine their hand, and the hands are ranked according to the poker hand rankings mentioned below.

The player with the best hand wins the game. If both players have the same hand value, we have a tie.

The poker dice hands are (sorted by value with the best poker hand on top):

Hand	Description	Example
Five of a kind	Five identical dice values	3 3 3 3 3
Four of a kind	Four identical dice values	2 2 2 2 1
Full House	Three of a kind and a pair in a single hand	1 1 1 4 4
Straight	Five dice values in sequential rank	2 3 4 5 6
Three of a kind	Three identical dice values	2 2 2 5 6

[1] https://en.wikipedia.org/wiki/Poker_dice

Hand	Description	Example
Two pairs	Two pairs in a single hand	2 2 3 3 6
One pair	One pair	1 2 2 3 4
Bust	All dice values are different	1 3 4 5 6

Levels

This exercise is structured into four levels:

- Level 1 contains minimum requirements. Given enough time, everybody should be able to write this program correctly without a lot of help from teachers or other students.
- Level 2 is harder. You will probably need a little bit of help from teachers or advanced students to complete it.
- Level 3 is for advanced students who would like to enhance their coding skills even further.
- Level 4 brings everything together. Once you solved level 3, level 4 should not be a big problem.

Level 1: Minimum Requirements

In this level, you have to implement a single-player game without hand evaluation.

First a tip to get started: Create five variables for the five dice of your game. Additionally, create five variables storing the information which dice are fixed (i.e. not re-rolled).

```
int dice1 = 0, dice2 = 0, dice3 = 0, dice4 = 0, dice5 = 0;
bool fixed1 = false, fixed2 = false, fixed3 = false, fixed4 = false, fixed5
↪  = false;
```

- Implement a method `RollDice`. It should assign all dice that are currently not fixed with a random number between 1 and 6.
- Implement a method `PrintDice`. It should print the values of all dice on the screen (e.g. `2, 2, 1, 6, 5`).
- Implement a method `FixDice`.
  - First, it resets all previous fixing of dice. All dice are set to be re-rolled.
  - The user can fix/unfix dice by entering the dice number (e.g. input *1* fixes the first dice if it has been unfixed before, otherwise it unfixes the first dice). You can use `Console.ReadLine` or `Console.ReadKey`, whatever you prefer.
  - Keep asking the user about fixed dice until she presses *r* (for "*r*oll the dice").

- Implement the main game logic using the methods mentioned above.
    1. Roll the dice (method `RollDice`) and print the dice values on the screen (method `PrintDice`).
    2. Ask the user for fixed dice (method `FixDice`). If all dice are fixed, the game ends because the user does not roll the dice again.
    3. Repeat steps 1 and 2.
    4. Roll the dice for the final time and print the dice values on the screen.

This is a sample input/output for level 1:

```
Dice roll #1 (out of 3): 2, 2, 1, 6, 5
Which dice do you want to fix/unfix? (1-5, or 'r' to roll the dice)
1
Fixed: 1
2
Fixed: 1 2
3
Fixed: 1 2 3
3
Fixed: 1 2
r
---
Dice roll #2 (out of 3): 2, 2, 5, 5, 4
Which dice do you want to fix/unfix? (1-5, or 'r' to roll the dice)
1
Fixed: 1
2
Fixed: 1 2
3
Fixed: 1 2 3
4
Fixed: 1 2 3 4
r
---
Dice roll #3 (out of 3): 2, 2, 5, 5, 3
```

Level 2: Sorting Dice

Write a method `SortDice` that sorts the dice by value. `dice1` should contain the lowest value, `dice5` the highest value. Here are some examples:

Before sort	After sort
4 3 2 6 5	2 3 4 5 6
6 3 1 6 6	1 3 6 6 6
5 5 1 2 1	1 1 2 5 5

Add the `SortDice` method to the main game logic. At the very end, call it to sort the dice by value and print their values on the screen using the `PrintDice` method.

For sorting, you can use the *Bubble Sort* algorithm. It is simple. Just swap the value of two adjecent dice if they are not in the right order. Do that until all dice are in the right order. The video *Bubble sort with Hungarian, folk dance*[2] demonstrates the principle (the sorting itself starts here[3]).

Here is an example:

- Let's look at the starting dice values `4 3 2 6 5`.
- First round:
  - We start with the first pair 4 and 3. 4 is greater than 3, so swap the values. We now have `3 4 2 6 5`.
  - The next pair is now 4 and 2. 4 is greater than 2, so swap the values. We now have `3 2 4 6 5`.
  - The next pair is now 4 and 6. 4 is not greater than 6, so don't swap the values. We still have `3 2 4 6 5`.
  - The last pair is 6 and 5. 6 is greater than 5, so swap the values. We now have `3 2 4 5 6`.
  - Are the dice already sorted? No, so repeat everything from the beginning.
- Second round:
  - We start with the first pair 3 and 2. 3 is greater than 2, so swap the values. We now have `2 3 4 5 6`.
  - All other pairs are already sorted, so no more swaps will happen.
  - Are the dice sorted now? Yes, we are done.
- The result is `2 3 4 5 6`

[2] https://youtu.be/lv3vgjM8Pv4
[3] https://youtu.be/lv3vgjM8Pv4?t=53

Level 3: Analyze Result

Write a method `AnalyzeAndPrintResult`. It assumes that the variables `dice1` to `dice5` contain values in ascending order. It determines the *hand* (e.g. *Full House*) from the dice and prints it on the screen.

Add the `AnalyzeAndPrintResult` method to the main game logic. Call it at the very end to print the hand of the player.

Level 4: Two-Player Mode

- Put the game logic in a `Play` method and run it two times, once for each player.
- Write a method `DetermineWinner` that compares the hands of both players and prints out whether we have a tie. If not, it prints out who the winner is.

This is a sample input/output for level 4:

```
PLAYER 1:
=========
Dice roll #1 (out of 3): 2, 2, 1, 6, 5
Which dice do you want to fix/unfix? (1-5, or 'r' to roll the dice)
1
Fixed: 1
2
Fixed: 1 2
3
Fixed: 1 2 3
3
Fixed: 1 2
r
---
Dice roll #2 (out of 3): 2, 2, 5, 5, 4
Which dice do you want to fix/unfix? (1-5, or 'r' to roll the dice)
1
Fixed: 1
2
Fixed: 1 2
3
Fixed: 1 2 3
4
Fixed: 1 2 3 4
r
```

```
---
Dice roll #3 (out of 3): 2, 2, 5, 5, 3
---
Two pairs!

PLAYER 2:
=========
Dice roll #1 (out of 3): 2, 2, 3, 5, 4
Which dice do you want to fix/unfix? (1-5, or 'r' to roll the dice)
2
Fixed: 2
3
Fixed: 2 3
4
Fixed: 2 3 4
5
Fixed: 2 3 4 5
r
---
Dice roll #2 (out of 3): 5, 2, 3, 5, 4
Which dice do you want to fix/unfix? (1-5, or 'r' to roll the dice)
2
Fixed: 2
3
Fixed: 2 3
4
Fixed: 2 3 4
5
Fixed: 2 3 4 5
r
---
Dice roll #3 (out of 3): 2, 2, 3, 5, 4
---
One pair!
---
Player 1 wins!
```