

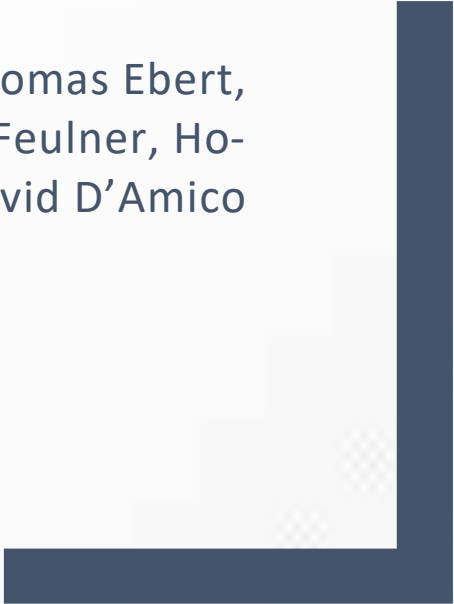


Java

ABSCHLUSSAUFGABE

Mini Mario Party

Laura Krone, John-Thomas Ebert,
Leyla Brendler, Luca Feulner, Ho-
sung Ryu, David D'Amico



Inhaltsverzeichnis

ZIELSETZUNG.....	2
HAUPTGAME.....	2
SPIELERKLASSE (LUCA)	2
WÜRFELKLASSE (LEYLA)	2
(AKTIONS-) FELD KLASSE (LAURA).....	3
MINISPIEL UND MINISPIELRÜCKGABEWERT (LEYLA UND LAURA)	3
HAUPTGAME (LAURA)	3
BALLON PLATZEN (LAURA)	4
BLACKJACK (LUCA).....	5
HÜRDENLAUF (HOSUNG)	6
KNIFFEL (DAVID).....	7
LABYRINTH (LAURA)	8
SCHERE STEIN PAPIER (LUCA)	9
TIC TAC TOE (LEYLA)	9
SCHIFFE VERSENKEN (JOHN).....	10
ANHANG	12

Zielsetzung

Die Zielsetzung unseres Projektes war es, dass jeder mindestens ein Minispiel programmiert und wir die Hauptspielklassen aufteilen. Am Ende wollten wir ein funktionsfähiges Hauptspiel mit verschiedenen kleineren Spielen (Minispiele) implementieren. Wichtig war uns, dass jeder an dem Projekt mitarbeiten kann, auch, wenn die Programmierskills nicht auf demselben Level sind. Jeder konnte für sein(e) Minispiel(e) entscheiden, welche Schwierigkeiten diese haben und inwieweit sie sich von den Originalen unterscheiden, um diese so selbstständig wie möglich zu implementieren und das Wissen der Vorlesung anwenden zu können.

Zu Beginn der Umsetzung haben wir gemeinsam einen Plan ausgearbeitet, ein Git Repository erstellt (Link am Ende des Absatzes) und die ersten Klassen des Hauptspiels implementiert, sowie verschiedene Minispiele zusammengetragen, die von unterschiedlichen Gruppenmitgliedern übernommen werden konnten. Der Fortschritt wurde über eine Exceltabelle und die Commits verfolgt. Es war keiner auf sich allein gestellt und alle haben den anderen in ihrer Kapazität geholfen und Verbesserungsvorschläge gegeben.

Das Ergebnis der Gruppenarbeit ist ein Hauptspiel, welches gegen einen Computer gespielt wird, mit 8 verschiedenen Minispielen, welche von unterschiedlicher Komplexität sind.

Des Weiteren wurde für jedes Minispiel ein Klassendiagramm erstellt, welche im Anhang vorhanden sind.

Um die einzelnen Minispiele zu testen, gibt es den MinispielAufrufTester. Bei dem kann der Klassenname des Minispiels eingegeben werden und die Schwierigkeit angepasst werden.

Link für Git-Repository: <https://github.com/LauraKrone24/MiniMarioParty.git>

Hauptgame

Spielerklasse (Luca)

Die Spielerklasse dient zum Erstellen und Bewegen der Spieler. Jeder Spieler enthält einen Namen, einen Wahrheitswert, ob es eine Person oder ein Computerspieler ist, eine Farbe, eine Liste der vorhandenen Würfel und seine aktuelle Feldposition. Alle Attribute sind privat und können nur beim Erstellen gesetzt werden. Somit enthält die Klasse die dazugehörigen Getter Methoden. Über den Konstruktor werden beim Anlegen des Spielers die Attribute initialisiert.

Die Klasse Spieler enthält die Methode fuegeWuerfelhinzu(), welche den aus dem Minispiel gewonnenen Würfel der Würfelliste des Spielers hinzufügt. Außerdem ist die Methode wuerfeln() enthalten, welche über die Würfelliste iteriert und jeden vorhandenen Würfel in der Liste Würfel ausführt, das Ergebnis zählt und den zusätzlichen Würfel nach Gebrauch wieder aus der Liste entfernt. Als letzte Methode ist bewegeSpieler() definiert, in ihr wird die davor definierte Methode wuerfeln() aufgerufen und das Ergebnis der aktuellen Position des Spielers addiert, solange es nicht über Feld 99 kommt. Im Falle eines Übertretens des Feldes 99, wird das Hauptgame beendet und es gewinnt der Spieler, der das Feld als erstes überquert hat.

Würfelklasse (Leyla)

Die abstrakte Würfelklasse enthält die Methode wuerfeln(), bei der jede Würfelzahl random erzeugt wird. Da wir drei verschiedene Würfelarten haben, war es am besten eine

allgemeine Methode für alle Würfel zu erstellen. Da alles private Variablen sind, mussten noch getter-Methoden hinzugefügt werden für das Hauptspiel. Auch ist ein Konstruktor enthalten, da es immer ein neuer Würfel ist, der initialisiert wird. Die drei unterschiedlichen Würfel sind jeweils eine Unterklasse, in denen der Konstruktor mit dem passenden Namen, Bild und Würfelzahlen (minimale und maximale Zahl auf dem Würfel) gesetzt wird. Wobei ein guter Würfel mit 4-6 und ein schlechter Würfel mit 1-3 angegeben ist.

(Aktions-) Feld Klasse (Laura)

Die Feld Klasse dient dem Zweck, die einzelnen Felder des Spielfeldes zu erstellen. Jedes Feld hat eine Nummer, sowie eine X und eine Y-Koordinate. Alle Attribute sind privat und können nur beim Erstellen gesetzt werden (final). Aus diesem Grund verfügt die Klasse auch noch über Getter Methoden für alle Attribute.

Die Klasse Aktionsfeld erbt von der Klasse Feld und erweitert diese, um die Methoden getMinispielnummer(), welche mithilfe Länge der Minispielliste aus dem Hauptspiel eine zufällige Nummer zurückgibt und starteMinispiel(), welche die Nummer aus getMinispielnummer() weiterverarbeitet. Der Ablauf der Methode starteMinispiel() sieht vor, dass zunächst das Minispiel an der entsprechenden Stelle aus der Minispielliste entfernt wird. Daraufhin wird der Spieler über einen Alert über das Aktionsfeld informiert und gefragt, ob er die leichte oder schwere Variante des Spiels spielen will. Die entsprechende Schwierigkeit wird, nachdem das Fenster geschlossen ist, eingestellt und das Minispiel wird gestartet. Nach Beendigung des Spiels wird der Rückgabewert als Alert ausgegeben und falls das Spiel nicht abgebrochen wurde, wird dem Gewinner der gewonnene Würfel hinzugefügt. Zum Schluss wird Hauptspiel die Methode nextSpieler() aufgerufen, um mit dem Hauptgame fortzufahren.

Minispiel und Minispielrückgabewert (Leyla und Laura)

Die abstrakte Minispielklasse erbt von der Application-Klasse. Da wir für die verschiedenen Minispiele einen einheitlichen Rahmen gestalten wollten, haben wir eine gemeinsame Oberklasse erstellt, von der die einzelnen Minispiele erben, um die Rückführung zum Hauptspiel zu vereinfachen und vereinheitlichen. Damit haben wir unser Wissen aus der Vorlesung zum Polymorphismus angewendet. Zuerst gibt es ein einheitliches Layout für die verschiedenen Minispiele, einer einheitliche „Menüleiste“ mit Zurück Button, Titel und Spielanleitung. Hier wird auch der Rückgabewert zum Hauptspiel definiert, der in den jeweiligen Unterklassen angepasst wird, genauso wie das Schwierigkeitslevel durch eine boolean Variable.

Minispielrückgabewert setzt für jedes Minispiel entsprechend die richtigen Werte (Gewinner, Abbruch und welcher Würfel zugeteilt wird), um diese an das Hauptspiel zurückzugeben. Ein leichtes Spiel gibt einen schlechten und ein schweres Spiel einen guten Würfel zusätzlich zurück.

Hauptgame (Laura)

Das eigentliche Hauptgame erbt von der Klasse Application. In dieser Klasse werden die Spieler, das Spielfeld sowie das Anzeigen von Änderungen in der Oberfläche verwaltet. Zur Erstellung der Oberfläche wurde kein FXML-File verwendet, sondern alle Elemente wurden direkt im Code implementiert.

Neben den Oberflächenattributen verwaltet die Klasse auch noch ein Array aus 100 Feldern, ein Array aus 2 Spielern, einen Integer Wert, welcher anzeigt, welcher der beiden Spieler gerade dran ist, sowie eine Liste aus Minispielen, in denen Objekte der verschiedenen Minispiel Subklassen enthalten sind. Die Methode start() initialisiert zunächst die Oberfläche, ohne

diese aber anzuzeigen. Dann gibt es einen Popup Dialog, in welchem der Spieler seinen Namen eingeben kann. Die Spieler und die Felder werden angelegt (Felder anlegen implementiert von Luca) und entsprechenden Positionen in den Arrays zugewiesen. Auch die Minispielliste wird an dieser Stelle initialisiert. Nun wird noch der Startspieler ermittelt und die Oberfläche noch einmal geupdatet, bevor sie angezeigt wird. Wenn der Computer anfangen darf, wird zudem der erste Zug ausgeführt. Der Zug für den Spieler wird erst bei Betätigen des Würfelbuttons ausgeführt.

Innerhalb der statischen privaten `zug()` Methode wird der aktuelle Spieler bewegt und die Position seiner Figur aktualisiert. Wenn es sich bei dem aktuellen Feld nun um ein Aktionsfeld handelt, wird zudem das ein Minispiel gestartet. Wenn die Variable `gewonnen` `true` ist, wird denn die Methode `gewinner()` mit dem aktuellen Spieler als Parameter gestartet, sonst wird, wenn aktuell kein Minispiel läuft, die Methode `nextSpieler` aufgerufen.

Diese Methode setzt erst die Nummer des aktuellen Spielers mit Hilfe der Methode `changeSpieler()` auf den nächsten Spieler um und updatet dann, nach einer kurzen Pause, die Oberfläche (`updateOberfläche()`). Nach einer weiteren Pause wird dann der nächste Zug freigegeben. Das bedeutet, entweder wird der Computerzug gestartet oder der Würfelbutton für den Spieler wird freigegeben.

Die Methode `gewinner()` stellt den Endpunkt des Spiels dar. In ihr wird ein Alert erstellt, welcher den Spieler informiert, ob er gewonnen oder verloren hat. Zudem wird das Spielfeld nach 2 Sekunden ebenfalls in dieser Methode geschlossen.

Ballon Platzen (Laura)

Bei dem Minispiel Ballonplatzen handelt es sich um ein Schnelligkeitsspiel, welches in seinen Grundfunktionalitäten an die Aufgabenstellung Ballonplatzen angelehnt ist.

Der inhaltliche Fokus dieses Spiels lag auf der Arbeit mit Threads, um parallel und unabhängig voneinander Veränderungen an verschiedenen Objekten bzw. Elementen auf der Oberfläche vorzunehmen.

Auch die Aspekte der Objektorientierung, wie Vererbung, abstrakte Klassen und Datenkapselung, spielten in diesem Spiel eine maßgebliche Rolle. Neben diesen großen Themen wurden in dem Minispiel auch kleinere Elemente der Vorlesungen wie z.B. Lambda-Expressions oder auch Java FX integriert.

Zielsetzung dieser Teilaufgabe war es ein Minispiel zu programmieren, bei dem der Spieler in einer festen Zeit durch das Klicken auf wachsende Ballons diese platzen lassen und somit Punkte sammeln kann. Die Ballons sollen dabei nach und nach an zufälligen Stellen auf dem Hintergrund erscheinen. Es sollten dabei für das Minispiel zwei verschiedene Schwierigkeitsstufen entwickelt werden.

Die Anzahl, der auf dem Spielfeld erscheinenden Ballonarten, unterscheidet sich je nach Modus zwischen zwei und drei. Normale Ballons haben im leichten Modus eine Wahrscheinlichkeit von ca. 92.5% und goldene Ballons eine Wahrscheinlichkeit von 7.5%. Im schwierigen Modus verschieben sich die Wahrscheinlichkeiten zu 33% für einen schwarzen, ca. 62 % für einen normalen und 5% für einen goldenen Ballon. Diese Anpassung ist eine von drei, um die Schwierigkeit die Punktzahl von 10 000, welche zum Gewinnen nötig ist, zu erhöhen.

Alle Ballons werden über eine abstrakte Klasse `Ballon` implementiert, welcher eine Subklasse von `Ellipse` ist. Die Klassen `BlackBallon`, `NormalerBallon` und `GoldBallon` erben alle von der Klasse `Ballon` und unterscheiden sich nur in der standardmäßigen Wertbelegung der Attribute `worth` und `color`. Jegliche Funktionalität beider Klassen ist in der Parentklasse `Ballon`

implementiert. Um die Ellipsen wie richtige Ballons aussehen zu lassen besitzen die Ballons zudem auch noch ein CubicCurve als „Ballonschnurr“.

Ballons werden ca. alle 0.25 Sekunden erzeugt und wachsen dann abhängig von der Schwierigkeitsstufe unterschiedlich lange. Das Erzeugen der Ballons wird dabei über einen rekursiven, durch eine PauseTransition verzögerten Aufruf der Methode ballonErzeugen() erzeugt. Das Wachsen des Ballons erfolgt dann in einem zusätzlichen Thread, um das Gesamtprogramm nicht zu behindern.

Die Funktionalität grow() ist dabei im Sinne der Objektorientierung in Ballon selbst implementiert.

Auch die Funktionen move(), welche den Ballon in der schwierigeren Variante in eine zufällige Richtung bewegt, und die Methode moveline(), welche die Position der CubicCurve passend zu Ballongröße und Position anpasst, sind aus diesem Grund an dieser Stelle hinterlegt worden.

Die Methode move() ist die zweite Anpassung, um das Spiel zu erschweren, da es so kniffliger wird den Ballon, besonders wenn er noch klein ist zu treffen.

Die letzte Anpassung für die schwierige Variante ist ein Faktor, der welcher sowohl die maximale Größe in Form eines Parameters der Methode grow() als auch die Anzeigezeit in maximaler Größe beeinflusst.

Das Platzenlassen der Ballons wird über einen EventListener auf jedem einzelnen Ballon verwaltet und startet die Punkteberechnung, welche wiederum ebenfalls in Ballon durchgeführt wird.

Die Punkte berechnen sich dabei anhand des Wertes (normal = 1, gold = 20, schwarz = -10) und 100 minus des aktuellen Radius des Ballons.

Das Spiel sollte, besonders im schweren Modus, nicht zu einfach werden und dennoch musste das Ziel nichtdestotrotz erreichbar sein, auch wenn der Spieler keine Maus, sondern ein Trackpad benutzt.

Aus dem Versuch beide Aspekte auszubalancieren, ergab sich dann die Kombination von Bepunktung der Ballons und Anzahl der nötigen Punkte bis zum Sieg.

BlackJack (Luca)

Das hier erstellte Spiel ist eine abgewandelte Version des beliebten BlackJack. Das Ziel des Spieles war ein selbstspielender Gegner, welcher vom Computer gespielt wird. Die Grundidee des Spiels ähnelt sehr dem Spielverfahren von Blackjack. So zieht sowohl der Spieler als auch der Computer von einem Kartenstapel, welcher aus zwei normalen Kartensets besteht, die jedoch kein Ass enthalten. Die Gewinnkriterien sind nahezu identisch wie bei Blackjack, mit der einzigen Ausnahme, dass wenn sowohl Computer als auch Spieler über die bekannte 21 kommen, die Runde als unentschieden gewertet wird. Um das Minispiel als Spieler gegen den Computer zu gewinnen, werden 5 gewonnen Spielrunden benötigt. Alle zusehenden grafischen Elemente wurden auf Basis von JavaFX implementiert. Die Hauptklasse „BlackJackMinispiel“ ist eine Subklasse der von Leyla erstellten Minispielklasse. Ebenfalls wurden zwei „Hilfsklassen“ erstellt, die zur Erstellung der benötigten Spielkarten dienen. So enthält die Klasse Spielkarten, den Konstruktor und die dazugehörigen Getter-Methoden. Die Klasse Karteninitialisieren dient zur Erstellung der Spielkarten, welche den Konstruktor aus der Klasse Spielkarten verwendet. Der Spieler hat die Möglichkeit von dem vorhandenen Kartenstapel beliebig viele Karten zu ziehen, um so nah wie möglich an die 21 zu gelangen. Das hierbei beste Ergebnis, was erzielt werden kann, ist das genaue Treffen der 21. Möchte der Spieler seinen Zug für diese Runde beenden, so steht ihm der Button Stop zur Verfügung. Mit Drücken des

Buttons fängt der Computer an nach und nach seine Karten zu ziehen. Hierbei wurde eine Pausetransition benutzt, um das Nachvollziehen der gezogenen Karten für den Spieler zu ermöglichen. Das Verfahren, welches für das Kartenziehen des Computers verantwortlich ist, basiert auf dem Verfahren von Blackjack. Der Computer zieht so lange eine Karte, bis die Summe der Kartenwerte 17 übersteigt. Sobald der Computer seinen Zug automatisch beendet hat, wird die Gewinnauswertung gestartet. Um als Spieler die Runde für sich zu entscheiden, muss die Punktzahl unter 22 sein und höher als die Zahl des Computers, solange dieser nicht die 21 überschritten hat. Erfolgt ein Gleichstand oder beidseitiges Überschreiten der 21 wird die Runde als Unentschieden gewertet. Bei allen anderen Möglichkeiten hat der Spieler die Runde verloren und dem Computer wird ein Punkt zugeschrieben. Bei der Programmierung des Spiels wurden mehrere Themen der Programmiervorlesung berücksichtigt. So enthält das Spiel Eltern und Kindklassen, Lambdaexpressions, Konstruktoren sowie die dazugehörigen Getter-Methoden und vieles mehr.

Hürdenlauf (Hosung)

Im Spiel "Hürdenlauf" springt der Spieler über Hürden und versucht dabei den Computer zu übertreffen, um einen Würfel zu erhalten.

Zu Beginn werden zwei Klassen erstellt: Die Klasse "PlayerObj" repräsentiert die Spielfiguren für Spieler und Computer, und die Klasse "Hurdle" repräsentiert die Hürden. Beide Klassen besitzen ein Attribut der Klasse "Rectangle", welches die "Hitbox" darstellt. Wenn sich diese beiden Rechtecke überlappen, wird das Spiel beendet.

In der Klasse "PlayerObj" wird die Methode "jump()" implementiert, die eine Schleife mit den Befehlen "moveUp()" und "moveDown()" enthält, um das Springen der Objekte zu ermöglichen. Beim Ausführen dieser Befehle wird das Rechteck zuerst um eine bestimmte Anzahl von Pixeln nach oben und dann nach unten verschoben. Im Spiel sind es 6 Pixel.

In der Klasse "Hurdle" werden die Objekte beim Aufrufen der Methode "move()" nach links verschoben.

Für die Ausführung des Spiels wird die Klasse "Hürdenlauf" implementiert, in der die Instanziierung und Ausführung der Befehle beider Klassen erfolgen. Zuerst werden zwei Panes erstellt, jeweils mit einem Objekt der Klasse "PlayerObj". Die Hürden werden mit dem Befehl "addHurdles()" erstellt und dargestellt. Die erstellten Hürden werden mit dem Befehl "moveHurdles()" verschoben. Diese Methoden werden jeweils in den Befehlen "handleplayergame()" und "handlecomputer()" für das entsprechende Pane ausgeführt. Die "handle"-Befehle werden mithilfe von "AnimationTimer" alle 0,85 Millisekunden wiederholt. Bei jedem Durchgang wird mit dem Befehl "checkCollision()" überprüft, ob sich die Rechtecke der Klasse "PlayerObj" und "Hurdle" berühren oder nicht.

Es wurde jedoch ein Problem festgestellt: Der "AnimationTimer" funktioniert nur richtig bei Laptops, die mit Apple M1- und M2-Chips ausgestattet sind, aufgrund unterschiedlicher Tickraten der Prozessoren.

Der Spieler kann durch Drücken der W-Taste den Befehl "jump()" ausführen. Im Gegensatz dazu erkennt der Computer mit dem Befehl "hurdleAhead()", wie weit eine Hürde von ihm entfernt ist, und springt entsprechend. Mithilfe von "Math.random" wird eine Zahl von 10 bis 20 festgelegt, um anzugeben, wie oft der Computer springt.

Wenn der Spieler oder der Computer mit den Hürden kollidiert, wird das Spiel mit dem Befehl "stopGame()" beendet. Erst wenn das Spiel des Spielers beendet ist, erfolgt die Gewinnauswertung.

Kniffel (David)

Kniffel ist eines der bekanntesten Würfelspiele weltweit und in einer abgewandelten Version Teil des gemeinschaftlichen Projektes. Für die Zielsetzung für dieses Minispiels gab es mehrere Gesichtspunkte. Es sollte ein Spiel werden, bei dem der Echtzeitspieler gegen den Computer spielen kann, welcher eigenständig seine Züge macht. Dabei haben beide insgesamt bis zu drei Würfe mit je fünf Würfeln, welche allesamt random erzeugt werden. Anhand der auf dem Hintergrund platzierten Punktetabelle kann der Echtzeitspieler die möglichen Punkte für ein Ergebnis sehen und durch Klicken auf den Zählen-Button selbst entscheiden, ob die Punkte vorzeitig gezählt werden sollen oder er das Risiko in Kauf nimmt und die Punkte nach dem dritten Würfeln automatisch gezählt werden. Dies unterscheidet sich nicht vom Verhalten des Computers. Dort wurde für die zwei möglichen Schwierigkeitsgrade eine Logik implementiert, welche es dem Computer ermöglichen ebenfalls vorzeitig Punkte anzunehmen – ab 15 Punkten bei der leichten Version und ab 25 Punkten bei der schweren Version. Da diese Version von Kniffel ebenfalls eines der implementierten Minispiele ist, erbt auch dieses Spiel von der Klasse Minispiel. Um die Logik des Würfelspiels umzusetzen, bedarf es noch weiterer Klassen. Daher wurde eine Klasse „Wuerfel“ erstellt, welche durch die dort vorhandene Methode dafür sorgt, dass den random erzeugten Würfeln auch das passende Bild übergeben werden kann. Dann gibt es noch zwei im Grunde identische Klassen für das Initialisieren der Würfel – eine Klasse für die Würfel des Spielers und eine Klasse für die Würfel des Computers. Innerhalb der Klassen wird eine Liste von „Wuerfel“ erstellt und die Grundlage für die verschiedenen Würfel geschaffen sowie deren dazugehöriges Bild zugewiesen. In der Klasse „KniffelMinispiel“, in der das eigentliche Spiel programmiert ist, sind dann sämtliche graphische Elemente auf Basis von JavaFX implementiert. In den Methoden „initialisiereWuerfelSpieler“ und „initialisiereWuerfelComputer“ werden dann die ImageViews der einzelnen Würfel erstellt und deren Platz auf dem Spielfeld festgelegt. Bei den Methoden „wuerfelVorgangSpieler“ sowie „wuerfelVorgangComputer“ wird dann die Logik des Spielers bzw. des Computers für den Würfelvorgang implementiert und die random Zahlen für die Würfel generiert. Anhand dieser random Zahl werden dann die Würfel auf dem Spielfeld erzeugt. Um dem Spieler ein visuell bestmögliches Erlebnis zu bieten, wurde mit einigen „PauseTransition's“ gearbeitet, wodurch ermöglicht wird, dass auch die Züge des Computers nachvollziehbar sind. Auch werden die random Zahlen in einer ArrayList gespeichert. Im weiteren Verlauf wird diese Liste dann ausgelesen, wodurch die Punkteermittlung erfolgen kann, welche durch die Nutzung von „if-Bedingungen“ bzw. „Collections“ ermöglicht wird. Nach jeder Punktevergabe findet eine Punkteprüfung statt, wonach dann eine Gewinnprüfung stattfindet.

Labyrinth (Laura)

Das Minispiel Labyrinth ist ein Strategie- und Schnelligkeitsspiel, bei dem es darum geht, so schnell wie möglich durch ein Labyrinth zu navigieren.

Zielsetzung des Minispiels Labyrinth ist es, dass sowohl der Computer als auch ein Spieler von zwei verschiedenen Startpunkten zu einem gemeinsamen Zielfeld in der Mitte eines Labyrinths, welches jedes Mal neu generiert wird, navigieren können. Der schnellere der beiden gewinnt dabei das Minispiel. Für den Computer sollen zwei verschiedene Algorithmen als Schwierigkeitsstufen implementiert werden.

Das Labyrinth Minispiel greift erlernte Inhalte zum Thema Algorithmen, Datenstrukturen und zum Thema Threads und Java FX Programmierung auf.

Labyrinth erbt wie jedes andere Minispiel von der Klasse Minispiel und somit auch von der Java FX Klasse Application.

Um die Aufgabe umzusetzen, wurde erst die Generierung des Labyrinths mithilfe der Klasse LabyrinthField implementiert, später wurden dann die Funktionen zur Navigation durch das Labyrinth für Computer und Spieler hinzugefügt.

Um ein Labyrinth zu generieren, bei welchem sowohl der Computer als auch der Spieler das Ziel von ihren jeweiligen Startpositionen erreichen können wird ein Breitensuche Algorithmus verwendet. Dabei sind die Seiten von Spieler und Computer nicht zwingend identisch, sodass es sein kann, dass einer der beiden durch Glück einen kürzeren Weg durch das Labyrinth hat. Die Klasse LabyrinthField, welche zur Erzeugung des Labyrinths verwendet wird und sowohl als Mauer als auch als Weg dient, erbt dabei von der JavaFX Klasse Rectangle wodurch die 441 LabyrinthField Objekte direkt auf der Oberfläche der Application dargestellt werden können. Jedes einzelne LabyrinthField fungiert dabei auch als ein Knoten und hat auch die für einen Breitensuchealgorithmus nötigen Attribute markiert und nachbarn. Um die Knoten zu einem Graphen zu verbinden, besitzt jedes Feld die Attribute top, bottom, left und right, sowie ein HashSet namens nachbarn welches alle diese Attribute zusammenfasst. Die Verbindung zwischen den Feldern ist bidirektional. Beim Erstellen eines Feldes auf der rechten Seite eines bereits erzeugten Feldes wird also vom rechten Feld das Attribut left und vom linken das Attribut right gesetzt. Für den eigentlichen Breitensuche Algorithmus wäre ein bidirektionales Verbinden über das nachbar Attribut ausreichend, doch ist es für die Navigation des Spielers wichtig, dass auch bekannt ist in welche Richtung ein Feld benachbart ist.

Die Navigation des Spielers im Labyrinth erfolgt über die einen EventListener, welche auf Tastenschläge reagiert und bei den Tasten A,W,S,D das aktuelle Feld des Spielers verschiebt. Wenn das neue Feld eine Mauer wäre, wird das aktuelle Feld nicht bewegt.

Für die Schwierigkeitsstufen des Computers wurden der Linke – Hand Algorithmus und der Breitensuche Algorithmus gewählt. Bei der schweren Variante hat der Computer also den Vorteil, dass er den optimalen Weg bereits kennt, bevor er losläuft.

Für den Linke-Hand-Algorithmus kontrolliert der Computer immer, ob ausgehend von seiner aktuellen Bewegungsrichtung das linke Feld von ihm frei ist, wenn ja „dreht“ er sich nach links sonst versucht er geradeaus zu laufen. Ist ein Geradeauslaufen nicht möglich dreht er sich so lange nach rechts, bis ein Weiterlaufen möglich ist.

Für die schwierige Variante wird der Pfad, der bereits durch den Breitensuche Algorithmus zum Überprüfen der Lösbarkeit des Minispiels genutzt wurde, verwendet. Der Computer läuft diesen Pfad nur ab.

Die Bewegung des Spielers und die des Computers wurden dabei in verschiedenen Thread implementiert, damit sich diese nicht gegenseitig behindern.

Beendet wird das Spiel, wenn einer der beiden das Zielfeld erreicht.

Schere Stein Papier (Luca)

Wie Schere, Stein und Papier funktioniert, muss man wohl keinem Kind erklären. Auch dieses Spiel hat es als eins der Minispiele in Mini Mario Party geschafft. Da es hier als Minispiel implementiert wurde, erbt es, wie die anderen Spiele, von der Elternklasse Minispiel. Auf unterschiedliche Schwierigkeitsstufen wurde hier verzichtet, da die einzige Option ein manipulierter Computer gewesen wäre, der jedes Spiel gewinnt und somit den Spielspaß nimmt.

Wie bei den anderen Minispielen wurde zuerst mit der GUI angefangen, um einen groben Überblick über das systematische Platzieren einzelner Objekte zu erhalten. Der Spieler findet vor sich drei unterschiedliche Buttons, die sich aus Schere, Stein und Papier ergeben, welche nach Belieben des Spielers gewählt werden können. Sobald der Spieler einen der Buttons gedrückt hat, erscheint zur schöneren Darstellung das Symbol des ausgewählten Buttons. Um nicht den Gedanken des mangelnden Computers aufkommen zu lassen, wählt der Computer zeitgleich eines der drei Symbole. Auch dieses wird zur schöneren Anschaulichkeit als Symbol dargestellt. Gewonnen hat der Spieler, der als Erstes fünf Runden für sich entschieden hat.

Damit ein Bild auf Knopfdruck erscheint, wurden bei den Buttons Eventhandler benutzt, die das Symbol des ausgewählten Buttons dem Spieler darstellt. Sobald der Button benutzt wurde, wird zufällig eine Zahl zwischen 0-2 generiert, die zur Auswahl des Computers dient. Zeitgleich wird mit if und else-if verglichen, wer die Runde gewonnen hat. Sobald der Spieler oder Computer fünf Runden für sich entscheiden konnte, wird das Minispiel beendet und dem Gewinner der zusätzlich gewonnene Würfel in die Würfelliste zugefügt.

Tic Tac Toe (Leyla)

TicTacToe ist eines der Minispiele und erbt daher von der Minispielklasse. Die Zielsetzung dieses Spieles war es zwei verschiedene Schwierigkeitsgrade zu implementieren, einen leichten bei dem der Computer random setzt und ein schwerer bei dem der Computer über einen Algorithmus verfügt.

Angefangen wurde mit der Oberfläche und der leichten Version des Spiels. Zuerst wurde implementiert, dass die Person vor dem Bildschirm gegen sich selber spielen kann, damit überprüft werden konnte, ob alle Gewinnmöglichkeiten abgefragt werden. Dann wurde der random Computer hinzugefügt und als letztes der Algorithmus für den schweren Computer. Danach wurden nur noch Feinheiten im Code behoben und das Layout angepasst.

Bei der leichten Variante des Spiels wird zufällig ausgewählt wer beginnt durch Math.random mit einer Zahl von 1-2. Bei der schweren beginnt immer der Spieler, da der Algorithmus darauf abgestimmt ist. Der Spieler setzt durch Klicken auf den jeweiligen bevorzugten Button, somit wird die zeichenSetzen() Methode aufgerufen. Züge des Spielers und Computers können nicht rückgängig gemacht oder überschrieben werden. Einige Probleme bei der Erstellung des Spiels waren, dass der Computer nicht automatisch gesetzt hatte, nach neun Zügen automatisch unentschieden angezeigt wurde, ohne nochmal zu überprüfen ob einer gewonnen hat oder sich das Programm aufgehängt hat beim neunten Zug. Das automatische Setzen wurde einfach behoben durch das Aufrufen der computerSetzen()/computerSchwerIstDran() Methode in zeichenSetzen(). Bei den neun Züge Problemen war es jeweils eine falsche Abfrage. Auch wurde versucht alles in Methoden abzulegen, um den Code übersichtlicher und verständlicher zu gestalten.

Zur Unterscheidung der beiden Schwierigkeitsgrade im Code wurde mit if-else Abfragen gearbeitet.

Bei der leichten Variante wird bei jedem Zug des Computers eine random Zahl generiert und mit einer Liste abgeglichen in denen die belegten Buttons sind, um ein Abbruch des Spiels durch doppeltes Setzen seitens des Computers zu verhindern. Hier ist die Gewinnrückgabe an das Hauptspiel so aufgeteilt, dass der Spieler nur den schlechten Würfel bekommt, wenn er gewinnt. Ansonsten bekommt der Computer ihn, da random setzten keine Gewinnmöglichkeit des Spielers erkennt und diese nicht verhindert außer durch Zufall.

Bei der schweren Variante wurde sich für ein MinMax() Algorithmus entschieden, welcher auf das Spiel angepasst wurde. Der Spielergewinnzug wird mit 2, Unentschieden mit 1, Computergewinnspielzug mit 0 gewertet. Falls noch nicht belegte Buttons vorhanden sind, wird mit -1 gewertet. Die beiden min() und max() Funktionen rufen sich jeweils gegenseitig auf, um den besten Zug für den Computer zu finden. Beim Durchspielen ist dann aufgefallen, dass der Spieler gar nicht gewinnt. Denn der Computer spielt perfekt und wenn auch der Spieler perfekt spielt kommt es immer zu einem Unentschieden. Also wurde versucht den ersten Zug des Computers random zu generieren, damit der Spieler auch gewinnen kann, aber das hat nicht funktioniert und den Algorithmus fehlerhaft werden lassen. Daher wurde die Gewinnrückgabe an das Hauptspiel angepasst, indem der Spieler den guten Würfel bekommt, wenn er unentschieden spielt und ansonsten bekommt der Computer ihn.

Aus der Vorlesung verwendete Inhalte sind Algorithmen, Polymorphismus, Lambda Expressions, Schleifen, Methoden und switch-case.

Schiffe Versenken (John)

Bei dem Spiel Schiffe Versenken in unserem Projekt handelt es sich um eine Abwandlung des gleichnamigen Strategiespiels. Der gewählte Modus heißt U-Boot-Jagd was bedeutet, dass es nur U-Boote (ein-Kästchen Boote) gibt. Weiterhin kann der Nutzer seine Boote nicht manuell platzieren, sondern bekommt sie am Anfang des Spiels zufällig generiert. Gespielt wird gegen einen Computer der zwei verschiedene Schwierigkeitsstufen hat.

Wie jedes andere Minispiel erbt Schiffe Versenken von der Klasse MiniSpiel, zu Beginn werden die globalen Variablen gesetzt, startet das Spiel werden zuerst die Default UI-Elemente angepasst. Anschließend werden die benötigten Labels und Buttons zum Verstehen und Bedienen des Spieles gesetzt und die benötigten Methoden aufgerufen. Als erste Methode wird generateBoats() aufgerufen welche die Boote (10 Boote für den Spieler und den Computer) generiert sowie der CButtonClickHandler gesetzt wird. Danach werden mit initPgame() und initCgame() die Spielfelder der beiden Spieler erzeugt, diese bestehen aus jeweils 36 Buttons. Nachdem die Felder erstellt sind, werden sie durch ColorPboats() und ColorCgame() optisch angepasst. Hierbei setzt ColorPboats() die Farbe der Boote des Spielers und ColorCgame() die Optik der Felder des Computers. Anschließend werden noch alle Labels und Buttons die in der start() Methode erzeugt wurden dem Haupt-Pane p angehängen mit p.getChildren().addAll(*Name der Labels und Buttons*).

Die Logik des Spieles befindet sich in der inneren Klasse CButtonClickHandler, zuerst werden auch dort die nötigen Variablen gesetzt und der Konstruktor festgelegt. Über den Konstruktor werden die benötigten Label und Variablen der Klasse Schiffe Versenken an die Klasse übergeben. In der Methode handle() befindet sich nun die Logik des Spiels, welche durch Klicken eines Feldes des Computerspielfelds ausgelöst wird. Zuerst wird überprüft welcher Schwierigkeitsgrad ausgewählt wurde (Übergabe bei der Auswahl zu Beginn des Spieles).

Ist leicht ausgewählt, läuft das Spiel wie folgt: Der Spieler beginnt und wählt ein Feld aus unter dem er ein Boot vermutet, dieser Schuss wird dann durch eine einfache if-Abfrage geprüft. Ist es ein Treffer, befindet sich also hinter der Koordinate des Buttons ein Boot so wird dieses aus dem Boot-Array entfernt (auf 0 gesetzt) und das Feld wird für den Spieler sichtbar rot markiert (zerstörtes Boot), Counter und Labels werden entsprechend aktualisiert. Der Spieler darf nach einem Treffer erneut einen weiteren Button auswählen. Ist hinter dem gewählten Button kein Boot so wird dieser mit "XXX" (verfehlter Schuss) markiert und der Computer ist am Zug. Der "einfache" Computer wählt nun zufällig ein Feld des Spielers aus und feuert, die Verarbeitung folgt demselben Schema wie beim Spieler (Treffer = rot, Computer feuert erneut; verfehlt = "XXX", Spieler ist am Zug). Dieser Ablauf wird so lange wiederholt, bis eine Flotte zerstört ist (Alle Werte des Boot-Arrays = 0). Dies wird überprüft durch die `gewinnueberprüfung()` Methode in der Klasse `Schiffe versenken`, welche die beiden Counter der Spieler überprüft. Im Falle eines Sieges oder einer Niederlage setzt sie alle Spielfelder und dazugehörige Labels auf nicht sichtbar und stellt dafür den `end` Button sowie das `PlayerWins` oder `PlayerLooses` Label auf sichtbar. Betätigt der Nutzer nun den `end` Button wird er zur Lobby zurückgeführt und erhält seine Belohnung (bei Sieg), dies wird durch die `cancel()` Methode gewährleistet. Die `cancel()` Methode überprüft den Siegerstatus sowie die Schwierigkeit, um die entsprechende Belohnung auszugeben, beides durch eine if-Abfrage, zum Schluss wird das Fenster des Spieles geschlossen durch `stage.close()`.

Ist hingegen schwer ausgewählt, agiert der Computer wie folgt:

Der grundlegende Spielablauf ist derselbe wie bei dem "einfachen" Computer. Der "schwere" Computer hat nun jedoch die Fähigkeit sich seine Schüsse zu merken und feuert dadurch nicht mehrfach auf dasselbe Feld. Diese Änderung wird durch das `shots`-Array gewährleistet, welches die Schüsse des Computers speichert. Dieses Array wird nun nachdem festgestellt wurde das kein Schiff hinter der Koordinate des Feldes liegt mit den Koordinaten abgeglichen, ist die Koordinate bereits im Array so sucht der Computer eine neue, ist sie es nicht wird trotzdem gefeuert und der Computer verfehlt.

Die Belohnungen, die ausgegeben werden, sind nach Schwierigkeit gestaffelt, leicht gibt einen schlechten Würfel, schwer einen guten Würfel und eine Niederlage logischerweise keinen Würfel.

Anhang

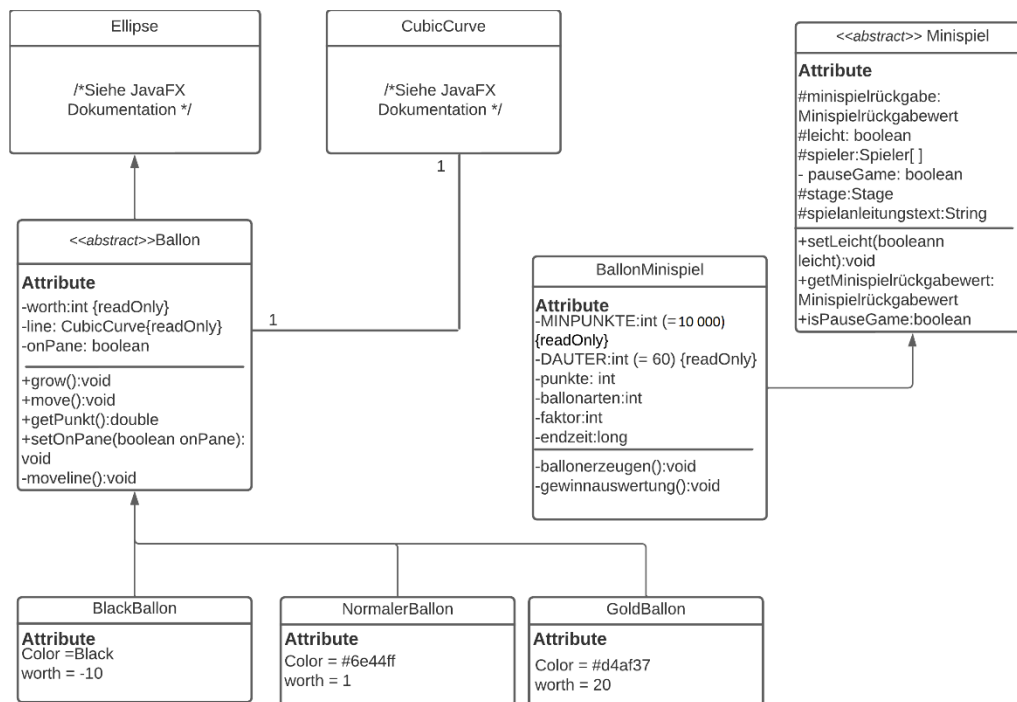


Abb. Klassendiagramm Ballon Minispiel

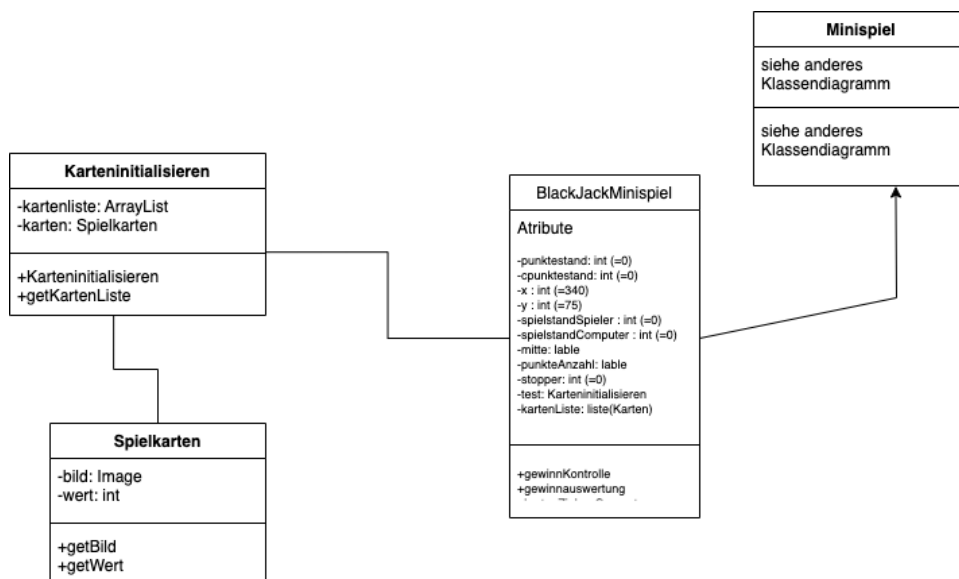


Abb. Klassendiagramm BlackJack Minispiel

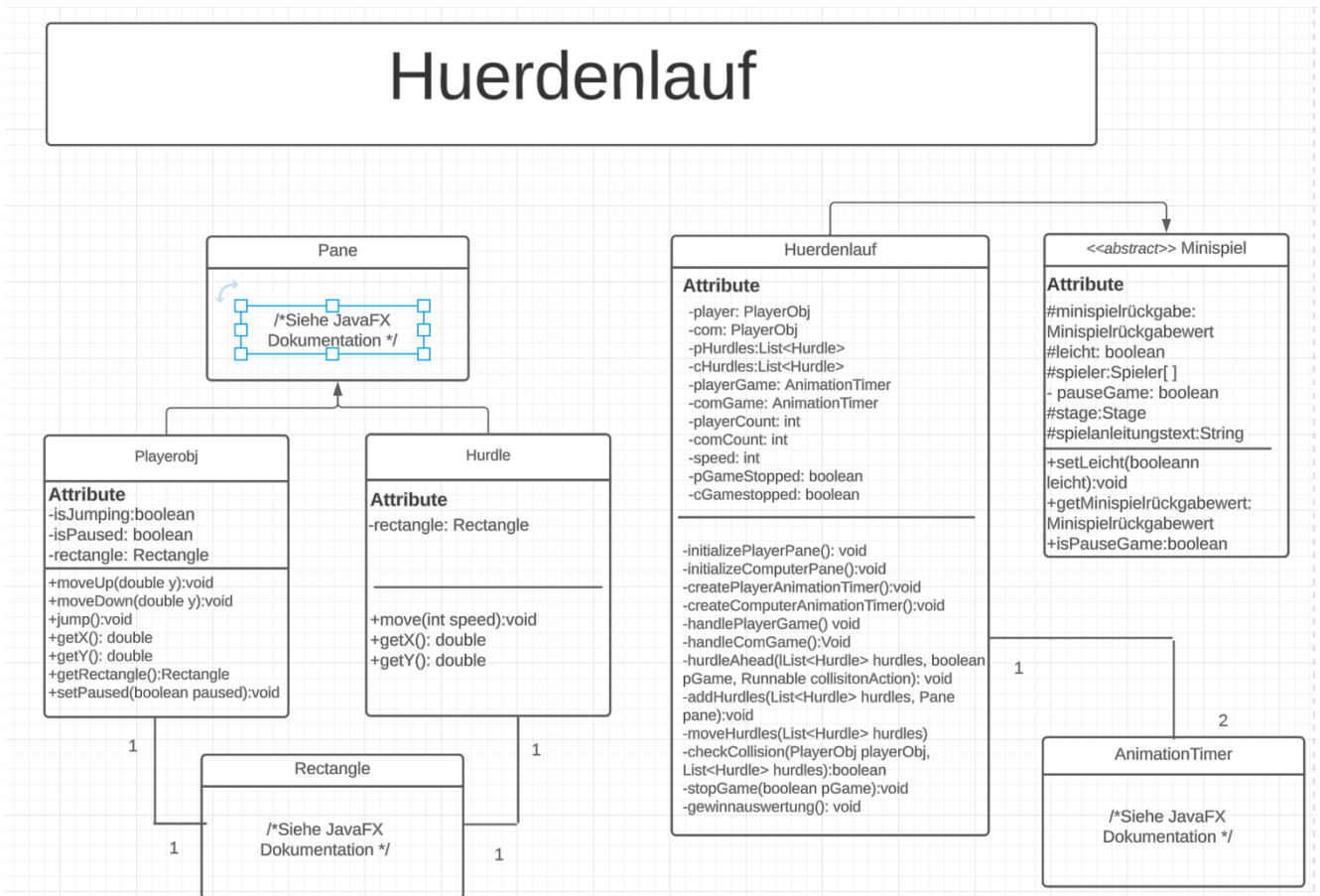


Abb. Klassendiagramm Huerdenlauf

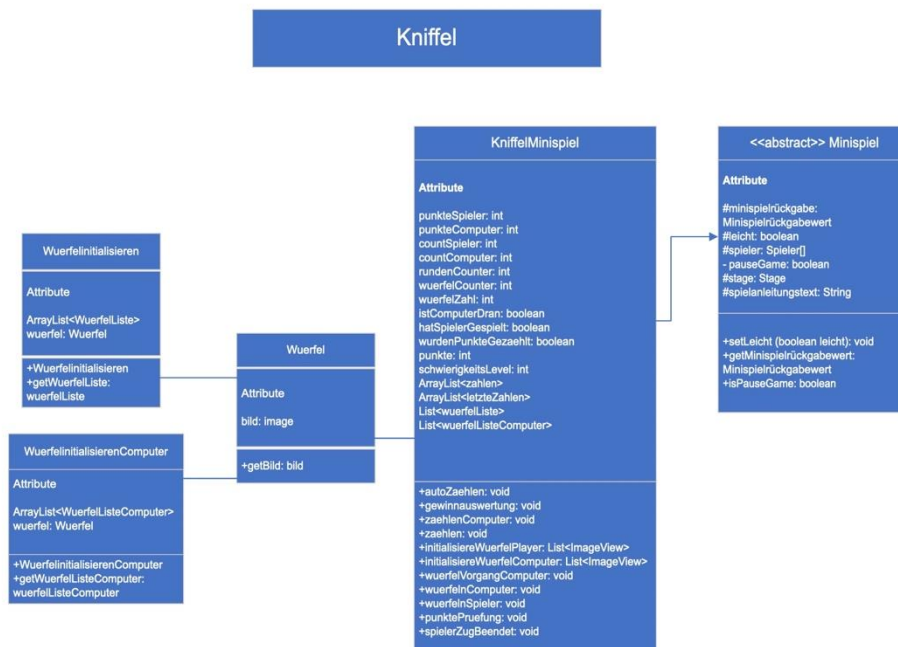


Abb. Klassendiagramm Kniffel Minispiel

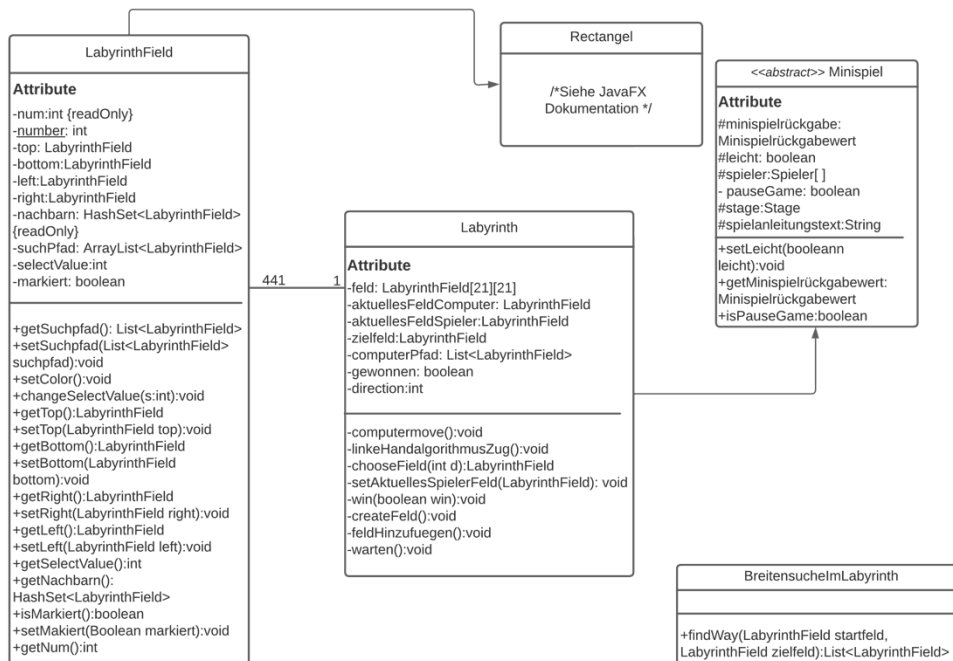


Abb. Klassendiagramm Minispiel Labyrinth

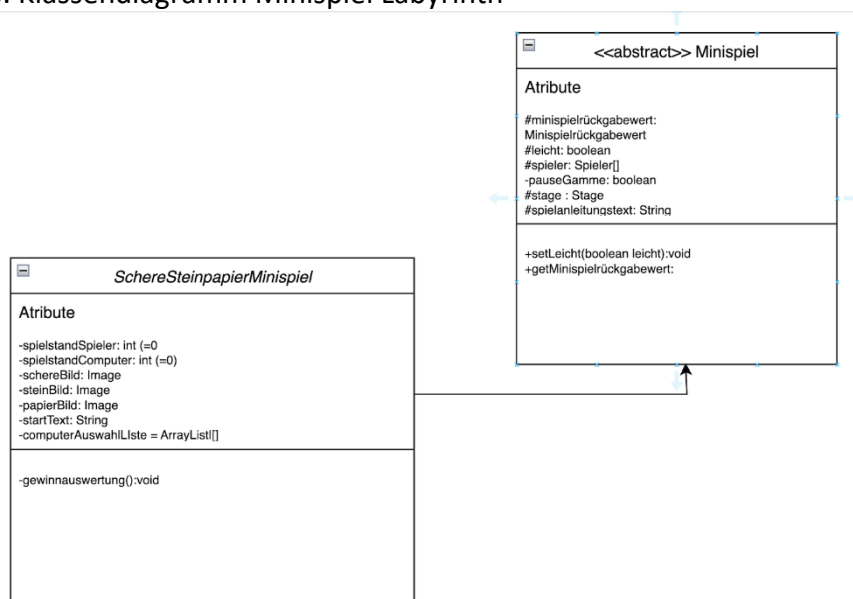


Abb. Klassendiagramm SchereSteinPapier Minispiel

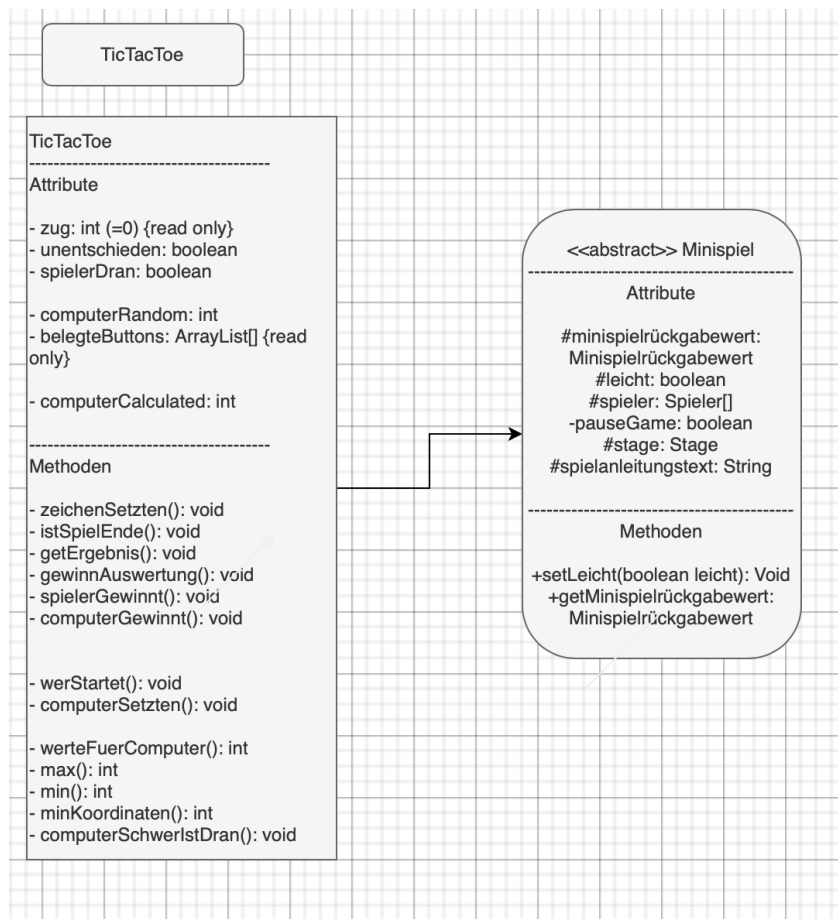


Abb. Klassendiagramm TicTacToe Minispiel

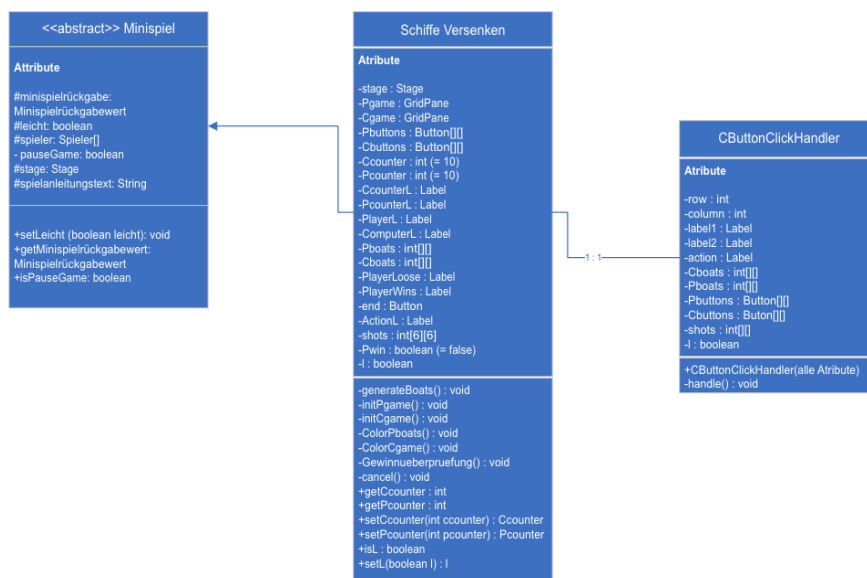


Abb. Klassendiagramm Schiffe Versenken