T1 Programmierung

Prof. Dr. rer. nat. Alexander Auch

Programmentwurf ON23A Gruppe 1

Roulette

Gruppenmitglieder
Tobias Wawak (7806387)
Jonas Bauer (2168965)
Julian Köhnlein (7561542)
Pia Kühnle (9395955)
Elia Küstner (2018795)

Inhaltsangabe

1. Spielidee

1.1 Einleitung (historischer Kontext)

Das Roulette-Glücksspiel hatte seinen Ursprung in Frankreich im 18. Jahrhundert in Paris. Der Name “Roulette” ist französisch und bedeutet “kleines Rad”. Das Spiel verbreitete sich schnell in ganz Europa und erlangte anschließend im 19. Jahrhundert auch Popularität in den USA, wodurch viele neue Varianten des Spiels entstanden. Roulette ist bis heute ein sehr populäres Spiel in der Gesellschaft weltweit.

1.2 Spielprinzip

Das Herzstück des Roulettes ist das rotierende Rad, das in nummerierte und farbige Taschen unterteilt ist. Die Nummerierungen reichen von 0 bis 36. Die Felder sind nach einem komplexen System angeordnet, das gewährleisten soll, dass die Chancen, auf die gewettet werden kann, möglichst gleichmäßig angeordnet sind. Auf dem Roulette Board sind die Felder abwechselnd in rot und schwarz gefärbt, eine Ausnahme bietet hier das Feld 0, welches in den meisten Roulette-Varianten grün gefärbt ist.

Das Spiel beginnt mit einer Phase des Wettens, in der der Spieler Geldbeträge auf mögliche Spielausgänge setzt. Im physischen Casino setzt der Spieler bei diesem Vorgang Chips auf ein sogenanntes Roulette Tableau (Abbildung 1). Hier gibt es viele Möglichkeiten, auf die der Spieler setzen kann, die je nach Version des Spiels variieren. Aufgrund der Komplexität von Roulette haben wir uns bei der Spiel-Implementierung auf die beiden Möglichkeiten des Setzens auf einzelne Zahlen und Farben beschränkt. Bei der Umsetzung des Spiels haben wir zudem darauf Wert gelegt, das Drehen des Rads visuell in der Konsole aufzugreifen.

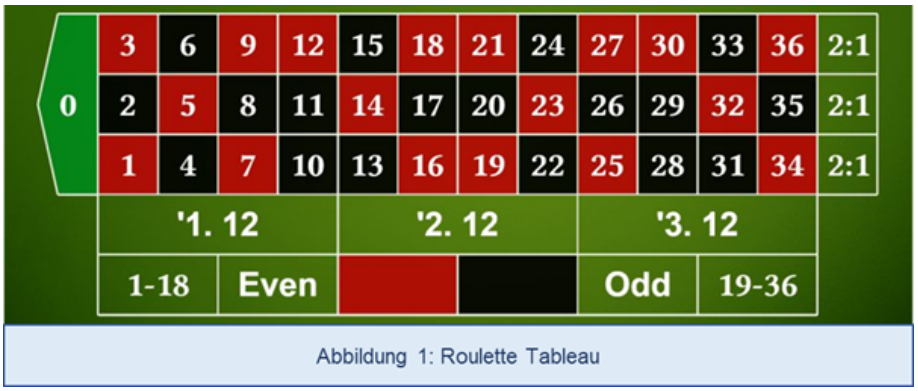

Abbildung 1: Roulette Tableau

Tabelle: Setzmöglichkeiten

Setzmöglichkeiten	Gewinnwahrscheinlichkeit	Gewinn
Farben (Rot / Schwarz / Grün)		
Rot / Schwarz	48,6%	1:1
Grün	2,7%	35:1
einzelne Zahlen		
Zahl	2,7%	35:1

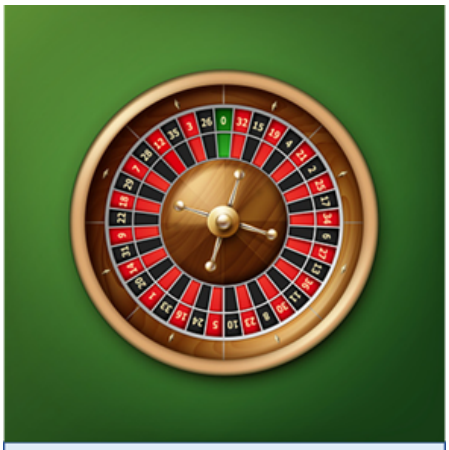

Abbildung 2: Roulette Rad

Nachdem die Spieler ihre Beträge gesetzt haben, wird das Rad (Abbildung 2) vom Croupier in Rotation versetzt und eine kleine Kugel gegen die Laufrichtung geworfen. Das Feld, auf dem die Kugel zum Stoppen kommt, bestimmt über die Gewinne und Verluste der einzelnen Spieler. Die Gewinne werden anschließend ausgezahlt.
Durch die vielen Wettmöglichkeiten können die Wetten je nach Risikobereitschaft und Budget angepasst werden. Durch das einfache Spielkonzept eignet sich Roulette gleichermaßen für Anfänger und erfahrene Spieler. Durch sein charakteristisches Rad und die Spannung bei jedem Spin hat es seinen festen Platz in der Welt der Glücksspiele.

2. Klasse Roulette

2.1 Variable

Zuallererst wird in der Klasse Roulette eine Boolean Variable mit dem Namen “play” initialisiert und mit dem Wert true gefüllt. Diese Variable sagt aus, ob das Spiel gerade läuft oder nicht.

2.2 Arrays

In der Klasse werden die zwei Arrays: “gameSelection” und “colourSelection” definiert. Das Array “gameSelection” enthält die Werte Zahl und Farbe, also die Auswahl des Spiels, die später zum Einsatz kommt. Im Array Farben sind die drei Farben gespeichert, die auf einem Roulette-Board vertreten sind, also grün, schwarz und rot.

2.3 Hashmaps

In der Roulette Klasse werden zwei Hashmaps definiert, die Hashmaps wheel und colourDots. In colourDots wird jeder Farbe ein Farbpunkt zugeordnet, was später bei der Darstellung des Roulette Rads benötigt wird. Die Hashmap “wheel” ordnet jeder Zahl, die sich auf dem Roulette Rad befindet, die dazugehörige Farbe zu.

3. Methoden

3.1 allgemeine Informationen

Um Redundanzen im Code möglichst zu vermeiden, haben wir Bestandteile / Elemente des Codes, die sich häufig wiederholen, in Methoden ausgelagert. Das Beheben von Bugs, die Verständlichkeit des Codes für die anderen Teammitgliedern und die Länge des Codes profitieren von dieser ergriffenen Maßnahme.

3.2 main ()

Die main-Methode hat die Signatur **public static void main(String[] args) throws InterruptedException.** Diese Methode begrüßt zuerst den Spieler mit einer Willkommensnachricht auf der Konsole. Anschließend wird die Methode **addToAccount()** aufgerufen, welche dem Nutzer die Möglichkeit gibt sein Konto mit einem Betrag zwischen 1-1000 Euro aufzuladen. Im Anschluss wird die Methode **printAccount()** aufgerufen, welche den Kontostand ausgibt. Folgend hierauf wird **roulette()** aufgerufen, welche das tatsächliche Spiel enthält. Alle drei Methoden befinden sich in einem Try-Catch-Block, welcher bei Exceptions, die Fehlermeldung “Es gab einen Fehler. Bitte Spiel neu starten!" auf der Konsole ausgibt.

3.3 addToAccount()

Die Methode hat die Signatur **public static int addToAccount(Scanner scanner)** und umfasst die Aufladung des Benutzerkontos. Der Spieler wird durch den Satz “Wie viel möchten Sie aufladen? (1-1000€ möglich)” dazu aufgefordert, seinen gewünschten Wert in die Konsole einzugeben. Die Eingabe des Spielers wird per Scanner erfasst und in die Integer Variable input gespeichert. In der Bedingung einer while-Schleife wird die Methode **isAccountInputValid()** aufgerufen, in der überprüft wird, ob die Eingabe gültig ist, also zwischen 1 und 1000 liegt. Solange diese Methode “false” zurückliefert, wird dem Spieler folgender Satz auf der Konsole ausgegeben: "Bitte einen Betrag zwischen 1€ und 1000€ eingeben:” und der Aufladeprozess startet erneut. Wenn die Eingabe korrekt war, gibt die Methode **isAccountInputValid()** “true” zurück, womit die While-Schleifenbedingung nicht mehr erfüllt ist und diese damit beendet wird. Mit dem return der Variable “input” wird der eingegebene Wert zurückgegeben und in die Variable “account”, die den Kontostand enthält, geschrieben.

3.4 printAccount()

Die Methode hat die Signatur **private static void printAccount(int account)** und dient dazu, den Kontostand durch System.out.println() auf die Konsole zu schreiben. Durch die if-else-Anweisung wird hierbei der Kontostand bei positivem Kontostand in grün und bei negativem Kontostand in rot auf die Konsole geschrieben.

3.5 roulette ()

Die Methode hat die Signatur **public static void roulette (int konto, Scanner scanner) throws InterruptedException.** Die Methode dient dazu, den Spielablauf zu regeln und das mehrfache Spielen zu ermöglichen. Der Spieler spielt so lange Roulette, wie der Wert von play=true ist und damit die Bedingung der while-Schleife erfüllt ist. Der anfangs deklarierte Wert von play entspricht true, sodass automatisch eine Runde Roulette gestartet wird. Am Ende jeder Runde kann der Wert von play durch die Methode **newGame()** durch die Eingabe von Nein bzw. N auf false geändert werden, was das Spielen einer weiteren Runde Roulette verhindert und das Spiel beendet.

Eine Runde Roulette läuft folgendermaßen ab:

Der Spieler wird aufgefordert anzugeben, ob er auf eine Farbe oder eine Zahl setzen möchte. Seine Antwort wird anschließend vom Scanner erfasst und durch die Methode **isInputValid()** geprüft. Solange die Eingabe nicht gültig ist, wird der Spieler durch die while-Schleife immer wieder zur erneuten Eingabe aufgefordert. Sobald die Eingabe korrekt

ist, prüfen if-Bedingungen, ob es sich bei der Eingabe um Zahl oder um Farbe handelt. Hierbei ist die Groß- und Kleinschreibung aufgrund von IgnoreCase egal. Wurde von dem Spieler Farbe eingegeben, wird die Methode **gameColour()** aufgerufen und durchlaufen. Wurde von dem Spieler Zahl eingegeben, wird die Methode **gameNumber()** aufgerufen und durchlaufen.
Im Anschluss an die Spielrunde wird dem Spieler sein neuer Kontostand ausgegeben und der Spieler dazu aufgefordert, anzugeben, ob er erneut spielen möchte oder nicht. Anschließend wird der eingegebene Wert in der Variable playAgain gespeichert und durch die Methode **newGame()** auf Gültigkeit geprüft. Solange diese Methode den Wert false zurückliefert, wird der Spieler über die fehlerhafte Eingabe informiert und zu einer neuen Angabe aufgefordert.
Wenn der Spieler sich für nein entschieden hat, wird der Wert von play durch die Methode **newGamel()** auf false gesetzt, wodurch die Bedingung der while Schleife nicht mehr erfüllt ist und keine neue Spielrunde gestartet wird. Stattdessen wird ein letztes Mal der Kontostand ausgegeben und der Nutzer mit der Nachricht “Danke fürs Spielen!” verabschiedet.

3.6 placeBet()

Die Methode hat die Signatur **private static int placeBet(Scanner scanner, int account)** und dient dazu den Geldeinsatz des Spielers zu steuern. Durch einen Satz auf der Konsole wird er dazu aufgefordert, den Betrag, den er setzen möchte, anzugeben. Dieser Betrag wird via Scanner erfasst und in der Integer Variable “einsatz” returned. In der Bedingung einer While Schleife wird anhand der Methode **isBetValid()** überprüft ob die Eingabe gültig war. Für ungültige Eingaben wird die Nachricht ‘“Konto überzogen! Bitte kleineren Betrag angeben!” auf der Konsole ausgegeben. Wenn die Eingabe gültig ist, wird diese Methode beendet und die Variable “bet” per return zurückgeliefert.

3.7 isBetValid()

Die Methode hat die Signatur **public static boolean isBetValid(int account, int bet)** und prüft ob der Spieler sich den gewählten Einsatz leisten kann oder ob dieser seinen Kontostand übersteigt.

3.8 isAccountInputValid()

Die Methode hat die Signatur **public static boolean isAccountInputValid(int bet)** und dient zur Überprüfung der Eingabe des Spielers in der Methode **addToAccount()**. Die Methode setzt sich aus einer kurzen if-else-Anweisung zusammen die für einen Wert zwischen 1 und 1000 “true” und für alle Werte außerhalb dieses Zahlenbereiches “false” an die Methode **addToAccount()** zurückliefert.

3.9 isInputValid()

Die Methode hat die Signatur public static boolean **public static boolean isInputValid(String bet, String[] result)** und dient zur Überprüfung der Eingabe die der Spieler beim Platzieren einer Wette tätigt. Anhand einer for-Schleife wird die Eingabe des Spielers mit einem Array, das mögliche gültige Eingaben umfasst, verglichen. Ist der Vergleich erfolgreich, gibt die Methode den Wert true zurück, ist dies nicht der Fall, gibt sie false zurück.

3.10 gameNumber()

Die Methode hat die Signatur **private static int gameNumber(Scanner scanner, int account, Random rand) throws InterruptedException** und umfasst den Vorgang eines Spiels, bei dem auf eine Zahl gewettet wurde. Der Spieler wird aufgefordert, eine Zahl zwischen 0 und 36 einzugeben, die anschließend von dem Scanner erfasst wird und in der Variable input gespeichert wird. Anschließend wird durch eine if-Bedingung geprüft, ob sich die Eingabe im gültigen Bereich von 0 bis 36 befindet. Trifft dies nicht zu, so greift das else ein und fordert den Spieler zu einer erneuten Eingabe auf.
Ist die Eingabe jedoch gültig, so kann der Spieler seinen Einsatz, welcher in der Variable einsatz gespeichert wird, über die Methode **placeBet()** festlegen. Der Einsatz wird anschließend vom Kontostand des Spielers subtrahiert.
Daraufhin wird die Methode **printNumberAnimation()** aufgerufen, wodurch das Drehen des Rads visuell nachgeahmt wird. Dann wird die Gewinnzahl per Zufall generiert und in der Variable result gespeichert.
Die Methode **printWinningNumber(result)** wird aufgerufen, wodurch dem Spieler die Gewinnzahl in einer Nachricht in der jeweils korrekten Farbe auf der Konsole ausgegeben wird.
Anhand von einer if-else-Bedingung wird überprüft, ob die gesetzte Zahl mit der Gewinnzahl übereinstimmt. Ist dies der Fall, wird die Methode **win()** aufgerufen, wodurch dem Spieler der Gewinn auf der Konsole mitgeteilt wird und der Gewinn auf das Spielerkonto aufgerechnet wird.
Ist dies nicht der Fall, wird dem Spieler seine Niederlage durch den Aufruf der Methode **lose()** mitgeteilt.
Die Variable account wird von der Methode zurückgeliefert, sodass der Kontostand bei erneutem Spielen oder Aufgeben aktuell ist.

3.11 gameColour ()

Die Methode hat die Signatur **private static int gameColour(Scanner scanner, int account, Random rand) throws InterruptedException** und umfasst den Vorgang eines Spiels, bei dem auf eine Farbe gewettet wurde. Der Spieler wird dazu aufgefordert, die Farbe anzugeben, auf die er setzen möchte. Die Eingabe wird anschließend von dem Scanner eingelesen und durch die while-Schleife kontrolliert. Hierzu wird **isInputValid()** in der Bedingung der while-Schleife aufgerufen. Solange die Eingabe nicht den Anforderungen entspricht, weil die Methode **isInputValid()** nicht den Wert true zurückgibt, wird der Spieler durch die while-Schleife zu einer erneuten Eingabe einer Farbe aufgefordert.
Ist die Eingabe jedoch gültig, so kann der Spieler seinen Einsatz, welcher in der Variable bet gespeichert wird, über die Methode **placeBet()** festlegen. Der Einsatz wird anschließend vom Kontostand des Spielers subtrahiert.
Anschließend wird eine startPosition zwischen 0 und 36 und ein rollCount zwischen 20 und 29 per Zufall festgelegt. Durch die beiden zufallsgenerierten Werte wird in der Variable winningNumber die Gewinnzahl ermittelt. In der Variable winningColour wird mithilfe der winning Number der zugehörige Farbpunkt aus der Hashmap wheel gespeichert.
Auf der Konsole wird nun das visuelle Drehen des Rads durch die Methode **printColourAnimation()** dargestellt.

Sofern die eingegebene Farbe grün ist und die Farbe Grün auch der winningColour entspricht, wird die Methode gewonnen ausgelöst und ein Multiplikator von 36 für diese Methode festgelegt.
Sollte dies nicht der Fall sein, ob die eingegebene Farbe der winningColour entspricht. Trifft dies zu, so wird die Methode gewonnen mit einem Multiplikator von 2 aufgerufen.
Sind beide if-Bedingungen ohne Erfolg, so hat der Spieler nicht gewonnen und die Methode **lose()** wird aufgerufen. Die Variable account wird von der Methode zurückgeliefert, sodass der Kontostand bei erneutem Spielen oder Aufgeben aktuell ist.

3.12 printColourAnimation()

Die main-Methode hat die Signatur **private static void printColourAnimation(int startPosition, int winningNumber) throws InterruptedException.** Hier wird die Animation des Roulette-Rads simuliert, indem die Farben schrittweise ausgegeben werden. Zu Beginn der Methode wird die Integer Variable “currentPos” definiert, die als Wert die aktuelle Position aus der Variable “startPosition” bekommt. Eine while-Schleife läuft solange, bis die aktuelle Position kleiner oder gleich 37 und ungleich der winningNumber ist. Für jeden Durchgang der Schleife wird ein Farbpunkt in der zur currentPos gehörenden Farbe ausgegeben. Die darauffolgende if-Bedingung überprüft, ob die aktuelle Position der Nummer, die zum Gewinn führt, entspricht. Ist dies der Fall, wird ein Farbpunkt in der Farbe des Gewinnfelds auf der Konsole ausgegeben und die Methode endet hier mit einem return. Sollte die currentPosition über 37 sein wird die currentPosition auf 0 gesetzt. Und in einer while-Schleife solange ein passender Farbpunkt für die aktuelle Position ausgegeben bis diese der winnnigNumber entspricht. In diesem Fall wird der zur Endposition gehörende Farbpunkt auf der Konsole ausgegeben und die Methode ist zu Ende.

3.13 printNumberAnimation()

Die Methode hat die Signatur **private static void printNumberAnimation () throws InterruptedException**. Die Methode visualisiert das Drehen des Rads für das Zahlenspiel. Hierbei wird mit einer For-Zählschleife 8-fach “Die Kugel dreht sich…” gefolgt von einer per Zufall generierten Zahl zwischen 0 und 36. Die einzelnen Sätze mit einer Pause von 0,4 Sekunden durch **Thread.sleep(400);** auf die Konsole geschrieben.

3.14 printWinningNumber ()

Die Methode hat die Signatur **private static printWinningNumber (int result).** Die Methode sorgt dafür, dass dem Spieler die Gewinnzahl beim Farbenspiel in der richtigen Farbe ausgegeben wird. Anhand einer if-else Bedingung wird überprüft, welchem Farbpunkt in der Hashmap wheel der Wert der Variable result zugeordnet wird. Dann wird dem Spieler das Ergebnis in der passenden Farbe auf der Konsole ausgegeben.

3.15 win()

Die Methode hat die Signatur public **static int win (int account, int bet, int multiplier) throws InterruptedException.** Durch die Methode wird dem Spieler mitgeteilt, dass er gewonnen hat und der Betrag, den er durch seinen Gewinn erhält. Der erhaltene Betrag wird errechnet, indem der Einsatz mit einer Variable multiplier multipliziert wird. Der Multiplier wird bei eintretenden Gewinn - je nach gewählter Setzmöglichkeit - der richtige Wert zugeordnet, bevor dem Spieler sein Gewinn auf der Konsole durch **win()** ausgegeben wird.

3.16 lose ()

Die Methode hat die Methodensignatur **private static void lose (int bet) throws InterruptedException.** Durch die Methode wird dem Spieler mitgeteilt, dass dieser verloren hat und der eingesetzte Betrag, den der Spieler dadurch verliert. Der Verlust wird dem Spieler in roter Schriftfarbe auf der Konsole ausgegeben.

3.17 newGame()

Die Methode hat die Signatur **private static boolean newGame (Scanner scanner, String antwort).** In der Methode wird geprüft, ob die in der Methode **roulette()** angeforderte Eingabe für das Weiterspielen gültig ist oder ungültig ist. Dies wird mit einer if-else if-Bedingung geprüft. Ist vom Scanner ausgelesene und in Antwort gespeicherte Eingabe Ja oder J (hierbei ist die Groß- und Kleinschreibung aufgrund IgnoreCase egal) wird die Variable play auf true gesetzt, wodurch die Bedingung von while (play == true) erfüllt ist und eine neue Runde des Spiels gestartet wird, und die Methode gibt den Wert true zurück, wodurch der Spieler nicht erneut durch die Bedingung von while (newGame(scanner,erneutSpielen) != true) aufgefordert wird, Ja oder Nein einzugeben. Wenn der Spieler Nein oder N - wieder unabhängig von der Groß- und Kleinschreibung - eingegeben hat, wird der Wert von play auf false gesetzt, wodurch keine neue Runde des Spiels gestartet wird, und die Methode gibt den Wert true zurück, wodurch der Spieler nicht zu einer erneuten Eingabe von Ja oder Nein aufgefordert wird.
Ist keine der beiden if-Bedingungen erfüllt, weil der Spieler eine ungültige Eingabe getätigt hat, gibt die Methode false zurück und der Spieler wird zur erneuten Eingabe aufgefordert.

3.18 checkAccountInput()

Die Methode hat die Signatur **private static int checkAccountInput(Scanner scanner).** Mit einer While-Schleife wird der Spieler zu einer erneuten Eingabe des Kontostands aufgefordert, sofern die Methode **isAccountInputValid()** den Wert false zurückgegeben hat.

3.19 checkNumberInput()

Die Methode hat die Signatur **private static int checkNumberInput (Scanner scanner).** Mit einer While-Schleife wird der Spieler zu einer erneuten Eingabe der Spielzahl aufgefordert, sofern die Methode **isNumberValid()** den Wert false zurückgegeben hat.

3.20 isNumberValid()

Die Methodensignatur ist **private static boolean isNumberValid (int input)**. Diese Methode dient dazu, die eingegebene Wunschzahl zu prüfen, ob diese zwischen 0 und 36 liegt. Ist dies nicht der Fall returned sie false, ist die Eingabe korrekt wird true returned.

4. Fazit

Wir haben durch unseren Programmentwurf eine vereinfachte Version des Roulettes geschaffen. Hierbei haben wir die Einbindung eines Kontostands, die Ausgabe in Farben und das visuelle Drehen des Rads auf der Konsole ermöglicht. Das Programm könnte beispielsweise noch um weitere Setzmöglichkeiten oder automatisiertes Setzen ergänzt werden.