Assignment 6: Uno

Due: 23:59, Sun 10 Dec 2023 Full marks: 100

Introduction

The objective of this assignment is to let you practice the use of inheritance and polymorphism. Usage of vectors and pointers is also included. You will write a C++ console application program to simulate the classic Uno card game.

The official game rules can be found on Mattel's products website - click the link "DKX44: UNO® - English". To implement and test this game program more easily, we ignored some tedious game rules and simplified some game rules as follows:

1. We have only one game round instead of many rounds that accumulate scores to determine the winner. We have our customized rules (to be described in next section) to determine the winner.
2. In our program, when the turn goes to a player, the player can choose to draw a card instead of playing a card. If the card drawn matches the top of the discard pile, he/she must play the card (while the official rules allow the player to keep the card).
3. For Wild Draw 4 cards, the official instruction stated that "You can only play this card when you don't have a card in your hand that matches the color of the card previously played." We will simply always allow playing this card without letting the next player challenge its legality.
4. There is no need and no way for players of our game to say "Uno" for their last card in hand.

The different types of cards with their functions and points are listed in Table 1.

Table 1: Types of Uno cards, their names, uses, symbols, number of cards, and points

Card Type	Name	Use of the Card	Symbol	Cards Per Color	Cards in Total	Points
Number Cards	Number 0-9	N/A	N/A	1	40	Face value
Action Cards	Skip	Skip the next player's turn	*S*	2	8	20
	Reverse	Reverse the turn order	*R*	2	8	20
	Draw 2	Make the next player draw 2 cards and skip their turn	*D*	2	8	20
Wild (Action) Cards	Wild Card	Change color of play	*WC*	N/A	4	50
	Wild Draw 4	Change color of play; Make the next player draw 4 cards and skip their turn	*WD*	N/A	4	50

Program Specification

This section describes the starter code, class hierarchy, some fundamental classes, the TODO classes that require your major work, and the program flow.

Copyright © 2023 CSE, CUHK

Starter Code

- To help you get started, you are provided with all the required source files. There is a total of 28 files implementing this program.
- Your task is to fill in your code for the missing parts marked with TODO comments in only 10 of these files. The last column of the Table 2 and 3 indicate whether the file has some TODO task(s) for you. Search the word "TODO:" to locate where you should fill in the missing code. Read the instructions in the TODO comment to know what is expected to do. You may remove the TODO comments after finishing the missing code.

Table 2: Header files for class definitions

	File	Description		TODO
1	`Card.h`	`Card` interface	A base class modeling a number card in Uno.	N
2	`ActionCard.h`	`ActionCard` interface	A base class modeling an action card in Uno.	N
3	`Skip.h`	`Skip` interface	A class modeling a Skip card.	N
4	`Reverse.h`	`Reverse` interface	A class modeling a Reverse card.	Y
5	`Draw2.h`	`Draw2` interface	A class modeling a Draw 2 card.	N
6	`WildCard.h`	`WildCard` interface	A class modeling a Wild card.	N
7	`WildDraw4.h`	`WildDraw4` interface	A class modeling a Wild Draw 4 card.	N
8	`Deck.h`	`Deck` interface	A base class modeling a deck of cards.	N
9	`DrawPile.h`	`DrawPile` interface	A class modeling the draw pile in an Uno game.	N
10	`DiscardPile.h`	`DiscardPile` interface	A class modeling the discard pile in an Uno game.	N
11	`Player.h`	`Player` interface	An abstract base class modeling an Uno game player.	N
12	`Bot.h`	`Bot` interface	A class modeling a computer player of Uno.	Y
13	`Man.h`	`Man` interface	A class modeling a human player of Uno.	N
14	`Uno.h`	A header file defining some global constants and structure.		N

Table 3: Source files for class implementations and client code

	File	Description	Relation with other classes	TODO
15	`Card.cpp`	`Card` implementation	Base class	N
16	`ActionCard.cpp`	`ActionCard` implementation	Subclass of `Card`	Y
17	`Skip.cpp`	`Skip` implementation	Subclass of `ActionCard`	N
18	`Reverse.cpp`	`Reverse` implementation	Subclass of `ActionCard`	Y
19	`Draw2.cpp`	`Draw2` implementation	Subclass of `Skip`	N
20	`WildCard.cpp`	`WildCard` implementation	Subclass of `ActionCard`	Y
21	`WildDraw4.cpp`	`WildDraw4` implementation	Subclass of `WildCard` and `Draw2`	N
22	`Deck.cpp`	`Deck` implementation	Base class; has a vector of `Card` pointers	Y
23	`DrawPile.cpp`	`DrawPile` implementation	Subclass of `Deck`	N
24	`DiscardPile.cpp`	`DiscardPile` implementation	Subclass of `Deck`	N
25	`Player.cpp`	`Player` implementation	Abstract base class; has a vector of `Card` pointers	Y
26	`Bot.cpp`	`Bot` implementation	Subclass of `Player`	Y
27	`Man.cpp`	`Man` implementation	Subclass of `Player`	Y
28	`unogame.cpp`	Uno game client program	The `main()` function here	Y

- We suggest the following development order.
  - Finish the base classes first: `ActionCard, Deck, and Player;`
  - Then subclasses of `ActionCard`: Reverse and WildCard;
  - Then subclasses of `Player`: `Bot` and `Man`;
  - Finally, the game client program: `unogame.cpp`.

Copyright © 2023 CSE, CUHK

Class Hierarchy

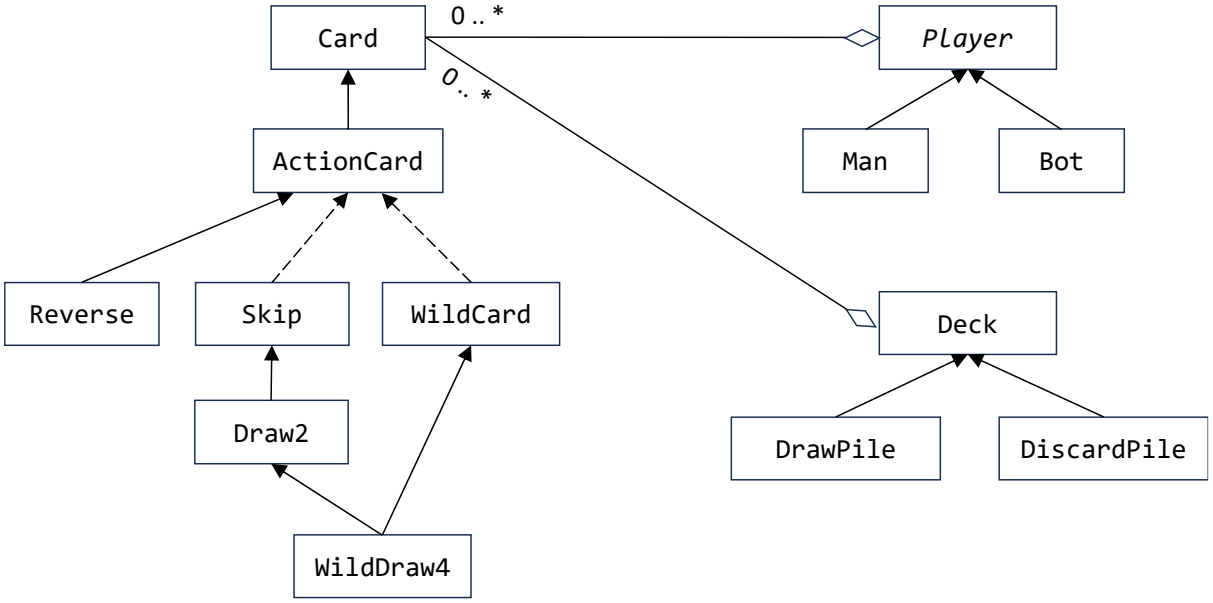

Figure 1: The class inheritance hierarchies and relationship between classes

Why is the class hierarchy designed in this way? In terms of code reuse, this design is beautiful. Subclasses can reuse the data members and member functions in their direct/indirect base classes. For example, playing Draw 2 cards need to skip the turn of the next player. This action has been implemented in a Skip card. So, making `Draw2` be a subclass of `Skip` can reuse that function. Likewise, a Wild Draw 4 need to perform three actions: (1) change the color to play for the next turn ; (2) make the next player draw 4 cards; (3) skip the next player's turn. Making `WildDraw4` be a subclass of both `WildCard` and `Draw2` (*multiple inheritance*) will allow it to reuse the functions of `WildCard`, `Draw2` and `Skip` to achieve these actions without redundant coding.

Similarly, `Man` and `Bot` are designed as subclasses of the `Player` class for reusing the parent-level data and methods. They can also *override* their parent's version of a method when they exhibit difference in behavior such as their card picking logic.

The draw pile and discard pile in the game are both decks of cards. So, we design `DrawPile` and `DiscardPile` as subclasses of the `Deck` class for code reuse. The difference is that the draw pile is filled with all cards while the discard pile is empty in the game beginning. The gameplay will keep moving cards from the draw pile to players' hands, and then to the discard pile eventually. The `Deck` class has defined a vector to store cards. They just need to reuse it. We draw cards from the vector's front for the draw pile and insert (or *stack up*) cards to the vector's back for the discard pile.

Some Fundamental Classes

Card

`Card` is the base class modeling the number cards. It has two attributes `color` and `value`.

```
private:
    Color color;
    Value value;
```

Their data types are the following *enumeration classes* respectively, which stipulate their valid values as integers, starting from zero, internally:

```
enum class Color : int { Red, Yellow, Green, Blue, Wild };
enum class Value : int { N0, N1, N2, N3, N4, N5, N6, N7, N8, N9, A = 20, W = 50 };
```

Their usage is illustrated as in the following code snippet:

```
Color c1 = Color::Yellow;  // c1 stores 1 internally
Color c2 = Color(1);  // equivalent to Color::Yellow
int index = int(Color::Blue);  // index = 3

Value v1 = Value::N3; // v1 stores 3 internally
Value v2 = Value(8);  // equivalent to Value::N8
int val1 = int(v1);   // val1 = 3
int val2 = int(v2);   // val2 = 8
```

ActionCard

`ActionCard` is the base class modeling the action cards. It inherits the two attributes `color` and `value` from the `Card` superclass, and have the following additional data member:

```
private:
    Action action;
```

The valid values for this attribute are defined in the following enum class:

```
enum class Action : int { Skip, Reverse, Draw2, Draw4, ChgColor };
```

For action cards, card matching should be based on “color or action” instead of “color or value”. This requires your implementation of its `match()` method.

For `Skip`, `Reverse` and `Draw2`, they should set their `value` attribute to `Value::A` which means 20 points in their constructors. For `WildCard` and `Draw4`, their constructors should set it to Value::W which means 50 points according to Table 1.

Deck

`Deck` is a base class modeling a collection of cards. Its constructor accepts a flag called `fill`. When creating a `DrawPile` object, it will pass `fill` as true so that all Uno cards of various types will be created and pushed into its `cards` vector (inherited from `Deck`). When creating a `DiscardPile` object, it will pass `fill` as false to make the deck empty. In the `Deck` class, the only job you need to do is to finish the body of the `print()` method, which will be used when the game is run in debug mode. You need to print the string representations (obtained via the `toString()` method in `Card` or its subclasses) and the values (obtained via the `getValue()` method in `Card`) of all cards of the deck. List at most 10 cards per line (see the beginning part of Sample Run 2).

Global Constants and the `GameState` Structure

There are two global constants and a C++ struct called `GameState` defined in the `Uno.h` header file you need to note. They are recapped as follows.

```
// Global constants
const int PASSED = -1;
const int DRAWN = -2;
```

Copyright © 2023 CSE, CUHK

```
struct GameState {
    Player** players;
    DrawPile* drawPile;
    DiscardPile* discardPile;
    bool* turnSkipped;
    int* cardsToDraw;
    int* turn;
    int* delta;
    int P;
    bool debugMode;
};
```

The PASSED constant is used when you need to signal that the player has no matching card to play, and the draw pile is empty. That means the player has no valid move and must pass the turn to the next player.

The DRAWN constant can be used when you need to signal that the player has just drawn a card from the draw pile, but the card does not match the top card of the discard pile. In this case, the turn is again given to the next player. This however does not imply the player has no valid move.

These constants can be used as return values from the `pickCard()` methods in the Man and Bot classes to tell the client code that the current player cannot play a card.

The `GameState` structure is used to ease the passing of some variables (which track the game state) from the game client side to a method in a certain class, which need to read or update the game state. For example, when a Reverse card is played, the `play()` method of the `Reverse` class can update the `delta` variable on the client side to invert the order of turns via the `delta` pointer in this structure which is passed to the `play()` method.

Player Classes

Your major work in this assignment is to implement the Man and Bot classes. They are subclasses of the *abstract* base class `Player`, which has a *pure virtual method* called `pickCard()`. Without knowing whether the player is a human or computer, it does not know how to pick a card. It depends on its subclasses to implement this method to provide the card picking logic.

Man

Man is a `Player` subclass modeling a human player. When a human player gets the turn to play, the card picking logic is as follows. To aid your understanding, below is an excerpt of Sample Run 1.

```
============================================================
Turn 9:
Discard Pile: [WD]  Current Color: Red     Draw Pile: 36
------------------------------------------------------------
Player 1 (Man):
 [0][R6]  [x][BD]  [x][GS]  [x][G9]  [x][YD]  [x][Y6]  [x][BR]  [D]raw
Enter option: 1↵
Invalid option!
Enter option: 5↵
Invalid option!
Enter option: 0↵
Discarded [R6]
============================================================
```

Copyright © 2023 CSE, CUHK

```
Turn 10:
Discard Pile: [R6]  Current Color: Red     Draw Pile: 36
-----------------------------------------------------
Player 2 (Man):
 [x][G8]  [x][Y9]  [2][R2]  [3][WD]  [x][B5]  [x][YS]  [x][G5]  [x][YR]  [D]raw
Enter option: 3↵
Discarded [WD]
Choose a color [R, Y, G, B]: X↵
Invalid option!
Choose a color [R, Y, G, B]: g↵
Color changed to Green!
```

1. The hand of cards will be shown with selectable indexes in square brackets next to them (as shown in the above sample output). A fixed width of 4 characters is used on the index part – for single digit indexes, there is a space padded before the opening square bracket, e.g. "␣[2]".
2. If the card cannot match the top of the discard pile, we show "␣[x]" next to the card, which means the card cannot be played.
3. You should print at most 10 cards per line if there are many cards in hand.
4. Show the [D]raw option if the draw pile still has cards. Two spaces are between the closing bracket of the last card option and the opening bracket of [D]raw. *(Note: no matter whether the player has playable cards in hand or not, he/she can choose to draw a card.)*
5. You may make an early function return with PASSED returned if there are no matched cards in hand and the draw pile is empty. This means the current player is forced to pass this turn.
6. Get the user input for the option (selected card index or letter 'D' for draw). Keep prompting the user until a valid option is received.
7. If the input is 'D' (case-insensitive), draw one card from the draw pile into this player's hand vector. The drawn card is added to the back of the vector.
8. Print "Drawn [XX]" where [XX] is the card drawn.
9. If the drawn card is playable, return its index.
10. Otherwise, return the constant DRAWN to tell that a card has been drawn but it is not playable. (The caller seeing DRAWN returned should skip playing any card.)
11. If the input is an integer that can index a playable card in hand, return the integer.
12. Print error message "*Invalid option!*" for invalid index or incorrect letter as input.

Bot

Bot is a Player subclass modeling a computer player. Its card playing logic has some differences from a human player. It performs the following in its version of the pickCard() method:

1. Show the hand in a hidden manner, i.e., print each card as "[Uno]", if the game was not started in debug mode.
2. Pick the **first** card in hand, from left to right, that matches the top card of the discard pile, and return its index in hand.
3. Only if there is no playable card in hand can a bot draw a card (if draw pile still has cards).
4. If the game is in debug mode, print "Drawn [XX]" where [XX] is the card drawn.
5. If the card drawn matches the discard pile's top card, return the drawn card's index (the last element in hand).
6. Otherwise, return the constant DRAWN.

Copyright © 2023 CSE, CUHK

ActionCard Subclasses

There are some specific subtypes of action cards you need to handle.

WildCard

`WildCard` is a subclass of `ActionCard` used to change the current color to another among the choices of Red, Yellow, Green and Blue based on the selection by the player playing this card. When a human player plays this card, the `play()` method of this class will show the prompt:

```
Choose a color [R, Y, G, B]:
```

to get the letter representing the selected color. Conversion of this letter to the corresponding Color enum value is necessary here. You need to show the error message *"Invalid option!"* if the user input is not one of the 4 letters.

If it is a bot playing this card, it will choose the most frequent color in the hand (i.e., most cards in the hand are in that color) to be the target color. This is done by calling the `mostFrequentColor()` method of the parent `Player` class. That method requires your implementation too. Read the TODO comments of `Player.cpp` to know what to do.

It will set the color of this card to the selected color.

Reverse

`Reverse` is a subclass of `ActionCard` used to reverse the turn order. If the turn order is clockwise, playing this card will make it become anticlockwise, and vice versa. To achieve this effect, you only need to invert the sign of the delta variable defined on the game client side via the delta pointer of the `GameState` parameter to this card's `play()` method. In particular, for 2-player games, playing a reverse card means the same as playing a skip card. In this case, you should set the `turnSkipped` flag on the game client side to true too.

Program Flow

The program flow of the game (`unogame.cpp`) is described as follows.

1. The program starts with prompting the user to enter a seed value for seeding the pseudo-random number generator, Y/N (case-insensitive) for playing in debug mode, and Y/N (case-insensitive), the maximum number of turns to play, and the number of computer and human players (within 2 to 5 players in total for simplicity). *(Done for you!)*
2. Create an array of players that point to `Bot` and `Man` objects being created. *(Done for you!)*
3. Define variables (such as `turn` and `delta`) that track the game state and initialize the GameState struct. *(Done for you!)*
4. Shuffle the deck and deal *H* cards (*H* = 7) to each player. *(Done for you!)*
5. Draw the first card onto discard pile. *(Done for you!)*
6. Start the game loop which performs the following:
   a. Print the "turn header" which shows discard pile's top card, current color, and current size of draw pile.
   b. Use the turn integer to index the players array to get a pointer to the current player.
   c. Print the name of the current player.
   d. If `cardsToDraw` > 0, current player draws the required # cards.

Copyright © 2023 CSE, CUHK

   e. If `turnSkipped` is true, current player skips picking and playing a card in this turn.
   f. Otherwise, call the `pickCard()` method to get the index of a selected card in hand.
   g. Then call the `playCard()` method with the obtained index if it is not PASSED and not DRAWN.
   h. Check game over condition. Exit the game loop if: (1) current player's hand has no cards; or (2) all players consecutively passed their turns (i.e., no one can play a card or draw); or (3) the maximum turns to play set by the user has been reached.
   i. Reset `cardsToDraw` and `turnSkipped` for clean state for next turn.
   j. Update the `turn` integer to let the next player become current.
7. Print the game over message.
8. List all players' hands and the sum of points of all cards in hand.
9. Print the name of the winner.

Deciding the Gamer Winner

The player who is the first to empty his/her hand is the winner in our game program. Our game beginning will also prompt the user to set a maximum number of turns to play. Within the specified maximum turns, if all players still have cards in their hands, then the winner will be the one who has the least sum of points of all cards in his/her hand. Players are labelled as Player 1, Player 2, Player 3, … and indexed in an array. In case that more than one players score the same lowest points, the first in the array of players will be the winner.

Note: to compute the total points of a hand, you need to implement the `handPoints()` method of the `Player` class to sum the `value` of each card in hand.

Some Points to Note

- You *cannot declare any global variables* in *all* your source files (except `const` ones).
- You can write extra functions in any source files if necessary. However, extra *member* functions (instance methods), no matter `private` or `public`, are *not* allowed.
- You should not change the class hierarchy or function signatures of all our provided classes.
- Your program is scalable to hand size *H* (i.e., the number of cards initially dealt to each player), which can be changed at compile time (not hardcoded).
- You may observe from the program flow that the game client code performs the gameplay in a polymorphic manner: looping through an array of supertype Player pointers and call the `pickCard()` and `playCard()` methods on each player without knowing (or explicitly detecting) whether the player is a man or a bot. This allows thin client code and extensibility with changing the client code. When you implement the client, remember not to spoil this good principle.

Reminders on Testing

- You may assume all user inputs won't have errors on the value's type and number of values required, e.g., if the expected input is an integer, the user will not enter an alphabet.
- Note the **difference in random number sequences generated on Windows and on macOS** due to different library implementations even for the same seed. The sample runs given below were conducted on macOS. For Windows users, please use the sample program for Windows to perform your correctness checks.

Copyright © 2023 CSE, CUHK

Sample Runs

In the following sample runs, the blue text is user input and the other text is the program printout. You can try the provided sample program for other input. *Your program output should be exactly the same as what the sample program produces* (same text, symbols, letter case, spacings, etc.). Note that *there is a space after the ':' symbol in the user prompt text*. *The final line ends with a newline*.

Sample Run 1

(2 men vs. 2 bots playing in non-debug mode)

```
Enter seed (or hit Enter): ↲
Seed: 123
Play in debug mode (Y/N)? n↲
Debug mode: false
Max. turns to play: ↲
Max. turns: 100
Enter # man and bot players: 2 2↲
#Men: 2; #Bots: 2
Player 1 (Man) drawn 7 card(s).
Player 2 (Man) drawn 7 card(s).
Player 3 (Bot) drawn 7 card(s).
Player 4 (Bot) drawn 7 card(s).
============================================================
Turn 1:
Discard Pile: [G1]  Current Color: Green   Draw Pile: 43
------------------------------------------------------------
Player 1 (Man):
 [0][GD]  [x][R6]  [x][BD]  [3][GS]  [4][G9]  [x][YD]  [x][Y6]  [D]raw
Enter option: 0↲
Discarded [GD]
============================================================
Turn 2:
Discard Pile: [GD]  Current Color: Green   Draw Pile: 43
------------------------------------------------------------
Player 2 (Man):
Player 2 (Man) drawn 2 card(s).
Turn skipped!
============================================================
Turn 3:
Discard Pile: [GD]  Current Color: Green   Draw Pile: 41
------------------------------------------------------------
Player 3 (Bot):
[Uno] [Uno] [Uno] [Uno] [Uno] [Uno] [Uno]
Discarded [G7]
============================================================
Turn 4:
Discard Pile: [G7]  Current Color: Green   Draw Pile: 41
------------------------------------------------------------
Player 4 (Bot):
[Uno] [Uno] [Uno] [Uno] [Uno] [Uno] [Uno]
Discarded [WC]
Color changed to Red!
============================================================
Turn 5:
Discard Pile: [WC]  Current Color: Red     Draw Pile: 41
------------------------------------------------------------
Player 1 (Man):
 [0][R6]  [x][BD]  [x][GS]  [x][G9]  [x][YD]  [x][Y6]  [D]raw
```

Copyright © 2023 CSE, CUHK

```
Enter option: D↵
Player 1 (Man) drawn 1 card(s).
Drawn [BR]
============================================================
Turn 6:
Discard Pile: [WC]  Current Color: Red     Draw Pile: 40
------------------------------------------------------------
Player 2 (Man):
 [x][G8]  [x][Y9]  [2][R2]  [3][WD]  [x][B5]  [x][YS]  [x][G5]  [x][YR]  [8][WC]
[D]raw
Enter option: 8↵
Discarded [WC]
Choose a color [R, Y, G, B]: Y↵
Color changed to Yellow!
============================================================
Turn 7:
Discard Pile: [WC]  Current Color: Yellow  Draw Pile: 40
------------------------------------------------------------
Player 3 (Bot):
[Uno] [Uno] [Uno] [Uno] [Uno] [Uno]
Discarded [WD]
Color changed to Red!
============================================================
Turn 8:
Discard Pile: [WD]  Current Color: Red     Draw Pile: 40
------------------------------------------------------------
Player 4 (Bot):
Player 4 (Bot) drawn 4 card(s).
Turn skipped!
============================================================
Turn 9:
Discard Pile: [WD]  Current Color: Red     Draw Pile: 36
------------------------------------------------------------
Player 1 (Man):
 [0][R6]  [x][BD]  [x][GS]  [x][G9]  [x][YD]  [x][Y6]  [x][BR]  [D]raw
Enter option: 1↵
Invalid option!
Enter option: 5↵
Invalid option!
Enter option: 0↵
Discarded [R6]
============================================================
Turn 10:
Discard Pile: [R6]  Current Color: Red     Draw Pile: 36
------------------------------------------------------------
Player 2 (Man):
 [x][G8]  [x][Y9]  [2][R2]  [3][WD]  [x][B5]  [x][YS]  [x][G5]  [x][YR]  [D]raw
Enter option: 3↵
Discarded [WD]
Choose a color [R, Y, G, B]: X↵
Invalid option!
Choose a color [R, Y, G, B]: g↵
Color changed to Green!
============================================================
Turn 11:
Discard Pile: [WD]  Current Color: Green   Draw Pile: 36
------------------------------------------------------------
Player 3 (Bot):
Player 3 (Bot) drawn 4 card(s).
```

Copyright © 2023 CSE, CUHK

```
Turn skipped!

... (too long, skipped)

============================================================
Turn 75:
Discard Pile: [R8]  Current Color: Red     Draw Pile: 6
------------------------------------------------------------
Player 1 (Man):
 [x][BD]  [x][BS]  [D]raw
Enter option: d↵
Player 1 (Man) drawn 1 card(s).
Drawn [B7]
============================================================
Turn 76:
Discard Pile: [R8]  Current Color: Red     Draw Pile: 5
------------------------------------------------------------
Player 4 (Bot):
[Uno]
Discarded [RD]
**********
Game Over!
**********
Player 1 (Man) owes   47 point(s): [BD] [BS] [B7]
Player 2 (Man) owes    9 point(s): [Y0] [B9]
Player 3 (Bot) owes  109 point(s): [R1] [B2] [Y8] [B0] [R5] [RS] [Y3] [BR] [WC]
Player 4 (Bot) owes    0 point(s):
The winner is Player 4 (Bot)!
```

Sample Run 2

(3 bots playing in debug mode; no human player)

```
Enter seed (or hit Enter): 987654321↵
Seed: 987654321
Play in debug mode (Y/N)? y↵
Debug mode: true
Max. turns to play: ↵
Max. turns: 100
Enter # man and bot players: 0 3↵
#Men: 0; #Bots: 3
Cards created:
[R0]=0  [R1]=1  [R2]=2  [R3]=3  [R4]=4  [R5]=5  [R6]=6  [R7]=7  [R8]=8  [R9]=9
[RR]=20 [RS]=20 [RD]=20 [RR]=20 [RS]=20 [RD]=20 [Y0]=0  [Y1]=1  [Y2]=2  [Y3]=3
[Y4]=4  [Y5]=5  [Y6]=6  [Y7]=7  [Y8]=8  [Y9]=9  [YR]=20 [YS]=20 [YD]=20 [YR]=20
[YS]=20 [YD]=20 [G0]=0  [G1]=1  [G2]=2  [G3]=3  [G4]=4  [G5]=5  [G6]=6  [G7]=7
[G8]=8  [G9]=9  [GR]=20 [GS]=20 [GD]=20 [GR]=20 [GS]=20 [GD]=20 [B0]=0  [B1]=1
[B2]=2  [B3]=3  [B4]=4  [B5]=5  [B6]=6  [B7]=7  [B8]=8  [B9]=9  [BR]=20 [BS]=20
[BD]=20 [BR]=20 [BS]=20 [BD]=20 [WC]=50 [WD]=50 [WC]=50 [WD]=50 [WC]=50 [WD]=50
[WC]=50 [WD]=50
Player 1 (Bot) drawn 7 card(s).
Player 2 (Bot) drawn 7 card(s).
Player 3 (Bot) drawn 7 card(s).
Draw pile after shuffling and dealing:
[Y2]=2  [WC]=50 [G0]=0  [RR]=20 [R4]=4  [Y4]=4  [G1]=1  [YR]=20 [WC]=50 [BR]=20
[G6]=6  [G3]=3  [GD]=20 [R3]=3  [B6]=6  [Y5]=5  [R7]=7  [YR]=20 [Y0]=0  [GR]=20
[B3]=3  [Y1]=1  [RS]=20 [WD]=50 [G2]=2  [G4]=4  [R1]=1  [BD]=20 [GR]=20 [BS]=20
[R9]=9  [B2]=2  [B9]=9  [Y7]=7  [RD]=20 [Y8]=8  [B0]=0  [R5]=5  [WD]=50 [RS]=20
[Y3]=3  [BR]=20 [WC]=50 [R8]=8  [B7]=7  [B8]=8  [B4]=4  [B1]=1  [YD]=20 [WC]=50
============================================================
```

Copyright © 2023 CSE, CUHK

```
Turn 1:
Discard Pile: [BS]  Current Color: Blue    Draw Pile: 50
------------------------------------------------------
Player 1 (Bot):
[GD] [R6] [BD] [GS] [G9] [YD] [Y6]
Discarded [BD]
======================================================
Turn 2:
Discard Pile: [BD]  Current Color: Blue    Draw Pile: 50
------------------------------------------------------
Player 2 (Bot):
Player 2 (Bot) drawn 2 card(s).
Turn skipped!
======================================================
Turn 3:
Discard Pile: [BD]  Current Color: Blue    Draw Pile: 48
------------------------------------------------------
Player 3 (Bot):
[G7] [GS] [R0] [RD] [WD] [RR] [YS]
Discarded [RD]
======================================================
Turn 4:
Discard Pile: [RD]  Current Color: Red     Draw Pile: 48
------------------------------------------------------
Player 1 (Bot):
Player 1 (Bot) drawn 2 card(s).
Turn skipped!
======================================================
Turn 5:
Discard Pile: [RD]  Current Color: Red     Draw Pile: 46
------------------------------------------------------
Player 2 (Bot):
[G8] [Y9] [R2] [WD] [B5] [YS] [G5] [Y2] [WC]
Discarded [R2]
======================================================
Turn 6:
Discard Pile: [R2]  Current Color: Red     Draw Pile: 46
------------------------------------------------------
Player 3 (Bot):
[G7] [GS] [R0] [WD] [RR] [YS]
Discarded [R0]
======================================================
Turn 7:
Discard Pile: [R0]  Current Color: Red     Draw Pile: 46
------------------------------------------------------
Player 1 (Bot):
[GD] [R6] [GS] [G9] [YD] [Y6] [G0] [RR]
Discarded [R6]
======================================================
Turn 8:
Discard Pile: [R6]  Current Color: Red     Draw Pile: 46
------------------------------------------------------
Player 2 (Bot):
[G8] [Y9] [WD] [B5] [YS] [G5] [Y2] [WC]
Discarded [WD]
Color changed to Yellow!
======================================================
Turn 9:
Discard Pile: [WD]  Current Color: Yellow  Draw Pile: 46
```

Copyright © 2023 CSE, CUHK

```
------------------------------------------------------
Player 3 (Bot):
Player 3 (Bot) drawn 4 card(s).
Turn skipped!
======================================================
Turn 10:
Discard Pile: [WD]  Current Color: Yellow  Draw Pile: 42
------------------------------------------------------
Player 1 (Bot):
[GD] [GS] [G9] [YD] [Y6] [G0] [RR]
Discarded [YD]
======================================================
Turn 11:
Discard Pile: [YD]  Current Color: Yellow  Draw Pile: 42
------------------------------------------------------
Player 2 (Bot):
Player 2 (Bot) drawn 2 card(s).
Turn skipped!
======================================================
Turn 12:
Discard Pile: [YD]  Current Color: Yellow  Draw Pile: 40
------------------------------------------------------
Player 3 (Bot):
[G7] [GS] [WD] [RR] [YS] [R4] [Y4] [G1] [YR]
Discarded [WD]
Color changed to Yellow!


... (too long, skipped)

======================================================
Turn 97:
Discard Pile: [B0]  Current Color: Blue    Draw Pile: 0
------------------------------------------------------
Player 3 (Bot):
[R8]
Turn passed!
======================================================
Turn 98:
Discard Pile: [B0]  Current Color: Blue    Draw Pile: 0
------------------------------------------------------
Player 1 (Bot):
[RD] [R5] [B4]
Discarded [B4]
======================================================
Turn 99:
Discard Pile: [B4]  Current Color: Blue    Draw Pile: 0
------------------------------------------------------
Player 2 (Bot):
[RS] [YD] [WC]
Discarded [WC]
Color changed to Red!
======================================================
Turn 100:
Discard Pile: [WC]  Current Color: Red     Draw Pile: 0
------------------------------------------------------
Player 3 (Bot):
[R8]
Discarded [R8]
```

Copyright © 2023 CSE, CUHK

```
**********
Game Over!
**********
Player 1 (Bot) owes   25 point(s): [RD] [R5]
Player 2 (Bot) owes   40 point(s): [RS] [YD]
Player 3 (Bot) owes    0 point(s):
The winner is Player 3 (Bot)!
```

Submission and Marking

- Submit a zip of all the 28 files to Blackboard (https://blackboard.cuhk.edu.hk/).
- Insert *your name*, *student ID*, and *e-mail* as comments at the beginning of your source file.
- You can submit your assignment multiple times. Only the latest submission counts.
- Your program should be *free of compilation errors and warnings*.
- Your program should *include suitable comments as documentation*.
- ***Do NOT share your work to others*** and ***do NOT plagiarize.*** Both senders and plagiarizers shall be penalized.

Copyright © 2023 CSE, CUHK