CSI2372

Project

Fall 2017

Playing Dice

In this year's project, you are asked to program two versions of a console game based on a simple game of rolling dice. In both versions of the game, the points from a roll are recorded on a score sheet. The goal of both versions of the game is to achieve the highest score overall. Interestingly, the players take turns rolling dice but all players can record (some of) the points for all rolls. The versions of the games are displayed by printing the score sheet for each player on the console.

Version A (Qwinto)

In this version, the players take turns rolling up to three dice but all players can record the points for all rolls. The player who rolls the dice can decide to roll one, two or three dice. The dice are colour-coded red, yellow and blue. The score sheet is organized in a sheared matrix with three rows and five overlapping columns. The rows are also colour-coded red, yellow and blue. The players can enter points only in a row if the dice of the corresponding colour was used but the score entered are the points from all dice rolled, e.g., if the yellow and blue dice was rolled, the points from the yellow and blue dice can be entered in the yellow or blue row. However, the numbers in each row must strictly increase and in each overlapping column, no number is allowed to repeat. If a player cannot score his or her own roll of the dice, it counts as a failed throw. Each failed throw will count as minus 5 points in the final score for the player. After four failed throws, the game ends. The game also ends if one player has filled two rows completely. The scoring is a mixture between the number of scores entered in each row plus a special bonus for a filled overlapping column. Completely filled rows are counted differently. Instead of counting the number of entries, the score of the right-most entry is counted.

The score sheet for this version needs 4 rows in our adaption: one row for each colour of dice: red, yellow and blue, as well as a fourth row for the failed throws. We need a maximum of twelve columns. The bonus fields are marked within % while invalid fields are marked XX. Notice that since the scores have a range form 1-18, we need a space of two characters.

An empty score sheet should look like this:

```
Player name:
              ------------------------------
Red           |  %  %  |XX|  %  %  |  |  |  |
           ---------------------------------
Yellow     |  |  |  |  |  |XX|  %  %  |  |
        ------------------------------
Blue    |  |  %  %  |XX|  |  |  |  %  %
        ------------------------------
Failed throws:
```

At the end of a game, a score sheet may look like as below:

```
Player name: Jane Doe            Points: 28
              ------------------------------
Red           | 2% 3% 6|XX| 9%11%12|13|15|16|
           ---------------------------------
Yellow     | 1| 3| 4| 5|  |XX|12%13%  |  |
        ------------------------------
Blue    | 1| 3%  % 6|XX| 7|  |  |10%  %
        ------------------------------
Failed throws: 1 2 3
```

Jane Doe scored 28 points: The red row is complete and hence the right most score is used (16 points). The yellow row has 6 entries and the blue row has 5 entries. Two bonus columns are completed for an extra of 3 plus 13 points (the scores in the corresponding fields marked with %) and three failed throws are recorded at 3 times -5 = -15.

This game is an adaption of the game Qwinto by Bernhard Lach and Uwe Rapp which was published by Gigamic, Nürnberger-Spielkarten-Verlag and White Goblin Games. You can find an English review at https://www.boardgamegeek.com/boardgame/183006/qwinto.

Version B (Qwixx)

In this version, the players take turns rolling up to six dice. The player who rolls the dice can decide to record one or two scores, the other players can decide to record none or one score. The dice are colour-coded red, yellow, green, blue and white and there is a dice of each colour but two white dice. The two white dice are available for scoring by all players while the player that rolled the dice can decide to combine one coloured dice with one of the white dice for an additional second score. The score sheet is organized in a matrix with four rows. The rows are also colour-coded red, yellow, green and blue. Looking at the rows from left to right, the red and yellow row count up from 2 to 12 while the blue and green rows count down from 12 to 2. The players can enter points in a row only from left to right. Fields

in the row can be skipped but a player cannot go back to the left out fields. The players can record the white dice in any row. The player with the current turn can score a combination of a colored and a white dice but only in the row of the colour of the dice. If a player cannot score at least one of his or her own roll of the dice, it counts as a failed throw. Each failed throw will count as minus 5 points in the final score for the player. After four failed throws, the game ends. The game also ends if two rows have been locked. A player locks a row by scoring at least five fields in a row including the last field (either a 2 or 12). If a row is locked, the dice of the corresponding colour is removed from the game. The scoring is a weighted count of the number of scores entered in each row minus the penalty for failed throws.

The score sheet for this version needs five rows in our adaption: one row for each colour of dice: red, yellow and blue, as well as a fifth row for the failed throws. We need a maximum of twelve columns. The right most column is the lock field. Notice that since the scores have a range form 2-12, we need a space of two characters.

An empty score sheet should look like this:

```
Player name:
        ------------------------------------
Red     | 2| 3| 4| 5| 6| 7| 8| 9|10|11|12| U
        ------------------------------------
Yellow  | 2| 3| 4| 5| 6| 7| 8| 9|10|11|12| U
        ------------------------------------
Green   |12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| U
        ------------------------------------
Blue    |12|11|10| 9| 8| 7| 6| 5| 4| 3| 2| U
        ------------------------------------
Failed throws:
```

At the end of a game, a score sheet may look like as below:

```
Player name: Jane Doe            Points: 26
        ------------------------------------
Red     | 2| 3| 4| 5| 6| 7| 8|XX|XX|11|XX| U
        ------------------------------------
Yellow  | 2|XX| 4| 5|XX|XX| 8| 9|10|11|12| U
        ------------------------------------
Green   |12|11|10|XX| 8|XX| 6|XX|XX| 3|XX| L
        ------------------------------------
Blue    |12|XX|10| 9| 8| 7| 6|XX| 4| 3| 2| U
        ------------------------------------
Failed throws: 1 2
```

In the above example, Jane Doe scored 28 points: The red and yellow rows have three entries which count for 6 points each, and the blue row has two entries which counts for three points. The green row has five entries and was locked by Jane Dow, which counts as six entries (5 scores plus the lock) and hence as 21 points. Two failed throws are recorded at 2 times -5 = -10.

The entire scoring table is as follows:

Entries	1	2	3	4	5	6	7	8	9	10	11	12
Points	1	3	6	10	15	21	28	36	45	55	66	78

This game is an adaption of the game Qwixx by Steffen Berndorf which was published by Nürnberger-Spielkarten-Verlag and others, for a description see
https://boardgamegeek.com/boardgame/131260/qwixx

Class Design

The design of our implementation will allow a player to choose at run-time which game version to play with 1-3 players. The main loop of the game is common between the two versions of the game. Code reuse is one of our main design goals and we will use object-oriented and generic features of C++ to achieve it. We will follow the AAA approach meaning you should use “almost always auto”. You will have to overload operators, use the standard template library and perform error checking. You will also need to implement test drivers as a main routine, e.g., for the class `QwintoScoreSheet` in the file `qwintoscoresheet.cpp` which can be activated with a compile switch `TEST_QWINTOSCORESHEET`. Your code has not only to “work” but be commented and be well written for full marks. The marks for each class are given in square brackets.

Colour [0.5]

Colour should be a scoped enumeration with the values `RED`, `YELLOW`, `GREEN`, `BLUE` and `WHITE`.

RandomDice [1]

`RandomDice` should be a helper structure with only static objects. The structure has the task to hold all objects needed to use one `std::uniform_int_distribution` for all dice ensuring that the pseudo random numbers do not start at the same value.

Dice [1.5]

Dice should be a structure storing a `const Colour` and a face as integer between 1 and 6. Dice provides the function `roll` which changes its face value randomly between 1 and 6 and return the face rolled as int. You will need to overload the insertion operator to print the `Dice` to an output stream.

RollOfDice [3]

`RollOfDice` is a simple container structure that holds multiple dice as in a roll. The size of the container should grow and shrink to the number of `Dice` in the particular roll. Provide all necessary operators and functions to enable the use of range loops over a `RollOfDice`:

```
RollOfDice rd;
for ( Dice d : rd )
```

`RollOfDice` should provide the function `roll` which simply calls roll on all `Dice` in the container. It should also provide the function `pair` that returns a `RollOfDice` with two selected `Dice` in it. A `RollOfDice` should also have a conversion operator to integer that simply adds up all the faces in the roll. You will also need to overload the insertion operator to print the `RollOfDice` to an output stream.

QwintoRow<Colour> [2.5]

A row in Qwinto holds a fixed sized array. A `QwintoRow` is templated for a certain `Colour`, i.e., a non-type template parameter. Overload the indexing or subscript operator[] to add an integer score at a certain index. You should be able to write:

```
QwintoRow<RED> row; RollOfDice rd;
row[2] = rd;
```

The subscript operator will directly work with a roll of dice because of its conversion operator to integer. The operator cannot perform any error checking. There should be a separate function validate which should perform error checking for adding a `RollofDice` to the row at a certain index and returns true if an addition is valid but should not perform the addition. You will also need to overload the insertion operator to print the `QwintoRow` to an output stream, e.g., for ScoreSheet.

QwixxRow<Class,Colour> [2.5]

For a row in Qwixx different containers may be used. A `QwixxRow` is templated for a container type of the standard template library and a certain `Colour`, i.e., it uses a type and a non-type template parameter. As `QwixxRow` are always filled from left to right and the position is determined by the sum of the faces of the dice being added, we will use the compound assignment operator+= to add a `RollOfDice` of size two. The compound assignment operator should perform error checking and throw an exception on error. You will also need to overload the insertion operator to print the `QwixxRow` to an output stream, e.g., for `ScoreSheet`.

ScoreSheet [2]

The class `ScoreSheet` is the abstract parent class for the two different score sheets in Qwixx and Qwinto. It needs to hold the name of the player, the number of failed attempts and the overall score. A score should be entered by the function `score` which accepts a `RollOfDice` and the user selected colour and position counted from the left. The position should have a default parameter of -1 which is to mean that the position info is not used when the game Qwixx is played. Score is to return a boolean

indicating if the `Dice` can be scored. The function `score` is to call the protected pure virtual function validate internally. Another function is `setTotal` which calls the pure virtual function `calcTotal`, sets and returns the points for the final score. Finally, the not operator should be virtual and to return true if the `ScoreSheet` indicates that the game has ended. You need to overload the insertion operator for the class `ScoreSheet` for printing. This global operator should behave polymorphically, even though, there is no polymorphism for global operators and functions.

QwintoScoreSheet [2]

The class `QwintoScoreSheet` is a child class of `ScoreSheet` and implements all pure virtual functions of the parent, i.e., it is a concrete class. The class is to hold three `QwintoRow` of the appropriate colours. You need to overload the insertion operator for the class `QwintoScoreSheet` for printing.

QwixxScoreSheet [2]

The class `QwixxScoreSheet` is a child class of `ScoreSheet` and implements all pure virtual functions of the parent, i.e., it is a concrete class. The class is to hold four `QwixxRow` of the appropriate colours. Construct the red and yellow row with a `std::vector` and the blue and green row with a `std::list`. You need to overload the insertion operator for the class `QwixxScoreSheet` for printing.

Player [2]

The parent class `Player` provides functions for console input and output to a player. It should hold a boolean to indicate if the this player is the active player. The input/output functions are pure virtual functions and are named `inputBeforeRoll` and `inputAfterRoll`. The functions need to accept a `RollOfDice` by reference. The functions will have to behave differently depending if the current Player is active or not.

QwintoPlayer [1.5]

The class `QwintoPlayer` holds a `QwintoScoreSheet` and implements the functions `inputBeforeRoll` and `inputAfterRoll`.

QwixxPlayer [1.5]

The class QwixxPlayer holds a `QwixxScoreSheet` and implements the function `inputBeforeRoll, inputAfterRoll`.

Main Loop Pseudo Code [4]

```
Ask player to choose game version, number of players and names of players.
Create the corresponding players and RollOfDice for the game.
while end condition is not reached
      next player takes a turn i.e., becomes active
      get input from active player before roll
      roll dice and show result
      print scoresheet of active player
      get input from active player after roll
      score dice according to input from active player
      loop over all non-active players
            print scoresheet of non-active player
            get input from non-active player
            score dice according to input
loop over all players
      calculate points for player
      print scoresheet
print overall winner
```