Praktikum „SEP: Java-Programmierung“ WS 2018/19

Aufgabe 4: Rummikub

Thomas Lemberger und Martin Spießl

Rummikub

- Legespiel für zwei bis vier Spieler
- Ziel: Durch geschicktes Bilden von Kombinationen alle Steine loswerden
- Hier: Als Mehrspie
  Architektur)

Bild: Amazon

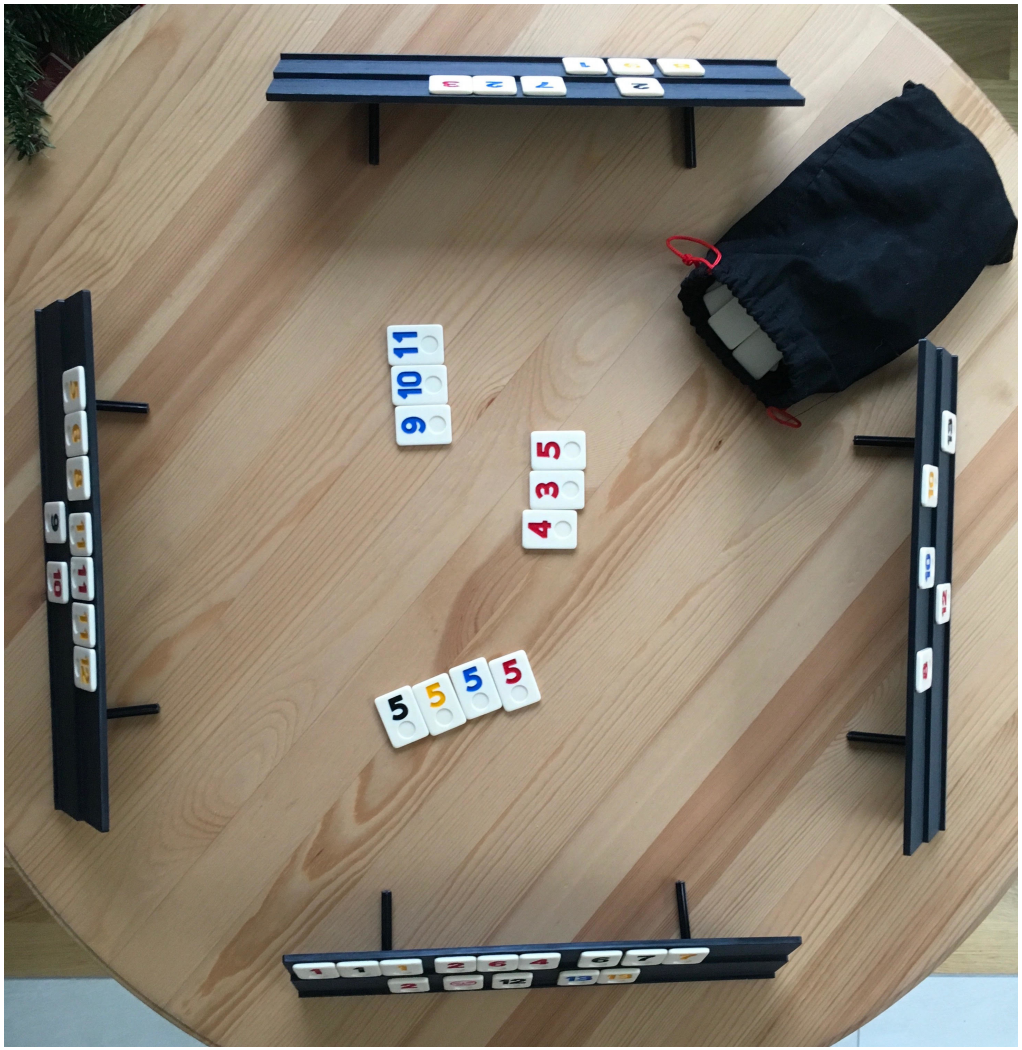
Packungsinhalt des Spiels

- 106 Steine
  - Je 2x Zahlen 1–13 in vier Farben (Rot, Schwarz, Gelb, Blau)
  - 2 Joker
- 4 Spielbretter
- 1 Beutel

Ziel des Spiels

- Spielziel:
  - Alle Steine auf der Hand ablegen
  - Bzw. möglichst wenig Punkte auf der Hand behalten
- Spielende: Erster Spieler alle Steine abgelegt

Spielaufbau



- 1 Nachziehbeutel
- 1 Ablagefeld
- Pro Spieler „Hand“ mit Steinen

Spielverlauf

- Der jüngste Spieler beginnt. ;-)
- Beginn: Jeder Spieler zieht 14 zufällige Steine und nimmt sie auf seine Hand
- Rundenverlauf, im Uhrzeigersinn: **Eine** von zwei Aktionen möglich:
  1. Einen Stein verdeckt ziehen (wandert auf die Hand)
  2. Steine ablegen

Steine Ablegen I

- Steine können von der Hand abgelegt werden, indem sie für Kombinationen auf dem Spielfeld benutzt werden
- Mögliche Kombinationen:
  - Gruppen der gleichen Zahl in unterschiedlicher Farbe (max. 4 Steine)

  - Straßen in einer Farbe (nach 13 kommt wieder 1, beliebig viele Steine einer Farbe)

- Jede Kombination besteht aus mindestens 3 Steinen
- Für erstes Ablegen („herauskommen“) 30 Punkte erforderlich
  - Jeder Stein hat den Wert seiner Zahl
  - Joker: Beim Ablegen Wert der Zahl, für die er benutzt wird
  - Joker: Zum Spielende 20 Punkte

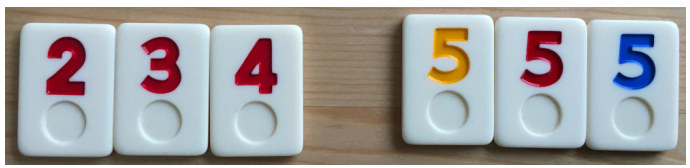
Steine Ablegen II

- Nach dem ersten Ablegen zusätzlich beim Ablegen möglich:
  - Steine auf dem Spielfeld dürfen beliebig oft beliebig neu kombiniert werden
  - Steine auf der Hand können an bestehende Kombinationen auf dem Spielfeld angelegt werden
  - Einzige Bedingung: Zu Ende des Zuges darf Spielfeld nur aus gültigen Kombinationen von je mind. 3 Steinen bestehen

Spielfeld:

Hand:

Neuer Zustand:



- Bedeutung des Joker bei Rekombinationen: freie Entscheidung

Aufgabenstellung (Teil 1)

- Rummikub GUI mit JavaFX
  - mit Nebenläufigkeit
  - Model-View-Controller muss umgesetzt werden
- Spielbar über Netzwerk mit Sockets
  - Ein Spieler ist Host (Server-Rolle)
  - Übertragen von Informationen als serialisierte Java-Objekte
  - Client schickt gewünschte Spielzüge an Host
  - Host prüft Spielzüge auf Validität
  - Falls valide: Host leitet Spielzug an alle Clients weiter
  - Falls invalide: Sender erhält Reject-Nachricht
- Host konfiguriert (falls möglich) und startet Spiel
- Port: 48410
- Jeder Spieler kann jederzeit aufgeben
- Bei Spielende wird:
  - jedem Spieler der Gewinner angezeigt
  - der Host gefragt, ob ein neues Spiel gestartet werden soll

Aufgabenstellung (Teil 2)

- Schließen des Programms
  - über üblichen Button (z.B. unter Windows der rote X-Button)
  - optional außerdem über Menü (File $\rightarrow$ Quit) oder Shortcut (z.B. Strg+Q)
- Außerdem zu erstellen und aktuell zu halten:
  - Klassendiagramm (MVC muss klar erkennbar sein)
  - Sequenzdiagramm zu vorgegebenem Szenario
  - Entwurf des geplanten GUI (später ist GUI automatisch aktuell)

Verpflichtende Abgaben I

- Heute: Teamgeist (erste Teambesprechung)
- Bis diesen Freitag, **21.12.2019**:
  - LRZ-Gitlab Repo für Projekt, uns und Tutor einladen
  - Entwurf der Diagramme
  - Bis Freitag, **04.01.2019**: Diagramme mit eventuellen Nachbesserungen/eingearbeiteten Vorschlägen
  - Unbedingt auch schon Einarbeiten, mit (prototypischer) Implementierung anfangen–hier aber keine Abgabe
- Bis **14.01.2019**:
  - Eure *Definition of Done*
  - Vorläufiger Arbeitsplan
  - Erste **ausführbare** Version des Programms mit Liste, was aktuell möglich ist

Verpflichtende Abgaben II

- Bis **21.01.2019**: nächste ausführbare Version des Programms mit Liste der Verbesserungen
- Bis **28.01.2019**: ausführbare Version des Programms mit allen geplanten Features
- Bis **04.02.2019**: finales Programm, Diagramme und Testbericht
- Am **04.02.2019, 12.15 Uhr**: Jedes Team 5 Minuten Programm präsentieren (ohne Folien)

- Alle Abgaben direkt im Repo (im Master-Branch)
  - Ordner „Abgaben“ erstellen
  - Repo erhält von uns zu jeder Abgabe einen signierten Tag

Diagramme/GUI-Entwurf

- Abgaben dürfen handschriftlich erfolgen, so lange **gut** lesbar
- Klassendiagramm und Sequenzdiagramm müssen nicht korrektes UML sein, aber alle wichtigen Informationen korrekt darstellen
- GUI-Entwurf: Ähnlich einer Blaupause
  - Je detaillierter, um so leichter ist die Umsetzung!
  - Empfehlung: Hilfslinien einzeichnen/einmalen (z.B. Gitter-Struktur oder geplante Container in JavaFX)
- Tool-Empfehlungen:
  - Diagramme: Webseite Draw.io, Microsoft Visio
  - JavaFX Mockups: Scene Builder
  - Stift + Lineal

Definition of Done

- Legt fest, wann in eurem Projekt ein Task/Feature fertig ist
- z.B.:
  - Unit-Tests für Logik vorhanden
  - ein erfolgreicher Code-Review von anderem Teammitglied
  - Erfolgreich händisch getestet,
  - Dokumentation (JavaDoc+Extern) vollständig
  - Liste der aktuellen Programm-Features aktualisiert
  - Gitlab-Issue geschlossen
  - ...
- Teammitglieder einigen sich auf einen gemeinsamen Maßstab und kontrollieren sich gegenseitig, um ihn einzuhalten
- Keine neuen Tasks anfangen, so lange ein vorheriger noch nicht beendet ist (vermeidet Ablenkung und den hohen Cost of Delay)

Arbeitsplan

- Plan für Vorgehen bei der Implementierung, mit vorläufiger Aufgabeneinteilung und evtl. Abhängigkeiten zwischen den Aufgaben.
- Empfehlungen:
  - Geschätzte(s) Schwierigkeit/Risiko/Dauer der Aufgaben aufschreiben und berücksichtigen
  - Dinge so umsetzen, dass man sie auch direkt Testen kann (z.B. nicht erst Spiel-Lobby und danach Startbildschirm)

Abgaben Programm

- Abgabe als ausführbare jar-Datei
- Nennung benötigter Java-Version
- Code im Repo

Inhalt Testbericht

- Nennung genutzter Testmethoden (manuell, Unit-Tests, Regressions-Tests, ...)
- Definition der genutzten Metrik für Testabdeckung (z.B. Statement-Coverage, Branch-Coverage)
- Testabdeckung in Prozent, im Idealfall auf Paket/Klassenebene

Organisation der Teamarbeit

- In Teambesprechungen sind **alle** gefragt
- Bei jedem Tutorentreffen erzählt jedes Mitglied **kurz** über erledigte Aufgaben, aktuelle Aufgaben und Probleme
- Jedes Teammitglied hat im Arbeitsplan klar definierte Verantwortung, die sich aber ändern kann (mit Begründung)
- Teamarbeit ist erwünscht. Möglichkeiten: Pair-Programming, Code Reviews, manuelles Testen, Tests für Logik des anderen schreiben, . . . (und natürlich gegenseitige Hilfe bei Problemen)
- Fragen immer so schnell wie möglich klären, sonst geht viel Zeit verloren
- Aufschreiben, was ihr wie lang gemacht habt
  - Nur so lernt ihr, Aufwände richtig zu schätzen und eure Zeit richtig einzuteilen
  - Erleichtert Aufstellen der Liste der Features bei Abgaben
  - Erleichtert die Antwort auf die Frage: „Was hast du die ganze Zeit gemacht?“

Tipps für Teamarbeit

- Kommunikation möglichst informell und häufig gestalten: WhatsApp, Slack, Twist, . . . ; im Notfall Moodle
- Direkt gleichen Codestyle/gleiche IDE verwenden
- Gleiche Java-Version verwenden
- CI nutzen - Unit Tests direkt schreiben
- Design-Patterns helfen euch, geben aber keine Bonuspunkte - möglichst simpel denken!

Resourcen

- JavaFX (siehe JavaFX-Folien)
- Diagramme: Vorlesung Softwaretechnik, Wikipedia
- Nebenläufigkeit: `https://tinyurl.com/JavaInselThreads`
- Sockets in Java: `https://tinyurl.com/JavaInselSockets`
- Serialisierung in Java
  `https://tinyurl.com/JavaInselSerial`
- Observer-Pattern:
  `https://tinyurl.com/JavaInselObserver`

  `https://docs.oracle.com/javafx/2/binding/jfxpub-binding.htm`