IMS
Institut für Mikroelektronische Systeme
Leibniz Universität Hannover

Programmierpraktikum Technische Informatik (C++)
Aufgabe 07

Hinweise
Abgabe: Stand des Git-Repositories am 21.6.2022 um 9 Uhr.
Die Dateien zur Bearbeitung dieser Aufgabe erhalten Sie, indem Sie die neue Aufgabe aus dem Aufgabenrepository in Ihr lokales mergen. Dies geschieht mit `git pull common main` innerhalb Ihres Repositories. Die Lösungen committen Sie bitte in Ihr lokales Repository und pushen sie in Ihr Repository auf den Gitlab-Server.

Teilaufgabe 1 (2 Punkte)

Im Verzeichnis `blackjack` soll ein Programm vervollständigt werden, welches das in Casinos meistgespielte Karten-Glücksspiel Black Jack (Siebzehn und vier) umsetzt. Ein Aufruf des Programms ohne Parameter soll interaktive Partien eines Spielers (Mensch) gegen den Dealer (Computer) unter Verwendung eines unbegrenzten Decks ermöglichen. Wird das Programm mit einer Zahl n als Parameter aufgerufen, so simuliert es n Partien des Spiels an einem vollständig besetzen Spieltisch mit 7 Spielern (Computer) gegen den Dealer (Computer) unter Verwendung eines begrenzten Decks, bestehend aus 6 Kartenpaketen, und gibt die Ergebnisse jeder Runde auf der Konsole aus.

Für die Bearbeitung der Aufgaben ist kein Vorwissen über die Spielregeln vonnöten. Falls weiteres Interesse oder Unklarheiten bestehen, können die Regeln unter `http://de.wikipedia.org/wiki/Black_Jack` nachgelesen werden. Dabei können Sie die Unterabschnitte Siebener-Drilling, Insurance, Split und Double ignorieren, weil diese unübliche Regeln oder Nebenwetten behandeln, die nicht umgesetzt werden sollen.

a) Ein Kartenpaket enthält 52 Karten. Jeweils 4 der Zahlenkarten von 2 bis 10 und jeweils 4 der Bildkarten Bube, Dame, König und Ass. Die Spielkarten werden nicht als Objekte behandelt, sondern durch ihren Wert im Spiel repräsentiert.
Zahlenkarten haben den Wert der abgebildeten Zahl, das Ass hat den Wert 11 und die übrigen Bildkarten den Wert 10.
Die von der Klasse `Deck` abgeleitete Klasse `InfiniteDeck` stellt ein Deck dar, bei dem die Wahrscheinlichkeit, eine bestimmte Karte zu ziehen, von den bereits gezogenen Karten unabhängig ist. Bei der Klasse `LimitedDeck` ist sie von vorher gezogenen Karten abhängig.

Appelstraße 4 | 30167 Hannover
fon 0511 762-19691 | fax 0511 762-19694
www.ims.uni-hannover.de | info@ims.uni-hannover.de

Programmieren Sie in `blackJack/src/deck.cpp` den Konstruktor der Klasse `LimitedDeck`! Dieser soll das erstellte Deck mit den Spielkarten für die übergebene Anzahl Kartenpakete befüllen und die Karten **zufällig** mischen.

Hinweise:

- Die bereits vorhandene Methode `Deck::fill` könnte sich als hilfreich erweisen.
- Zum Mischen der Karten ist die Funktion std::shuffle (s. `http://www.cplusplus.com/reference/algorithm/shuffle`) in Verbindung mit der bereits vorhandenen Random Engine `Deck::re` hilfreich.

b) Ergänzen Sie die Methode `LimitedDeck::getRandomCard`!
Diese Methode implementiert das Ziehen einer Karte aus dem Deck. Hierfür soll die hinterste Karte des Vektors `cards` aus diesem entfernt und zurückgegeben werden.

Hinweis: Für die nächste Teilaufgabe kann es sinnvoll sein, entfernte Karten in einer für diesen Zweck hinzugefügten Membervariable zu speichern.

c) Implementieren Sie die Methode `LimitedDeck::resetCards`!
Diese repräsentiert die Vorbereitung einer neuen Runde unter Verwendung einer automatischen Mischmaschine, wie sie in den meisten Casinos zum Einsatz kommt. Sie soll die durch `getRandomCard` entfernten Karten wieder in den Vektor `cards` einfügen und anschließend das Deck neu mischen.

d) Warum ist es – abgesehen von einem höheren Grad an Realismus – sinnvoll, den `std::vector` vollständig zu durchmischen und Karten vom Ende zu entnehmen, anstatt das Mischen zu sparen und Karten an zufälligen Indizes zu entnehmen? Schreiben Sie Ihre Begründung bitte in eine separate Datei.

Teilaufgabe 2 (1.5 Punkte)

Vervollständigen Sie die Implementation der Klasse `Player`! Dafür sind die folgenden Aufgaben zu erledigen:

a) Sehen Sie sich in der `blackJack/src/player.h` die Klasse `Player` an und implementieren Sie in der `blackJack/src/player.cpp` die Methode `Player::getHandValue`, die die Summe der Werte aller Karten des Spielers (gespeichert im Datenmember `cards`) zurückgeben soll!

b) Implementieren Sie die Methode `Player::hasBlackjack`! Diese Methode soll zurückgeben ob der Spieler einen Black Jack hat. Gemäß der Regeln ist dies der Fall, wenn der Spieler nur 2 Karten und eine Kartensumme von 21 hat.

c) Implementieren Sie die Methode `Player::isBust`, die zurückgibt, ob der Spieler sich

überkauft hat, also eine Kartensumme von über 21 hat!

d) Die Regeln sehen vor, dass Asse entweder mit einem Wert von 11 oder einem Wert von 1 zählen können, wobei der für den jeweiligen Spieler günstigere Wert verwendet wird. Im Programm werden Asse zunächst als 11 gezählt und müssen dementsprechend in Fällen, wo dies vorteilhaft ist, auf 1 abgewertet werden. Vervollständigen Sie dafür die Methode `Player::devaluateAce`! Diese Methode soll bei jedem Aufruf den Wert **eines** Asses im Vektor `cards` von 11 in 1 ändern. Es soll `true` zurückgegeben werden, falls ein Ass gefunden wurde, das abgewertet werden konnte, und ansonsten `false`.

e) Implementieren Sie die Methode `Player::takeHit`! Diese soll zunächst eine Karte aus dem Deck ziehen und der Hand des Spielers hinzufügen. Anschließend soll überprüft werden, ob der Spieler sich überkauft hat und, wenn ja, ob er sich durch die Abwertung möglicher Asse retten kann. Dabei sollen so lange Asse abgewertet werden, bis keine mehr übrig sind oder der Spieler nicht mehr überkauft ist.

Teilaufgabe 3 (1.5 Punkte)

Erstellen Sie in der `blackJack/src/player.h` die beiden von `Player` abgeleiteten konkreten Klassen `AutomatedPlayer` und `ManualPlayer`!

a) Erstellen Sie eine von `Player` abgeleitete konkrete Klasse `AutomatedPlayer`! Dafür müssen Sie die rein virtuelle Methode `evaluateHand` überschreiben. Diese Methode soll `true` zurückgeben, wenn der Spieler in der aktuellen Situation eine weitere Karte zieht. Verwenden Sie dafür die auf Wikipedia beschriebene Regel „Dealer must stand on 17 and must draw to 16“. Somit soll der `AutomatedPlayer` weitere Karten ziehen, wenn seine aktuelle Kartensumme kleiner als 17 ist.

b) Erstellen Sie eine von `Player` abgeleitete konkrete Klasse `ManualPlayer`! Für diesen `Player` soll die `evaluateHand` Methode die aktuelle Hand des Spielers auf der Konsole ausgeben und die Entscheidung, ob eine weitere Karte gezogen wird, per Konsoleneingabe dem Benutzer überlassen.

c) Vervollständigen Sie die Methode `Player::create`! Dabei handelt es sich um eine „`factory method`“ (s. `https://de.wikipedia.org/wiki/Fabrikmethode`).
Diese soll anhand des übergebenen `PlayerType` einen `std::unique_ptr` auf neues Objekt einer der von `Player` erbenden Klassen zurückgeben. Behandeln Sie auch den möglichen Fall, dass der übergebene `PlayerType` keiner der gültigen Werte des Enums ist.

Hinweis: Der `PlayerType` ist durch die `enum class PlayerType` in der `blackJack/src/player.h` definiert.