Project: *Rummikub*

Version 1

Deadline: April 3

(Wikimedia Commons)

The goal of the project is to simulate a simplified version of the game *Rummikub* in a console window, by the means of a C++ program. You will develop two successive versions of the program. In this first version we focus on the game mechanics.

1. The Game

Rummikub can be played among two, three or four players. Each player has a set of **tiles**, valued **1 to 13** in four different **colors**. With the tiles, the player will try to form **groups** and **runs** on the game **board**. If the player cannot place tiles on the board, then he or she will draw a new tile from the **bag**. The first player to place all of his or her tiles wins the game.

In his or her turn, each player can create new groups and runs, as well as place individual tiles in existing groups or runs. A *group* is a set of three or four tiles with the same number and different colors. A *run* is a set of tiles (from three to thirteen) in the same color and numbers in sequence (for example, 1234, 3456789 or 678).

Simplifications from the real game:

- To place tiles on the board for the first time, the players do not need that the tiles being placed add up to at least 30 points when adding their values together.
- There are no jokers.
- It is not possible for the players to reorganize existing groups or runs on the board. Individual tiles can be added to a group (from three to four tiles with no color repetition) or a run (on either extreme), but the groups or runs cannot be divided to form other groups or runs.
- The number of tiles available in each color can be a value between 8 and 13 (`NumTiles`).

Game Elements

The number of players will be a value between 2 and 4 (`NumPlayers`).

There is a total of `NumTiles` x 8 tiles, two sets per color. The four colors are: red, green, blue and yellow.

All the tiles are contained in the bag at the beginning. The initial tiles for the players, as well as those they get when they cannot place tiles, are obtained from the bag. The initial number of tiles for the players will be a value between 6 and 14 (`InitialTiles`).

Each player has a **rack** where his or her tiles are kept. The racks will have room for a maximum number of tiles (50 tiles by default).

The board is where the sets (groups and runs) are placed.

2. Implementation Details

Global constants for the game:

- `NumPlayers`: number of players (between 2 and 4).
- `NumTiles`: number of tiles available for each color set (between 8 and 13).
- `InitialTiles`: number of tiles for the players to start with (between 6 and 14).
- `MaxTiles`: maximum number of tiles a player can have (50 by default).

Data Structures

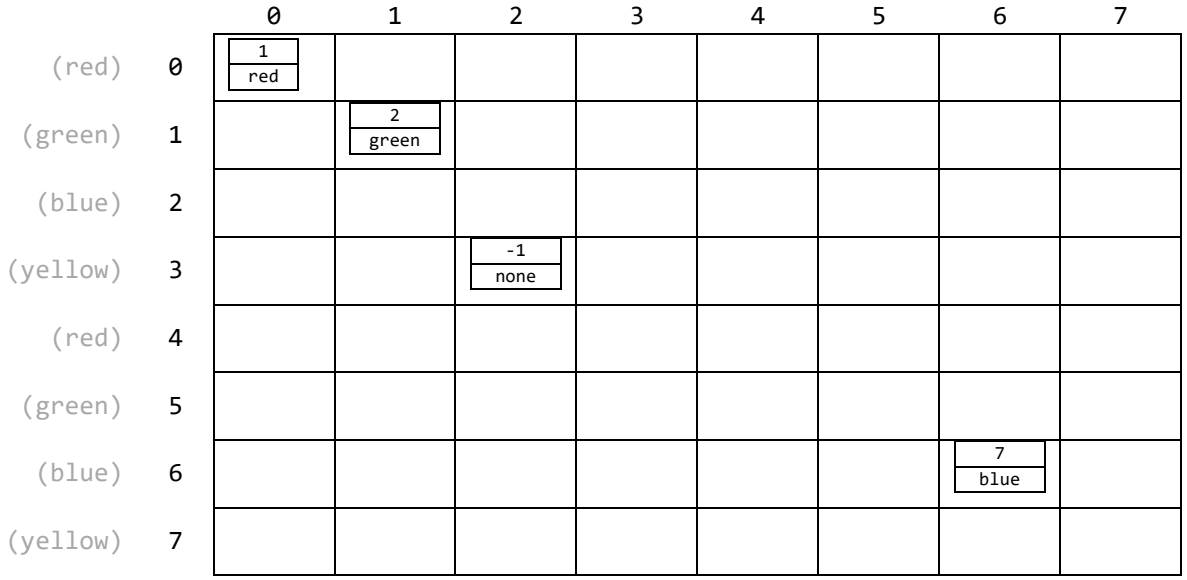
An enumerated type `tColor` for the colors will be needed, with the values for the tile colors (`red`, `green`, `blue` and `yellow`) and also `white` (to change back to the default text color) and another for tiles already taken from the bag (for example, `none`).

Each tile is identified by its number and its color. Declare a type `tTile` as a structure with an integer and a `tColor`.

The players' racks will contain lists of tiles. Declare a type `tRack` as a variable length list of `tTile`.

The players' racks will be kept in an array of `tRack`. Declare a type `tRacks` as an array of `NumPlayers` racks.

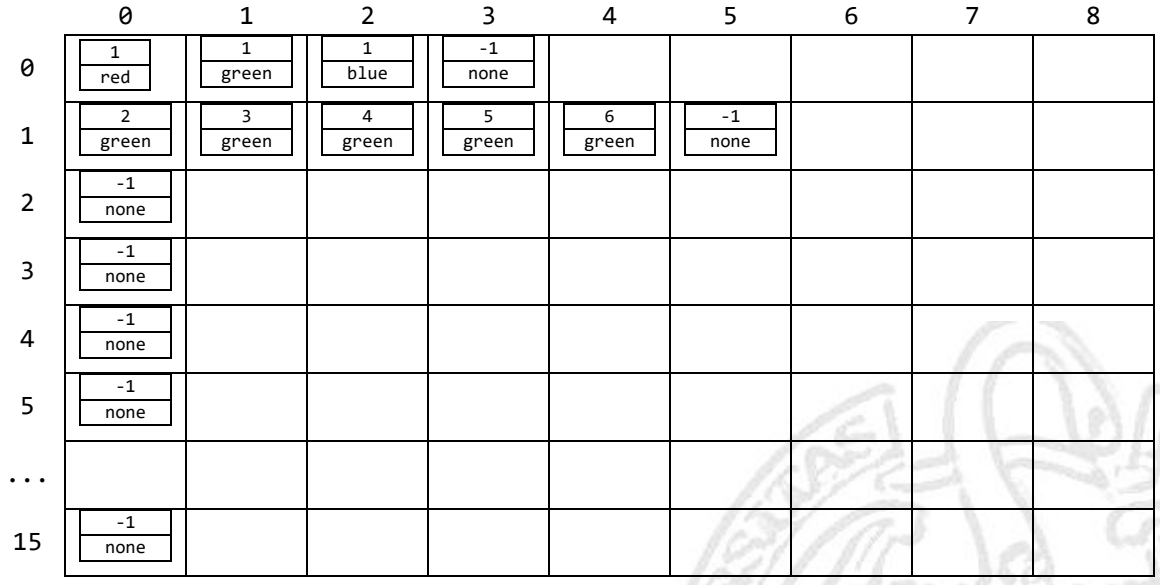
The bag will be a bi-dimensional array of 8 x `NumTiles` tiles (2 sets for each color). For example, with `NumTiles` equal to 8:



Declare the type `tBag` as a structure with a bi-dimensional array and a counter of tiles available. The tiles already drawn from the bag will change the color to `none` and the number to -1.

The board (`tBoard`) will be a variable-length list of sets (array of `tSet` and a counter). `tSet` will be an array of `NumTiles`+1 tiles in which a tile with -1 as number will be used as a sentinel.

For example, a board with two sets can be:



The maximum number of sets will be `NumTiles` x 2.

Program Behavior

After the main data structures are initialized (bag, racks and board), the player to play first will be chosen randomly. Each player, in his or her turn, tries to place tiles on the board. If the player places some tiles on the board, the turn is passed to the next player when finished, but if no tiles are placed, then the player must draw a new tile from the bag before passing the turn.

The players will be able to do the following, before passing the turn:

- Sort the tiles in the rack by number.
- Sort the tiles in the rack by color (and by number for each color).
- See the list of possible sets formed with the tiles in the rack.
- Place tiles on the board.

If the player chooses to place tiles, he or she will be able to indicate the tiles to be used to form a new group or run, as well as just one tile to be appended to one of the sets on the board. Of course, it must be confirmed that the tile sets are correct.

There may be a blocking situation when no player can place any tiles and there are no more tiles in the bag. In that case, the player with the least number of points in their rack (sum of the tiles' values) wins.

Subprograms

Implement at least the following subprograms:

- `int menu()`: displays the options for the user and returns the one chosen (valid).
- `void createBag(tBag& bag)`: initialize the `bag` of tiles.
- `tTile getTile(tBag& bag)`: returns a tile from the `bag`. Start with the tile in a row and a column chosen randomly. If the tile is available (color different than `none`), it is returned and in the `bag` the number is changed to -1 and the color to `none`. If the tile is not available, then the next available tile is searched by rows. If it is not found in the rest of that row, the search continues in the next rows, and if it is not found at the end of the bi-dimensional array, then the search continues from the beginning of the array.
- `void deal(tBag& bag, tRacks racks)`: gets `InitialTiles` tiles from the bag for each player and places them in the player's rack
- `void sortByNumber(tRack& rack)`: sorts the tiles in the `rack` by number (ascending).
- `void sortByColor(tRack& rack)`: sorts the tiles in the `rack` by color (and by number for each color).
- `bool operator==(tTile left, tTile right)`: returns `true` if the tiles have the same number and color or `false` in other cases.
- `int find(const tSet set, tTile tile)`: returns the index of the `tile` in the `set`, or -1 if the `tile` is not in the `set`.
- `void displayGroups(tRack rack)`: displays the possible groups that can be formed with the tiles in the `rack`.
- `void displayRuns(tRack rack)`: displays the possible runs that can be formed with the tiles in the `rack`.
- `void cleanSet(tSet set)`: sets all tiles in the `set` to -1 as number.
- `void addSet(tBoard& board, const tSet set)`: adds the `set` to the `board`.
- `void removeTiles(tRack& rack, const tSet set)`: removes the tiles in the `set` from the `rack`.
- `int newSet(tRack rack, tSet set)`: allows the user to create a new `set` with some tiles in his or her `rack`. Returns the number of tiles in the `set`.

- `bool placeTile(tSet set, tTile tile)`: tries to place the `tile` in the `set`; if the set is a group, the tile must be of the missing color; if the set is a run, the tile must be of the same color and maintain the sequence in one of the extremes. The function will return `true` if the tile was placed, or `false` if it was impossible to place.
- `bool play(tBoard& board, tRack& rack)`: allows the player to place tiles in the `rack` on the `board`: new groups and runs, as well as individual tiles in sets on the board.
- `int lowest(const tRacks racks)`: returns the index of the rack in the array of `racks` with the lowest sum of points (values of the remaining tiles).
- `void display(tTile tile)`: displays the `tile` (number in its color and with 2 characters, and followed by two spaces.
- `void display(tBag bag)`: displays the `bag` (to be used while debugging).
- `void display(const tSet set)`: displays the tiles in the `set` in one line.
- `void display(tBoard board)`: displays the sets on the `board`.
- `void display(tRack rack)`: displays the tiles in the `rack` in one line.
- `void displayPositions(int num)`: displays the numbers from 1 to `num` (4 characters per number) in white, to be displayed in the next line of the tiles in the rack, for the user to be able to easily select the tiles when playing.

Changing the text color

The numbers of the tiles will be displayed in the tile's color. The following procedure allows the user to change the color for the next text to be displayed. `textColor()` uses escape sequences to change the color and works in any operating system:

```
void textColor(tColor color) {
   switch (color) {
   case yellow:
      cout << "\033[1;40;33m";
      break;
   case blue:
      cout << "\033[40;34m";
      break;
   case red:
      cout << "\033[40;31m";
      break;
   case green:
      cout << "\033[40;32m";
      break;
   case white:
      cout << "\033[40;37m";
      break;
   }
}
```

Execution example

```
Turn for player 1...

Board: No sets

Rack:  3   8   4   1   8   1   3  10   1  10  10   9   1
1: Sort by number, 2: Sort by color, 3: Possible sets, 4: Place, 0: Next player >>> 1
Rack:  1   1   1   1   3   3   4   8   8   9  10  10  10
1: Sort by number, 2: Sort by color, 3: Possible sets, 4: Place, 0: Next player >>> 2
Rack:  1   1   3   4  10   8   1   8   9  10   1   3  10
1: Sort by number, 2: Sort by color, 3: Possible sets, 4: Place, 0: Next player >>> 3
 1   1   1
10  10  10
 8   9  10
Rack:  1   1   3   4  10   8   1   8   9  10   1   3  10
1: Sort by number, 2: Sort by color, 3: Possible sets, 4: Place, 0: Next player >>> 4
Rack:  1   1   3   4  10   8   1   8   9  10   1   3  10
       1   2   3   4   5   6   7   8   9  10  11  12  13
Tiles (0 at the end): 1 7 11 0
Set:  1   1   1   - Valid group!

Board:
 1:  1   1   1

Rack: 10   1   3   4  10   8   3   8   9  10
1: Sort by number, 2: Sort by color, 3: Possible sets, 4: Place, 0: Next player >>> 4
Rack: 10   1   3   4  10   8   3   8   9  10
       1   2   3   4   5   6   7   8   9  10
Tiles (0 at the end): 8 9 10 0
Set:  8   9  10   - Valid run!

Board:
 1:  1   1   1
 2:  8   9  10


Rack: 10   1   3   4  10   8   3
1: Sort by number, 2: Sort by color, 3: Possible sets, 4: Place, 0: Next player >>> 0

Turn for player 2...

Board:
 1:  1   1   1
 2:  8   9  10


Rack:  5   6   8   2   4   9   5   4   7   2   7   3   5
1: Sort by number, 2: Sort by color, 3: Possible sets, 4: Place, 0: Next player >>> 3
 5   5   5
 5   6   7
 2   3   4   5
```

```
Rack:  5   6   8   2   4   9   5   4   7   2   7   3   5
1: Sort by number, 2: Sort by color, 3: Possible sets, 4: Place, 0: Next player >>> 2
Rack:  5   6   7   2   3   4   5   7   8   2   4   5   9
1: Sort by number, 2: Sort by color, 3: Possible sets, 4: Place, 0: Next player >>> 4
Rack:  5   6   7   2   3   4   5   7   8   2   4   5   9
       1   2   3   4   5   6   7   8   9  10  11  12  13
Tiles (0 at the end): 1 7 12 0
Set:  5   5   5   - Valid group!

Board:
 1:  1   1   1
 2:  8   9  10
 3:  5   5   5


Rack:  9   6   7   2   3   4   4   7   8   2
1: Sort by number, 2: Sort by color, 3: Possible sets, 4: Place, 0: Next player >>> 4
Rack:  9   6   7   2   3   4   4   7   8   2
       1   2   3   4   5   6   7   8   9  10
Tiles (0 at the end): 4 5 6 0
Set:  2   3   4   - Valid run!

Board:
 1:  1   1   1
 2:  8   9  10
 3:  5   5   5
 4:  2   3   4


Rack:  9   6   7   2   8   7   4
1: Sort by number, 2: Sort by color, 3: Possible sets, 4: Place, 0: Next player >>> 4
Rack:  9   6   7   2   8   7   4
       1   2   3   4   5   6   7
Tiles (0 at the end): 6 0
Set:  7
Sets on the board where the tile can be placed: 2  -> Placed!

Board:
 1:  1   1   1
 2:  7   8   9  10
 3:  5   5   5
 4:  2   3   4


Rack:  9   6   7   2   8   4
1: Sort by number, 2: Sort by color, 3: Possible sets, 4: Place, 0: Next player >>> 0
```