Jinx Dokumentation

- Table of Contents
- Codebase
    - UML-Diagramm
    - Klassen
        - Main
        - Card
        - NumberCard
        - LuckyCard
        - NumberCardStack
        - LuckyCardStack
        - CardHand
        - LuckyCardHand
        - NumberCardHand
        - Dice
        - Field
        - GameController
        - Game
        - PlayerController
        - Player
        - AutonomousPlayer
        - SafeScanner
        - Comparator
        - DataConnection
        - RegistCon
        - Savehistory
        - AES
        - Login
        - ResourceManager
        - SaveData
        - FileFormatter
    - Records
        - Weight
        - HighScore
    - Enums
        - LuckyCardNames
        - CardColor
        - ConsoleColor
        - AgentDifficulty
- Features
- Roadmap
- Changelog
    - Abgabe 1
        - Added
    - Abgabe 2
        - Added
        - Changed

Codebase

UML-Diagramm

Klassen

Main

Die Main Klasse ist die simpelste von alle Klassen da, diese nur über die main Methode verfügt die das Program ausführt und ein

Objekt von GameContoller initalisiert um das Spiel zu starten

Card

Card ist eine abstrakte Klasse die eine Karte darstellen soll. Von Card erben NumberCard und LuckyCard

NumberCard

NumberCard erbt von Card und dient als Abbild der nummerierten Karten im Spiel Jinx. Jede NumberCard verfügt über eine Farbe die über CardColor definiert ist und eine Nummer von 1 bis 6.

LuckyCard

LuckyCard erbt von Card ist aber auch eine abstrakte Klasse die grob die verschiedenen Glückskarten im Spiel Jinx abbilden soll. LuckyCard hat zudem noch eine effect Methode die von den jewieiligen Subklassen überschrieben werden.

- LC123
  - Wenn der Spieler diese Karte nutzt kann er sich eine Zahl von 1-3 als sein Wurfergebnis wählen. Diese Karte kann auch nach den würfeln noch genutzt werden
  - Karte kann nur ein mal genutzt werden und muss danach aus dem Deck entfernt werden
- LC456
  - Wenn der Spieler diese Karte nutzt kann er sich eine Zahl von 4-6 als sein Wurfergebnis wählen. Diese Karte kann auch nach den würfeln noch genutzt werden
  - Karte kann nur ein mal genutzt werden und muss danach aus dem Deck entfernt werden
- LCPlus1
  - Wenn der Spieler diese Karte nutzt kann er sein Würfelergebnis um + 1 erhöhen, jedoch nicht wenn er eine 6 gewürfelt hat
  - Die Karte muss nie abgelegt werden
- LCMinus1
  - Wenn der Spieler diese Karte nutzt kann er sein Würfelergebnis um - 1 verringern, jedoch nicht wenn er eine 1 gewürfelt hat
  - Die Karte muss nie abgelegt werden
- LCSum
  - Lässt den Spieler mehrere Karten aus dem Feld entnehmen die in Summe der gewürfelten Zahl entsprechen. Falls der Spieler 2 Karten dieser Art in der Hand hat kann er das Würfelergebnis um + 1 erhöhen
  - Die Karte muss nie abgelegt werden
- LCPlusDiceThrow
  - Lässt den Spieler nocheinmal Würfeln
  - Die Karte muss nie abgelegt werden

NumberCardStack

Die Klasse NumberCardStack dient zur abbildung eines Kartendecks. NumberCardStack erbt von der generischen Stack Datenstruktur und generiert bei Konstruktor auffruf ein Deck mit allen NumberCards die man zum spielen von Jinx braucht. Ist im Projekt Ordner zudem eine NumberCards.csv Datei vorhanden so wird das Deck, sowie die Reihenfolge der Karten aus der .csv Datei übernommen.

LuckyCardStack

LuckyCardStack dient zur abbildung eines Kartendecks für Glückskarten. LuckyCardStack erbt von der generischen Stack Datenstruktur und generiert bei Konstruktor auffruf ein Deck mit allen LuckyCards die man zum spielen von Jinx braucht. Ist im Projekt Ordner zudem eine LuckyCards.csv Datei vorhanden so wird das Deck, sowie die Reihenfolge der Karten aus der .csv Datei übernommen.

Dice

Die Dice Klasse simuliert einen 6 seitigen Würfel der mit der Methode use() eine zufällige Zahl von 1-6 zurückgibt

Field

Die Field Klasse spiegelt das 4x4 Kartenfeld des Spieles Jinx da. Field ist als Singleton Pattern realisiert um immer nur eine

Instanz des Fields im Spiel zu haben.

GameController

Der GameController steuert das Spiel und managt die Runden, sowie auch die Highscores der Spieler.

Game

Game ist das Herz unseres Spiels hier werden die unterschiedlichen Phasen des Spiels abgebildet.

PlayerController

Der PlayerController registriert, speichert und verwaltet die Spieler im Spiel. Er kontrolliert auch welcher Spieler gerade am Zug ist

Der PlayerController ist als Singleton Pattern realisiert um in verschiedenen Klassen die gleiche Instanz zu haben.

Player

Player dient als Datenstruktur für die unterschiedlichen Spieler im Spiel

AutonomousPlayer

Der AutonomousPlayer ist eine Klasse die von Player erbt und einen künstlich Intelligenten Spieler simuliert. Er verfügt über eine Schwierigkeitsstufe die zum anfang des Spiels gewählt werden kann und durch AgentDifficulty definiert wird.

Der AP (Autonomous Player) kann gegen menschliche und andere künstliche Spieler spielen und entscheidet anhand von der gewichtung der Karten auf dem Feld und dem Würfelergebnis Welche Karte er zieht. Je nach Schwierigkeitsstufe berechnet er Sachen mal anders.

Kriterien zur gewichtung einer Karten:

- Zahl der Karte ist größer oder gleich der durchschnittlichen Kartenzahl auf dem Feld = +1 Gewicht
- Zahl der Karte ist kleiner als die durchschnittliche Kartenzahl auf dem Feld = -1 Gewicht
- Farbe der Karte kommt in unserer Hand vor = +1 Gewicht
- Farbe der Karte kommt nicht in unserer Hand vor = -1 Gewicht
- Farbe der Karte kommt oft (≥ 33%) in unserer Hand vor = +1 Gewicht
- Farbe der Karte kommt nicht oft (< 33%) in unserer Hand vor = -1 Gewicht
- Farbe der Karte ist in Gegnerhand = -1 Gewicht
- Farbe der Karte ist nicht in Gengerhand = +1 Gewicht
- Farbe der Karte kommt wenig (< 33%) in Gegnerhand vor = +1 Gewicht
- Karten Farbe kommt wenig auf dem Feld vor = +1 Gewicht
- Karten Farbe kommt oft auf dem Feld vor = -1 Gewicht
    - Die häufigkeit einer Karte wird durch die Formel

      $\lfloor| (\frac{Karten \; im \; Feld}{4}) - (Spieleranzahl - 2) |\rfloor$ bestimmt

Damit der AP die Karten gewichten kann muss er jedoch erst immer den gefährlichsten Gegner aus machen, da der AP in jeden Zug nur gegen den gefärlichsten Spieler spielt und nicht gegen alle Spieler

Kriterien zur Gewichtung des gefährlichsten Spielers

- Der Gegner hat mehr Karten auf der Hand als der durchschnittliche Spieler = +1 Gewicht
- Der Gegner hat weniger Karten auf der Hand als der durchschnittliche Spieler = -1 Gewicht
- Der Gegner mehr als 3 verschiedene Karten auf der Hand hat = +1 Gewicht
- Ger Gegner wniger als 3 verschiedene Karten auf der Hand hat = -1 Gewicht
- Der Gegner hat mehr Punkte als der Durchschnitt = +1 Gewicht
- Der Gegner hat weniger Punkte als der Durchschnitt = -1 Gewicht

SafeScanner

SafeScanner ist ein Wrapper um den java.util.Scanner. Er stellt spezielle Methoden zur verfügung die Fehler abfangen und nur bestimmte Eingaben zulassen

CardHand

CardHand dient als Struktur fuer die Numbercards und Luckycards, die der Spieler hat

LuckyCardHand

Besteht aus den Luckycards, die der jeweilige Spieler hat

NumberCardHand

Besteht aus den Numbercards, die der jeweilige Spieler hat

Comparator

Der Comparator ist fuer die Sortierung der Match-History zustaendig

DataConnection

Stellt die Verbindung zur Datenbank her und prueft, ob der Spieler, mit dem man sich anmelden moechte, registriert ist

RegistCon

Der Spieler registriert sich hier in der Datenbank oder meldet sich im Spiel mit Daten aus der Datenbank an

Savehistory

Schreibt die Match-histories von den Spielern in die Datenbank Gibt auch die geordnete und ungeordnete Liste der Match-histories aus

FileFormatter

Formatter fuer den Logger der Spielzuege

AES

Ist fuer die Verschluesselung der Passwoerter zustaendig

Login

Ist fuer den Login ueber die Textdatei zustaendig

ResourceManager

Mit dem Resourcemanager kann man relevante Spieldaten in eine .save-Datei schreiben und auch laden

SaveData

Die Datenstruktur fuer die relevanten Spieldaten, die man speichern moechte

---

Records

Weight

Generisches Record zum gewichten von Objekten wie Spieler oder Karten. Weight hat zwei Attribute object und weight, object ist ein generischer Typ und speichert die Referenz auf das zu gewichtene Objekt. Das weight Attribute ist vom Typ int und speichert das jeweillige Gewicht.

HighScore

Das HighScore Record, dient dazu Highscores zu speichern die aus der Highscore.txt Datei ausgelesen zu werden und während Programm lauf zu verfügung zu stellen.

Enums

LuckyCardNames

Datentyp für die Konstanten der Namen aller LuckyCards.

CardColor

Datentyp für die Konstanten der Farben für die NumberCard Klasse.

ConsoleColor

Ein Enum das eine Reihe an Unicode Konstanten hergibt zum ändern der Schriftfarbe und Hintergrundfarbe im Terminal.

AgentDifficulty

Datentyp für die Schwierigkeitsstufen der Spieler KI

- EASY
- MEDIUM
- HARD

Features

- Spiele mit 2-4 Spielern gleichzeitig
- KI Spieler in 3 Schwierigkeitsstufen (Easy, Medium, Hard)
- HighScores
- Undo Funktion falls man einmal zu viel gewürfelt hat
- Farbiger Output
- Prüfung der Csv Dateien (NumberCards.csv und LuckyCards.csv)
- 100% JavaDoc
- Spiel wird gespeichert
- Datenbank und .txt support
- Match-history gespeichert
- Spielzuege gespeicher
- Replay des letzten Spiels
- Nochmal spielen

Roadmap

- [x] Spiel speichern
- [x] Registrieren in .txt
- [x] Login per .txt
- [x] Registrieren in DB
- [x] Login per DB
- [x] Spielverlauf speichern

- [x] Spielverlauf anzeigen
- [x] Spielzuege loggen
- [x] Replay-Funktion

Changelog

Alle nennenswerten Änderungen an diesem Projekt werden hier dokumentiert.

Abgabe 1

25.10.2022

Added

- Grundstruktur des Spiels

Abgabe 2

15.11.2022

Added

- SafeScanner ist eine Wrapper Klasse für die java.utils.Scanner Klasse
- Undo Funktion lässt Spieler jetzt vorherige Würfelergebnisse zurückholen
- Farbige Kartenausgabe
- LCSum implementiert
- Neustart des Spiels nach Ende eines Spiels
- Gewinner wird nun angezeigt

Changed

- LuckyCardNames Enum
    - Das Enum LuckyCardNames hält die Namen jeder LuckyCard, so sollen spätere Bugs im Code abgefangen werden, die durch simple Rechtsschreibfehler passieren können
        - Beispiel

```
 // Bad
if(luckycard.name.equals("LcSum")) // Richtiger name = LCSum

// Good
if(luckycard.name.equals(LuckyCardNames.LCSum.name()))
```

- Jede LuckyCards Klasse weiß jetzt ihren eigenen Namen

```
 // Vorher
// Konnte schnell Bugs verursachen da man sich schnell verschreiben konnte
LuckyCard lc = new LCSum("LCSum");

// Nachher
LuckyCard lc = new LCSum()
```

Abgabe 3

Added

- Spiel speichern
- Replay
- DB und txt reg und login
- DB und txt spielverlauf