



Höhere Fachschule
Südostschweiz

Meine Schule. Meine Zukunft.

**NDS HF Applikationsentwicklung
Modul 1 Grundlagen Softwareentwicklung**

Dokumentation Semesterarbeit

„Vier Gewinnt“

Marc Spescha
Sascha Fluor
Daniel Nägeli

Chur, 31.01.2020

Management Summary

Mit dieser Arbeit haben wir das Spiel Vier Gewinnt digital mit Java und JavaFX realisiert. Ziel der Arbeit war es, dass im Unterricht erlernte zu vertiefen und erweitern. Dafür haben wir das Projekt im Sinne von Scrum gemanaged und für die Versionsverwaltung kam Git zum Einsatz. Der Programmcode wurde mit IntelliJ geschrieben und mittels JUnit geprüft.

Bei der Umsetzung der Vier Gewinnt App haben wir folgende Features eingebaut:

- Spiellogik mit automatischer Erkennung des Gewinners
- Benutzeroberfläche mit JavaFX und CSS Styles
- laufend aktualisierte Darstellung des Spielstandes
- Soundeffekte
- Text für Spielanleitung aus externer Datei
- Möglichkeit zum Neustart des Spiels

Inhaltsverzeichnis

1 Einleitung, Ziel	3
1.1 Vier Gewinnt.....	3
1.2 Zieldefinition.....	3
2 Ausgangslage	4
2.1 Spielablauf.....	4
2.2 UI-Mockups.....	5
2.3 Backlog.....	6
2.4 Optionales Backlog.....	9
3 Hauptteil	10
3.1 GUI.....	10
3.2 Spiellogik.....	13
3.3 Stoppuhr.....	14
3.4 Unit-Testing.....	15
3.5 Optionale Funktionen.....	16
4 Fazit und Schlussfolgerung	16
5 Abbildungsverzeichnis	17

1 Einleitung, Ziel

1.1 Vier Gewinnt

„Vier gewinnt“ ist ein Zweipersonen-Strategiespiel. Das Ziel des Spiels ist als erster vier der eigenen Spielsteine in eine Linie zu bringen. Dies kann horizontal, vertikal oder diagonal sein. „Vier gewinnt“ wird auf einem senkrecht stehenden hohlen Spielbrett gespielt. In den hohen Zwischenraum werfen die Spieler abwechselnd die Spielsteine. Das Spiel kann unentschieden enden wenn alle Spielsteine aufgebraucht sind und keiner der beiden Spieler eine Viererlinie gebildet hat.

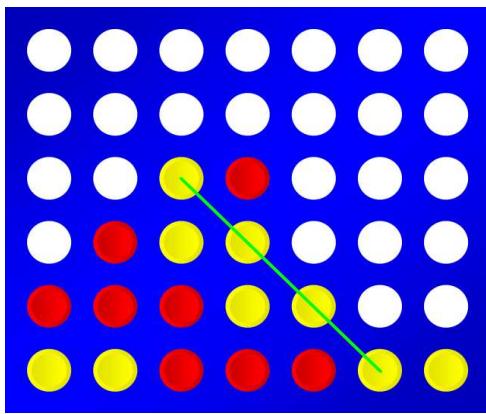


Abb. 1: Beispielbild Vier Gewinnt



Abb. 2: Beispielbild Vier Gewinnt

1.2 Zieldefinition

Wir möchten Vier gewinnt digital umsetzen.

Für die Programmierung der Spiellogik, respektive dem Spielmodell wird Java verwendet. Mit Java können wir plattformunabhängige Anwendungen entwickeln und der Programmcode kann in unterschiedliche Objekte aufgeteilt werden. Durch diesen Vorteil bleiben wir flexibel, was das Ändern oder Erweitern des Codes sowie das Aufteilen der Entwicklung im Team betrifft.

Für das GUI wird mit der JavaFX Bibliothek gearbeitet. JavaFX ist zukunftsorientiert und benutzerfreundlich, die Oberflächen können leichter und verständlicher programmiert werden. Eines der wichtigen Vorteile der JavaFX Bibliothek ist die Möglichkeit der Nutzung von Styles (CSS) die im WWW bekannt ist. Durch die Nutzung dieser CSS-Styles ist es möglich dem Aussehen mehr Individualität zu geben. Des Weiteren wollen wir die Code-Teile mittels Unit-Tests prüfen.

Im Sinne der objektorientierten Programmierung soll der Programmcode in übersichtliche Module, resp. Objekte aufgeteilt werden. Es ist unser Ziel, das GUI getrennt von der eigentlichen Spielelogik zu programmieren. So kann das GUI unabhängig vom restlichen Programmcode ergänzt, geändert oder gar komplett neugestaltet werden.

Voraussichtlich erhalten wir dadurch grob folgende Klassentypen:

- Graphical User Interface (Start-/Spielfenster, Spielanleitung)
- Vier Gewinnt Modell (repräsentiert das Spielbrett in Programmiersprache)
- Vier Gewinnt Test
- Weitere Unterteilungen während der Programmierung

Folgende Fähigkeiten soll das Spiel haben.

- Interaktive Darstellung des Spielbretts mit Maussteuerung
- Spielablauf und Logik von „Vier gewinnt“ implementieren
- Automatische Anzeige aktueller Spieler
- Automatische laufende Spieldauer des laufenden Spiels ab Klick auf den Start Button
- Automatische Anzeige der Anzahl Spielzüge der Spieler
- Nach Gewinn oder Gleichstand darstellen von Spiel-Zusammenfassung (Gewinner, endgültiges Spielbrett, Spielzüge, Spieldauer)
- Via Buttons wechseln zwischen Start-/Spielfenster und Spielanleitung

2 Ausgangslage**2.1 Spielablauf**

Das Spiel wird durch zwei menschliche Spieler per Maussteuerung an einem Rechner, resp. einer Applikations-Instanz gespielt und soll folgendermassen ablaufen:

Beim Start der Java-Applikation öffnet sich das Start-/Spielfenster (siehe Abb. 3). Dieses besteht aus Titel, Spielübersicht und dem Spielbrett. Ausserdem kann über Buttons das Spiel gestartet oder die Spielanleitung angezeigt werden. Das Programm kann über das fensterübliche Kreuz verlassen werden.

Beim Start des Spiels wird das Spielbrett geleert und Spieler 1 beginnt den ersten Spielzug. Im Verlauf des Spiels wird abwechslungsweise der Spieler gewechselt, dieser wählt per Mausklick im Spielbrett die Spalte in beliebiger Höhe aus, worin ein Stein herunterfallen soll. Nach jedem Spielzug wird das Spielbrett mit dem neu gefallenen Stein aktualisiert. Ebenfalls wird nach jedem Spielzug kontrolliert, ob einer der Spieler mit dem zuletzt gelegten Stein gewonnen hat. Solange kein Spieler gewonnen hat, wird der andere Spieler aufgefordert einen Zug zu machen. Sobald ein Spieler vier seiner Steine, horizontal, vertikal oder diagonal gelegt hat, hat dieser die Runde gewonnen. Wenn das eintrifft wird in dem gleichen Fenster angezeigt, welcher Spieler als Sieger in dieser Runde hervorgeht (grüner Kasten in Abb. 3). Gewinnt keiner der Spieler, wird "Unentschieden" ausgegeben. Ein weiterer Spielzug soll anschliessend nicht mehr möglich sein. Durch ein Klick auf Start wird ein neues Spiel gestartet.

2.2 UI-Mockups

Das Spiel soll so umgesetzt werden das innerhalb eines Fensters zwischen 2 Oberflächen gewechselt werden kann. Einer Start-/Spielseite (Abb.3) und einer Spielanleitungsseite (Abb.4). Innerhalb der Start-/Spielseite wird nach Abschluss der Spielrunde das Spielresultat angezeigt.

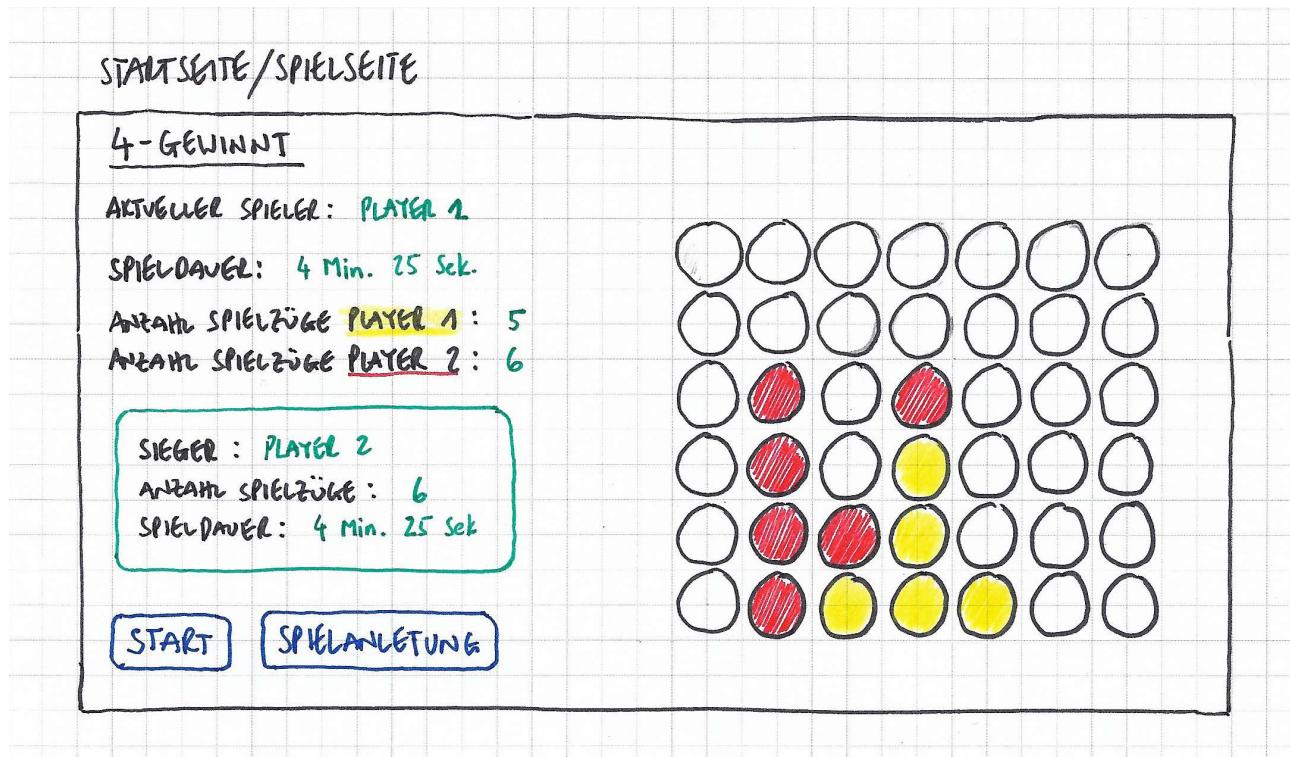


Abb. 3: UI Skizze Start-/Spielseite (Text innerhalb des grünen Rahmens erscheint erst nach Abschluss einer Spielrunde)

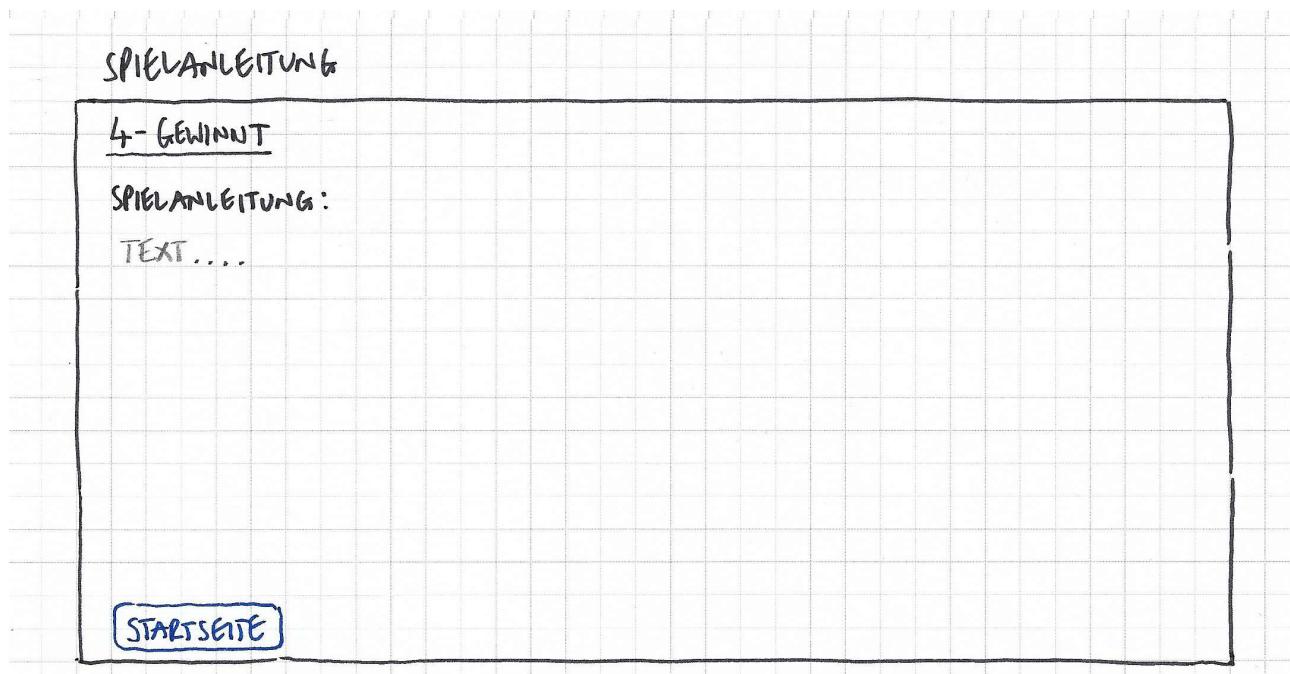


Abb. 4: UI Skizze Spielanleitung

2.3 Backlog

Das Projektmanagement wird im Sinne von Scrum mit einem Kanban-Board durchgeführt. Hierfür wird das durch Github zur Verfügung stehende Tool genutzt.

In der ersten Besprechung des Projekts wurden die User-Stories im Produkt Backlog definiert. Aufgrund unserer geringen Kenntnisse und Erfahrungen im Programmieren war dies nicht leicht. Uns fehlte die Vorstellung, welche Stories wie umgesetzt werden können und welchen Aufwand diese generieren. Da wir in diesem Projekt alle zugleich Produkt Owner sind haben wir beschlossen vorerst die gewählten User-Stories zu überprüfen, mögliche Akzeptanzkriterien und Lösungsansätze zu überlegen und den Aufwand abzuschätzen. In der nächsten Besprechung haben wir das Produkt Backlog festgelegt und uns im 2 Wochen Rhythmus zu den Sprint-Plannings sowie zu den Produkt Inkrementen getroffen. Dabei haben wir einander die neuen Funktionen des Programms vorgestellt und erläutert wie diese gelöst wurden. Nach gutheissen der neuen Funktionen der verschiedenen Gruppenmitglieder wurde anschliessend der nächste Sprint festgelegt und unter der Gruppe aufgeteilt. In den jeweiligen Besprechungen haben wir mögliche Lösungsansätze besprochen und in User-Stories als Tasks eingefügt. Verschiedene Lösungen wurden jedoch aufgrund der geringen Erfahrungen der Mitglieder erst beim Programmieren entdeckt und nachträglich vorgestellt.

Für die Versionsverwaltung der Dateien wurde das im Unterricht erlernte Tool GIT Bash verwendet. Hierbei haben wir auf 4 separaten Branches gearbeitet: der Master sowie die drei Featurebranches von Sascha, Daniel und Marc. Dies hat uns das gleichzeitige Arbeiten ermöglicht, da immer auf den jeweiligen Featurebranch der entsprechenden Person gearbeitet wurde. Bei erfolgreichem Abschluss eines Tasks wurde dieser Branch mittels Pullrequests auf dem Master durch Marc geprüft und ein Merge getätigter. Diese Änderungen auf den Master konnte die Gruppenmitglieder sodann auf Ihre FeatureBranches übernehmen.

Da wir die Tasks unter der Gruppe aufgeteilt haben und darauf geachtet haben, dass keiner am selben Codeteil arbeitet wurde die Verwendung von Forks überflüssig und Mergekonflikte wurden verringert. Falls nun nach Abschluss der Arbeit ein Mitglied oder Nichtmitglied unserer Gruppe auf die Idee kommt Anpassungen zu tätigen, so ist dies mittels Fork zu tätigen.

Während dem Arbeiten mit dem Git-Repository wurde erkannt, dass die Konfiguration mit .gitignore nicht von Anfang an getätigter wurde. Dies führte bei Checkout-, Pull- und Push-Befehlen zwischen Working Directory und git-Repository immer wieder zu Mergekonflikte, da die kompilierten .class Dateien ebenfalls berücksichtigt wurden. Dies wurde nachträglich angepasst, was zu diversen Mergekonflikten führte, welche behoben werden mussten. Es hat sich schliesslich herausgestellt, dass das Löschen der bereits vorhandenen .class Dateien auf dem Git-Repository nicht einfach ist, weshalb in diesem Projekt auf diesen Schritt verzichtet wird. Das Lösen dieses Problems wäre mit einem neuen Repository einfach behoben, jedoch gehen sodann auch die bis anhin getätigten

Versionen (Commits) verloren, was nicht der Idee von Git entspricht. Uns ist mit diesem Fehler die Wichtigkeit dieser Konfiguration für die nächsten Projekte bewusster geworden.

Als Spieler möchte ich, dass zu Beginn ein Fenster erscheint, auf welchem ich das 4Gewinnt Spielbrett sehe.	
Akzeptanzkriterien:	Story ist erfüllt, wenn das Spielbrett mit weissen Kreisen im gewünschten Raster angezeigt wird.
Tasks:	<p>Das Modell-Spiellbrett wird als zwei-dimensionaler String Array [rows][columns] erstellt, wobei die Position [0][0] die obere linke Ecke und die Position [m][n] die untere rechte Ecke ist. Die Anzahl m und n der Reihen und Spalten kann beliebig festgelegt werden.</p> <p>Das Array soll mit Strings gefüllt, welche die 3 möglichen Farben des 4Gewinnt Spielbretts darstellen: "White" für leer, "Red" für Spieler 1 und "Yellow" für Spieler 2. In der Instanziierung werden alle Inhalte auf "White" gestellt.</p> <p>Das Spielbrett wird als LayoutElement in Form eines Grid-Panes im GUI eingefügt. Das GridPane wird mit Buttons befüllt, welche die gleiche Indexierung und Positionierung wie das String-Array haben.</p> <p>Die Hintergrundfarbe der Buttons soll anhand der Strings im Array festgelegt werden.</p>

Als Spieler möchte ich, dass per Knopfdruck auf eine beliebige Position im Spielbrett abwechselungsweise die entsprechende Spalte von unten startend mit den roten oder gelben Steinen befüllt wird.	
Akzeptanzkriterien:	Story ist erfüllt, wenn bei Mausklick abhängig von der Spalte in der entsprechenden Spalte das untere Feld rot oder gelb wird.
	Story ist erfüllt, wenn bei nochmaligem Mausklick auf gleicher Spalte das nächst oberem weissem Feld rot oder gelb wird.
	Story ist erfüllt, wenn die Farbgebung der Felder abwechselungsweise zwischen rot auf gelb wechselt.
Tasks:	<p>Die Buttons im GridPane sollen mit einem Eventhandler versehen werden, welcher bei einem Mausclick eine Methode z.B. Drop mit dem Argument des entsprechenden Spaltenindexes ausführt.</p> <p>Die Methode Drop iteriert bei der entsprechenden Spalte von unten nach oben (Index m bis 0) hinunter. Sobald er einen Stringinhalt "white" entdeckt, ändert er dessen String auf "Red" oder "Yellow", je nachdem welcher Spieler dran ist.</p> <p>Die Spieler werden mit einfachen int-datentypen 0 (Spieler 1) und 1 (Spieler 2) indexiert. Ebenfalls wird ein entsprechendes String Array mit den Inhalten [0] "Red" und [1] "Yellow" erstellt. Spieler 1 (Index [0] und somit Farbe Rot) beginnt. Nach jedem Aufruf der Methode Drop wird der Spielerindex getauscht.</p>

Als Benutzer möchte ich, dass nach jedem Spielzug geprüft wird, ob Spieler 1 oder Spieler 2 vier Steine seiner Farbe horizontal, vertikal oder diagonal aufgereiht hat.	
Akzeptanzkriterien:	Story ist erfüllt, wenn nach jedem Spielzug eine Methode false oder true zurückgibt, wenn eine Anreihung von 4Steinen geglückt oder nicht geglückt ist.
Tasks:	Es müssen vier verschiedene Optionen der Ausrichtungen geprüft werden: horizontal, vertikal, slash-diagonal und backslash-diagonal. Alle Ausrichtungen müssen nach jedem Spielzug geprüft werden.
	Für die Überprüfung wird die String.contains()-Methode von Java genutzt. Basierend auf dem Spalten- und Reihen-Indexes des letzten Drops werden die Strings des Spielbrett-Arrays in den verschiedenen Ausrichtungen zum zuletzt geänderten String zusammengeführt.
	Beinhaltet einer der zusammengeführten Strings in den vier Ausrichtungen vier Mal direkt nacheinander die Spieler-Farbe, so soll eine entsprechende Methode den Boolean-Wert true zurückgeben. Wenn nicht, soll sie false zurückgeben.

Als Benutzer will ich, dass das Spiel/Start-Fenster weitere Informationen beinhaltet: Ein Titel, der aktuelle Spieler, die fortlaufende Spieldauer, die Anzahl Spielzüge pro Spieler	
Akzeptanzkriterien:	Story ist erfüllt, wenn im Fenster der entsprechende Titel des Spiels ersichtlich ist.
	Story ist erfüllt, wenn ersichtlich ist, ob Spieler 1 oder 2 an der Reihe ist.
	Story ist erfüllt, wenn eine dynamische Stoppuhr, welche ab Spielbeginn startet und die Sekunden und Minuten aufzählt, ersichtlich ist.
	Story ist erfüllt, wenn für Spieler 1 und Spieler 2 die jeweiligen aufgezählten Spielzüge dargestellt werden.
Tasks:	<p>Das Spielfenster in einem Boarderpane ausbauen: anhand von Labels den Titel im Top darstellen, die Informationen zu akt. Spieler, Anzahl Spielzüge und Spieldauer in einem Gridpane in der linken Spalte darstellen.</p> <p>Das Label mit dem aktuellen Spieler mit einem ChangeListener bestücken, welches aktualisiert wird, sobald der Modell-Wert des aktuellen Spielers wechselt.</p> <p>Eine separate Klasse Stoppuhr erstellen, welche das Interface Task implementiert. Anhand vom Date-Datentyp eine Methode erstellen, welche die Spieldauer als String in Minuten und Sekunden zurückgibt.</p> <p>Im GUI einen Service starten, welcher das vom Task erhaltene String mit dem Spieldauer-Label dynamisch verbindet (bind).</p> <p>Das Spielmodell mit zwei weiteren Variablen als int-Datentyp ergänzen und in einem ArrayList "playerMoves" mit der Indexierung [0] und [1] wie aktueller Spieler zusammenführen.</p> <p>Nach jedem Aufruf der Drop-Methode die Anzahl Spielzüge des jeweiligen Spielers aufzählen. Im GUI diese Information mit einem ChangeListener aktuell halten.</p>

Als Benutzer möchte ich, dass sobald ein Spieler gewinnt, keine weiteren Steine mehr gelegt werden können und das Spiel-/Startfenster soll folgende Information ausgeben: Gewinner, Anzahl Spielzüge des Gewinners und die Spieldauer.	
Akzeptanzkriterien:	Story ist erfüllt, wenn bei gewonnenem Spiel bei Mausklick im Spielfeld keine Steine mehr gelegt werden.
	Story ist erfüllt, wenn bei gewonnenem Spiel eine weitere Anzeige erscheint, welche den Gewinner, Anzahl dessen Spielzüge und die Spieldauer bis zum entscheidenden Spielzug darstellt.
Tasks:	<p>Die Buttons sollen nur dann auf einen Klick reagieren, solange es keinen Gewinner (Methode: hasAWinner: false) hat. Kondition auf dem Eventhandler des Buttons integrieren.</p> <p>Sobald die Methode des Modells hasAWinner true ist, soll im linken Feld des BoarderPanes ein zusätzlicher Gridpane erscheinen, welcher die Informationen Gewinner, Anzahl Spielzüge und schlussendliche Spielzeit darstellt.</p> <p>Sobald die Methode des Modells hasAWinner true ist, soll die Stoppuhr gestoppt werden. Kondition in der Stoppuhr-Klasse hinzufügen, welche sicherstellt, dass diese nur läuft, solange hasAWinner false ist.</p> <p>Die aktuellen "Werte" der Informationen sind gleich derer von Anfang an gezeigten Informationen.</p>

Als Spieler möchte ich, dass das Spiel-/Startfenster zwei Buttons erhält: Start und Spielanleitung. Per Klick auf Start soll das Spiel gestartet resp. Neu gestartet werden. Per Klick auf die Spielanleitung wird ein separates Fenster mit der Spielanleitung geöffnet.	
Akzeptanzkriterien:	Story ist erfüllt, wenn das Spielfenster zwei Buttons enthält, welche per Mausklick eine Aktion ausführen. Der Button "Spielanleitung" ändert die ganze Scene zu einem Fenster mit Text. Der Button "Start" startet das Spiel neu.
	Story ist erfüllt, wenn bei Klick auf Button "Spielanleitung" im selben Fenster die Scene ändert und einen Text mit der Spielanleitung angezeigt wird.
	Story ist erfüllt, wenn bei Klick auf Button "Start" das Spielbrett geleert wird und die Informationen (aktueller Spieler, Anz. Spielzüge, Spieldauer) wieder den Anfangswert zurückgestellt werden
	Story ist erfüllt, wenn die Spiellogik nach neustarten des Spiels weiterhin funktioniert.
Tasks:	<p>Im unteren Teil des BoarderPanes zwei Buttons hinzufügen und mit EventListener versehen. Bei Klick auf "Spielanleitung" soll die Scene auf eine zu erstellende Scene mit Spielanleitung wechseln.</p> <p>Um das Neustarten des Spiels zu gewährleisten müssen diverse Code-Teile, welche für die Instanziierung des Spielbretts und der Informationen zuständig sind in einer neuen Methode ausgelagert werden. Bei Klick auf dem Button soll diese Methode ausgeführt werden.</p>

2.4 Optionales Backlog

Beim Brainstorming zur Umsetzung der Semesterarbeit hatten wir einige Ideen, die wir aus Zeitgründen zurückgestellt haben. Falls die Zeit doch ausreicht, würden wir die eine oder andere Idee eventuell noch ins Spiel einbauen.

- Audio (z.Bsp. Sound wenn Spielstein fällt / Hintergrundmusik)
- Animation (z.Bsp. Mouse-over - Steine in Viererlinie blinken)
- Grafische Umsetzung (z.Bsp. Retro Look)
- Spieler können vor Spielstart mit Namen einloggen (Datenbank)
- Spielstand wird in Bestenliste eingetragen (Top10 / Datenbank)
(Rang, Name, Anzahl Spielzüge, Spieldauer bis Sieg, Datum)
- Das Spiel kann von zwei Rechner/Clients aus über LAN gespielt werden

Entwurfsskizzen für Setup und Top10 Fenster:

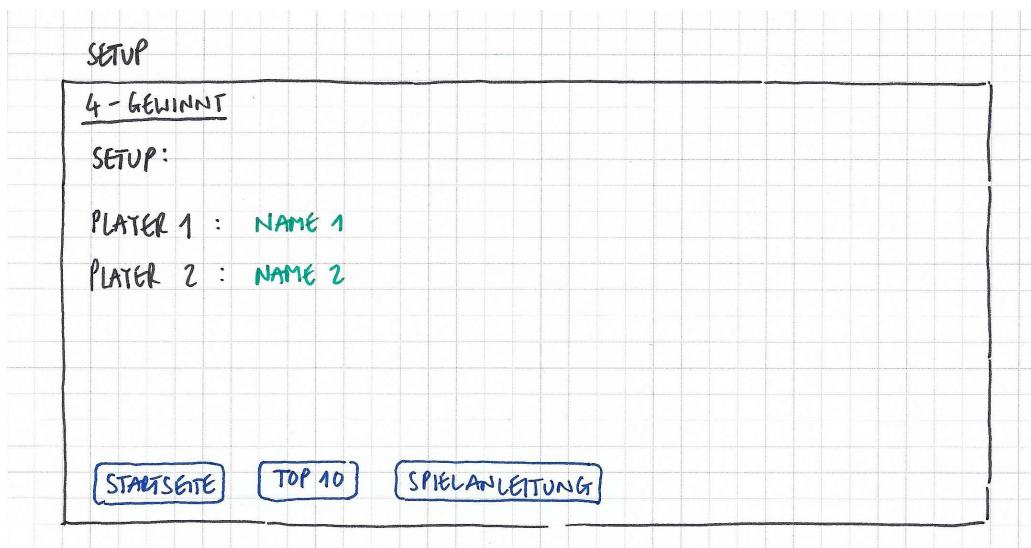


Abb. 5: UI Skizze optionale Setup-Scene

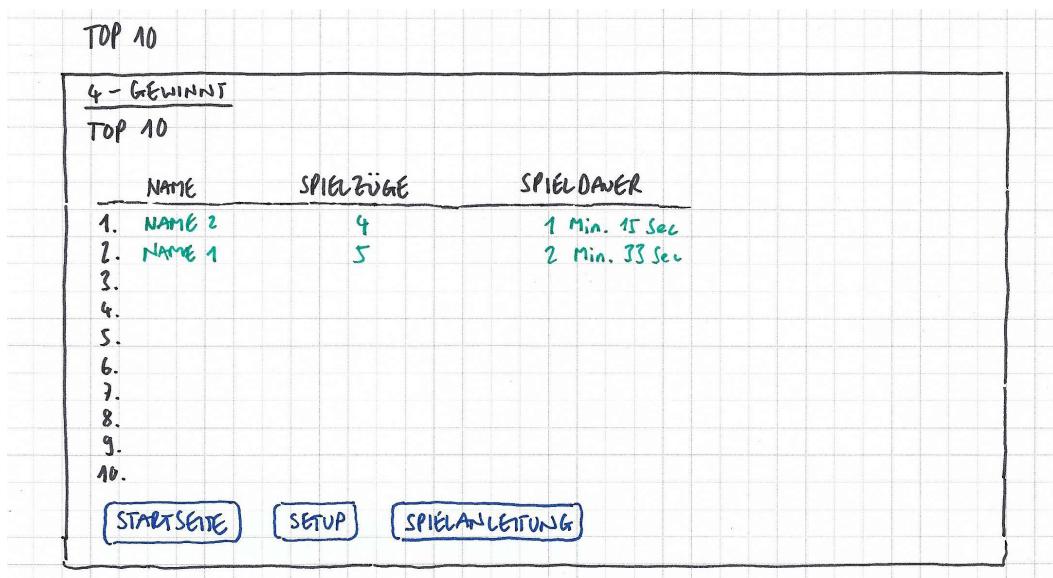


Abb. 6: UI Skizze optionale Top 10 Liste aus Datenbank

3 Hauptteil

3.1 GUI

Planung

Für das Anforderungsdokument haben wir von Hand Skizzen erstellt, um grob aufzuzeigen wie das GUI aussehen könnte. Dies war sehr hilfreich damit alle Gruppenmitglieder wussten wie das Endprodukt aussehen könnte. Wir haben es uns aber offen gelassen dies im Verlauf der Entwicklung anzupassen wenn sich herausstellen würde, dass die erste Skizze doch nicht optimal ist.

Schlussendlich blieben wir jedoch beim Entwurf und haben die finale Struktur der App gemäss ursprünglicher Skizze umgesetzt.

Aufbau

Das Spiel wurde so aufgebaut, dass innerhalb eines Fensters (Stage) zwischen 2 Oberflächen (Scenes) gewechselt werden kann. Einer Start-/Spielseite und einer Spielanleitungsseite. Die beiden Seiten wurden in der Grundstruktur mit BorderPane Elementen erstellt. Innerhalb des BorderPane wurde mit JavaFX GridPanes, Vbox, Hbox, Labels, Text und Buttons gearbeitet. Das Spielbrett (GameBoard) wurde in eine externe Klasse ausgelagert. Dies mit der Absicht die GUI Klasse übersichtlicher und schlanker zu machen.

Styling

Das optische Erscheinungsbild der 4-Gewinnt App wurde im ersten Schritt mit CSS Inline Styles innerhalb des Codes erreicht. Schnell wurde klar, dass dieses Vorgehen den Code „aufbläst“, unübersichtlich macht und auch Anpassungen umständlich zu tätigen sind da immer der ganze Code durchsucht werden musste. Wir haben dann den Grossteil des Stylings in eine externe CSS Datei ausgelagert, um flexibler bei Anpassungen zu sein.

Aufgrund des gewählten Lösungsweges für die Spiellogik konnte das styling nicht komplett in die externe CSS Datei übernommen werden.

Navigation

Die Navigation zwischen Start-/Spielseite und Spielanleitungsseite wurde mit Buttons erreicht welche mit setOnAction Eventhandler ausgestattet sind und damit zwischen den zwei Scenes wechseln können.

Restart Button

Um ein Spiel neu zu starten wurde ebenfalls ein Button eingesetzt. Der Button ruft mit einem setOnAction Eventhandler die Methode newGame auf welche das Spiel neu aufbaut und startet.

Spielanleitung Text

Der Spielanleitungstext wird mit BufferedReader aus einer externen Datei in einen String eingelesen und in ein Text Fenster (javafx.scene.text.Text) ausgegeben. Aufgrund der vorgegeben Textfensterbreite wird der Text automatisch umgebrochen.

WinnerInfo

Wenn einer der beiden Spieler gewinnt wird eine Spiel-Zusammenfassung eingeblendet.

Um dies zu erreichen wird das Gridpane bei Spielbeginn mit der setVisible Methode auf false gestellt, also unsichtbar. Beim Sieg eines Spielers wird dieser Wert wieder auf true geändert und die Zusammenfassung eingeblendet.

Folgend sind die Layouts für den Aufbau der App zu sehen. In der Entwurfsphase wurden diese von Hand erstellt, später wurde mit Adobe XD gearbeitet.

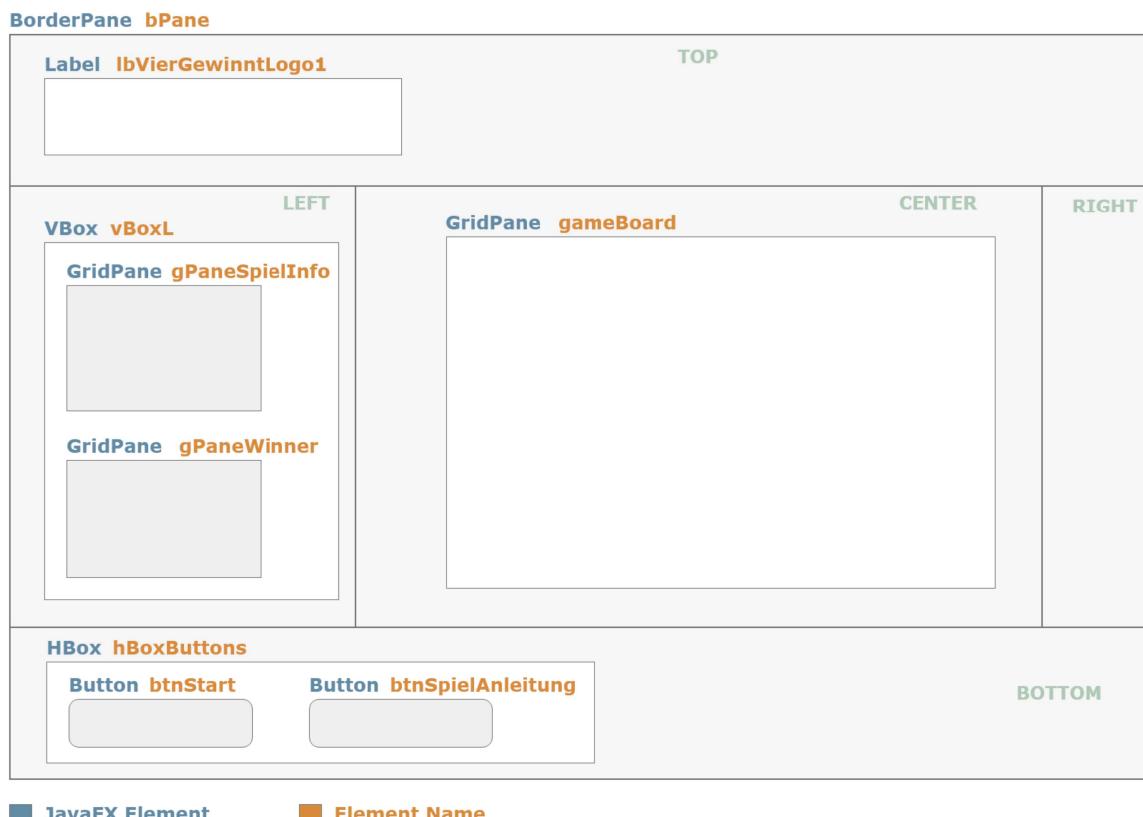


Abb. 7: Skizze JavaFX Layout Aufbau des Star-/Spielseite

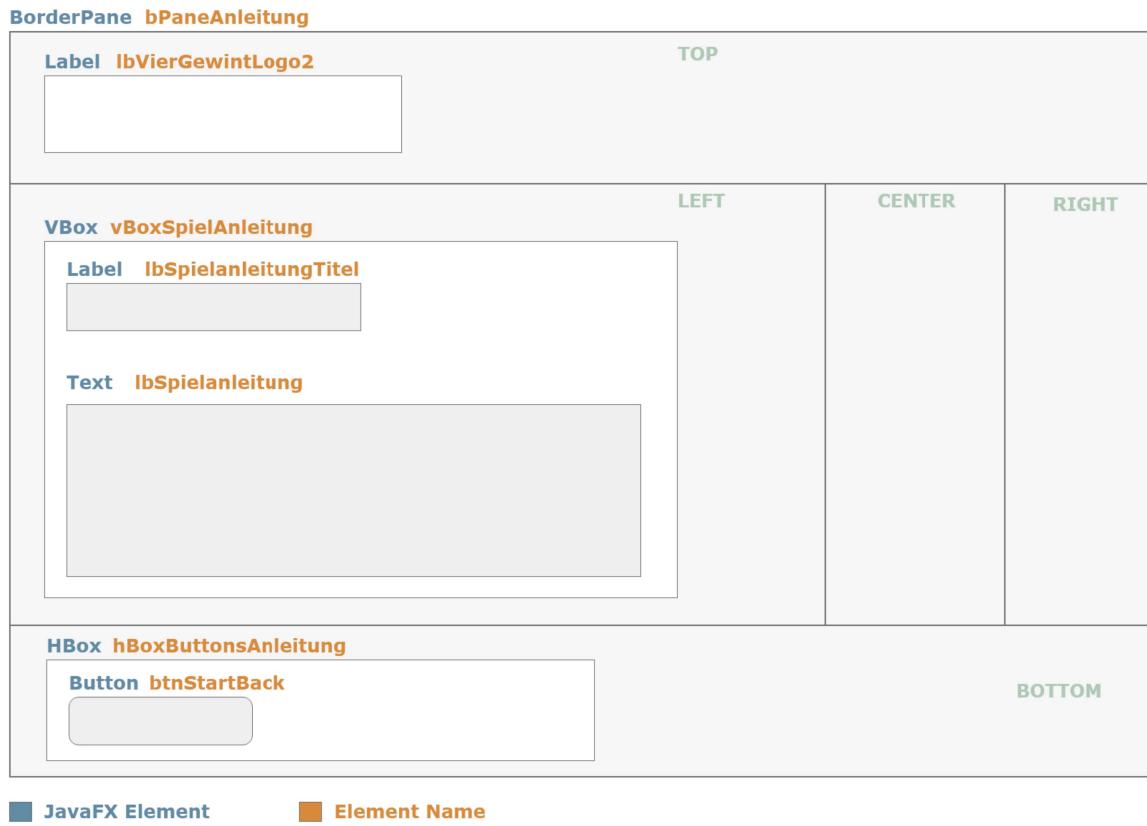


Abb. 8: Skizze JavaFX Layout Aufbau des Spielanleitungsseite

Folgend zu sehen das finale Erscheinungsbild der Vier Gewinnt App.

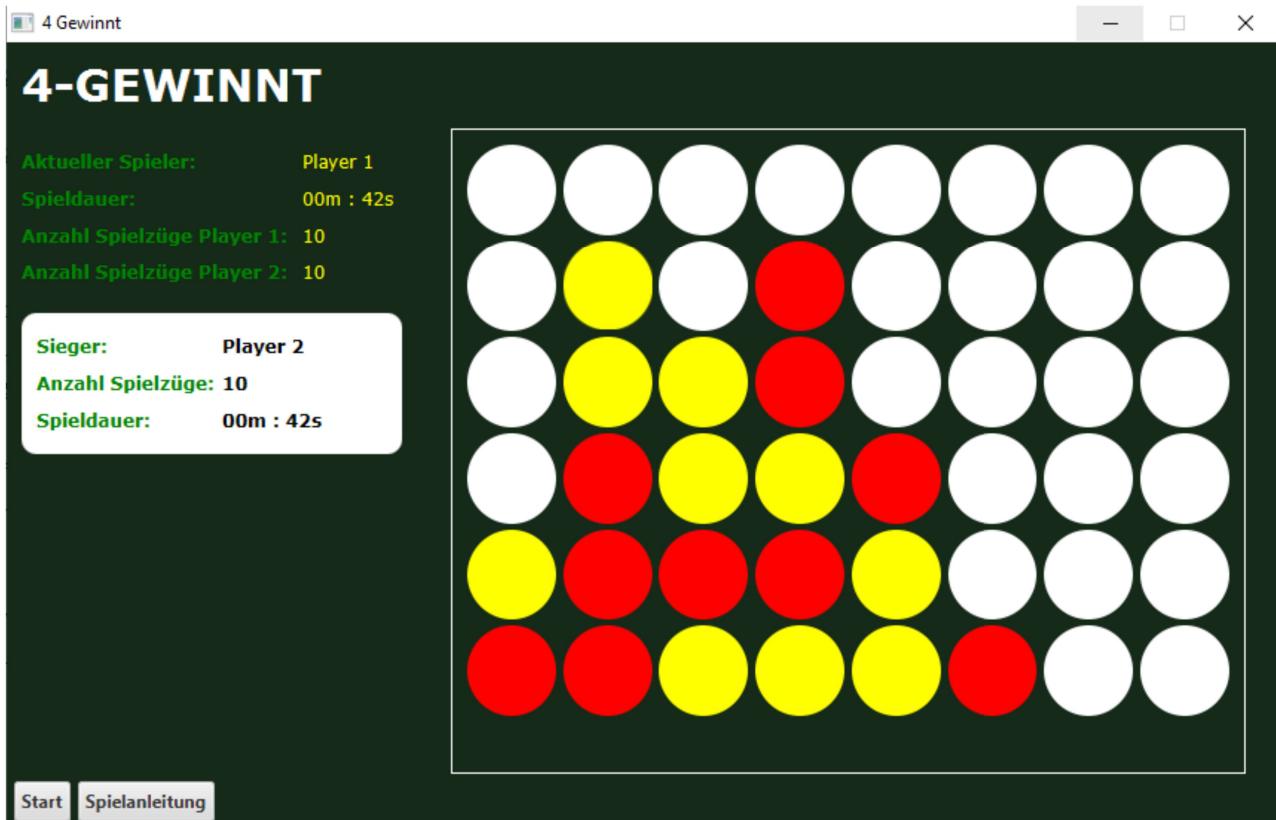


Abb. 9: Das finale Vier Gewinnt Spielfenster

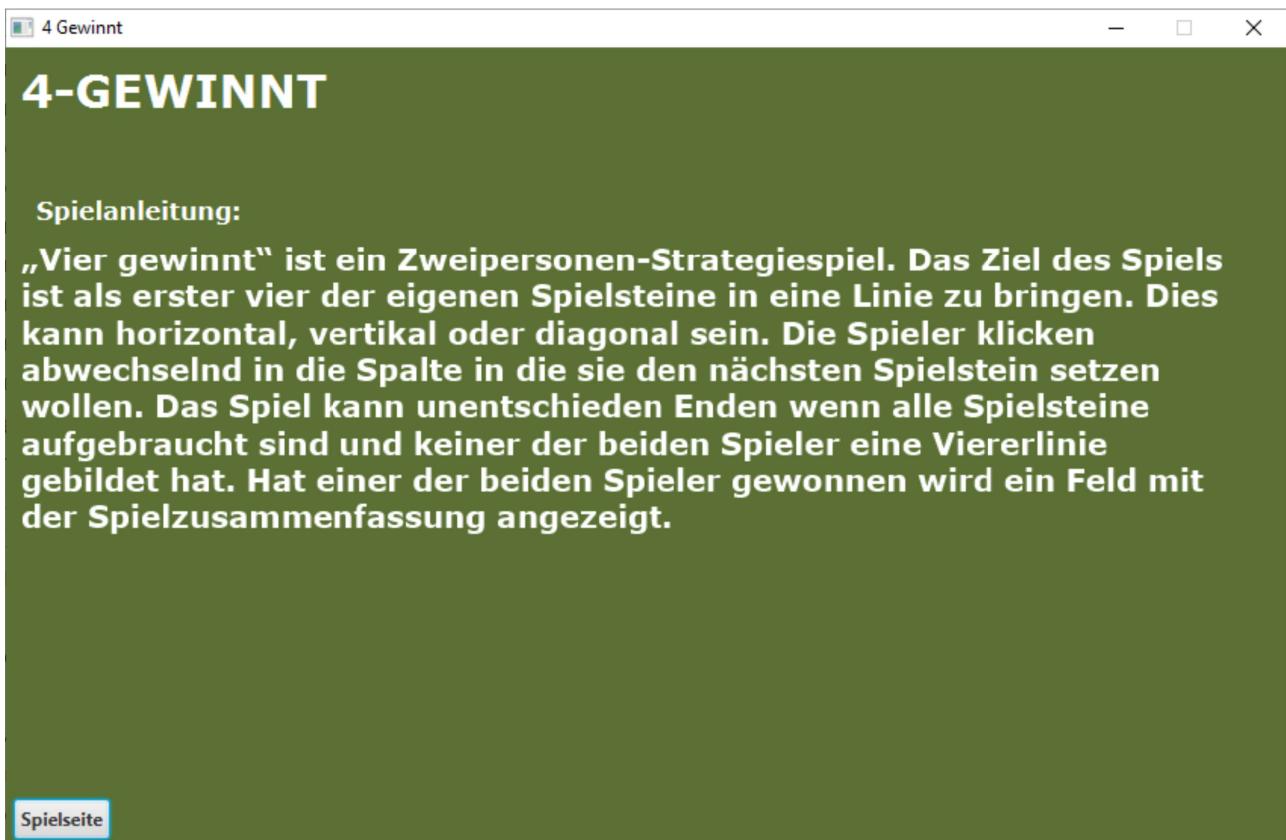


Abb. 10: Das finale Vier Gewinnt Spielanleitungsfenster

3.2 *Spiellogik*

Aufbau Spielbrett und Spieler

Aufgrund der zweidimensionalen Rastergeometrie des Spielbrettes haben wir uns entschieden das Modell im selben darstellbaren Raster einer zweidimensionale ArrayListe zu verwirklichen. Dabei sind die Positionen der Felder im Spielbrett durch die Indexierung der ArrayListe festgelegt. Während die erste Dimension der ArrayListe die Reihen darstellen sind in der zweiten Dimension die Spalten aufgereiht. So kann auf jegliche Position im Spielbrett über die Indexierung zugegriffen werden. Da die Felder des Spielbrettes mit Farben dargestellt werden, wird die ArrayListe mit Strings: "White", "Red", "Yellow" gefüllt, wobei "White" leere Felder darstellt. Die gewählten Strings können so sogleich für das Styling des GUIs verwendet werden. Bei der Instanziierung der ArrayListe wird diese sogleich komplett mit dem String "White" gefüllt.

Des Weiteren haben wir für den Spieler ein int-Datentyp festgelegt, welcher während dem Spiel zwischen 0 (für Spieler 1) und 1 (für Spieler 2) wechseln soll. Die entsprechende Spielerzahl nutzen wir als Indexierung für die Auswahl zwischen den Spielerfarben "Red" und "Yellow", welche in einer separaten ArrayListe instanziert werden. Auch für die Aufzählung der Spielzüge wurde eine weitere ArrayListe mit zwei int-Datentypen (wiederum [0] für Spieler 1, [1] für Spieler 2) erstellt.

Die Methoden

Der Spielzug wird in der Methode drop realisiert. Dieser benötigt als Argument die gewünschte Spalte, worin ein Stein gelegt werden soll. Die Methode prüft in einer Schlaufe in der entsprechenden Spalte der ArrayListe von der untersten Reihe aufwärts ob ein String mit "White" vorhanden ist, ist dies der Fall so befüllt Sie diesen mit der Farbe des aktuellen Spielers. Ist die ganze Spalte bereits mit den Strings "Red" oder "Yellow" befüllt gibt das Programm in der Console "Column is full" aus. Ausserdem inkrementiert sie beim aktuellen Spieler seine Spielzüge in der ArrayListe der Spielzüge.

Nach jedem Spielzug soll die Methode hasAWinner ausgeführt werden. Diese prüft mittels vier weiterer Methoden: horizontal-, vertikal-, slashDiagonal- und backSlashDiagonalAddition ob mit dem letzten Spielzug eine 4er Reihe erzielt wurde und gibt den boolean true oder false zurück. Zur Überprüfung des Gewinners wurde ein Prinzip von Sylvain Saurel aus seiner Webseite medium.com/@ssaurel angewandt. Die vier erwähnten XXXAddition Methoden geben dabei ausgehend von der Position des letzten drops in der jeweiligen Ausrichtung eine Zusammenführung der Strings aus der ArrayListe zurück. So werden z.B. alle Strings der letzten Reihe (horizontalAddition) von links nach recht zu einem String zusammengeführt. Die Methode hasAWinner überprüft sodann mittels Konditionen ob dies dadurch erhaltene Strings der vier Ausrichtungs-Additionen in sich vier aufeinanderfolgende Wörter der aktuellen Spielerfarbe Red oder Yellow enthältet.

3.3 Stoppuhr

Die Stoppuhr wurde aus dem gleichen Grund wie die Aufzählung der Spielzüge eingerichtet. Diese beiden Angaben würden es ermöglichen eine Datenbank mit Highscores zu verwirklichen. Für die Stoppuhr wurde wiederum eine separate Klasse erstellt. Da diese starten und stoppen soll, sobald ein Spiel erstellt, respektive ein Spieler gewonnen hat, muss diese als Argument das Spiel und die entsprechende Klasse ConnectFour als Argument erhalten. Die Klasse der Stoppuhr arbeitet mit dem Datentyp Date(), welche in einer while-schlaufe die Differenz zwischen aktueller Zeit und die Startzeit in Minuten und Sekunden ausgibt.

Die Stoppuhr wurde mit einer while-schlaufe erstellt, welche erst zu Ende geht sobald ein Spieler gewonnen hat. Deshalb muss die Stoppuhr auf einem anderen Thread als die Spiellogik, resp. das GUI laufen, weil der Prozess ansonsten in der while-schlaufe hängen bleibt. JavaFX bietet hierfür eine Lösung mit der Bibliothek concurrent. Diese Bibliothek bietet die Möglichkeit Code auf einem anderen als der JavaFX Thread auszuführen und zu kontrollieren. Hierfür lassen wir unsere Stoppuhr Klasse von der Klasse Task vom Typ <String> erben, da der Task schliesslich einen String mit der Zeitausgabe in Minuten und Sekunden ausgeben soll. Für die Vererbung muss die Methode call() implementiert werden, worin der auszuführende Code programmiert wird. Schliesslich wird in der Main-Methode, resp. Bei uns im GUI ein Service gestartet werden, welcher in sich das Task steuern kann.

3.4 Unit-Testing

JUnit ist ein einfaches Framework zum Programmieren von wiederholbaren Tests. Es wurde 1999 von Kent Beck entwickelt. Es ist in praktisch allen modernen IDE's integriert und wird vor allem zum Testen von einzelnen Methoden / Komponenten verwendet.

Mit Annotationen lassen sich die Methoden mit @Test kennzeichnen und die Methode kann nun als Testklasse ausgeführt werden.

Die Methode kann nun gestartet werden und zeigt an ob Fehler vorhanden sind oder ob der Test grün wird. Grün wird der Test, wenn die angegebene Erwartung mit dem Output der Methode übereinstimmt. Ist dies nicht der Fall wird der Test rot und man muss schauen wo die Unterschiede liegen.

ConnectFourTest Klasse:

Für alle wichtigen Methoden der ConnectFour Klasse wurden Unit Tests mit JUnit 5.4 erstellt. Noch mehr Tests empfanden wir nicht für nötig, da die einzelnen Methoden so trivial sind, dass in ihnen nicht mehr Fehler vorkommen können. Mock-Objekte waren keine nötig.

Testabdeckung:

Die Testabdeckung erscheint in Prozent auf den ersten Blick etwas niedrig:

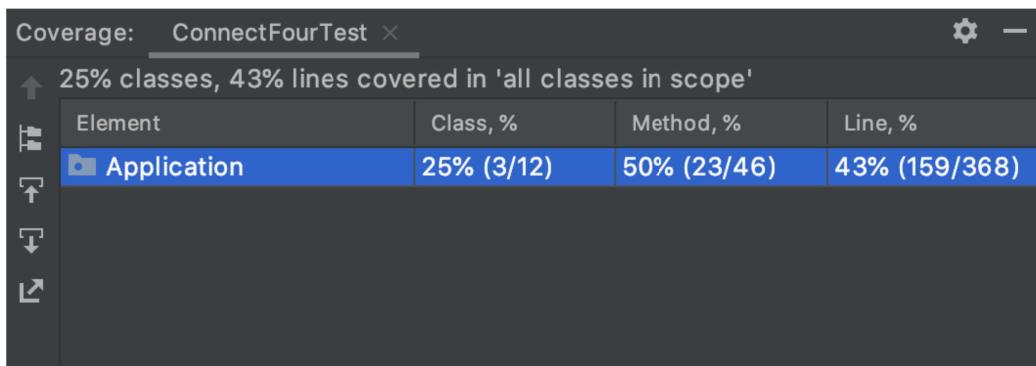


Abb 11: Coverage Übersicht aus IntelliJ

Dies resultiert daraus, dass wir nur die ConnectFour Klasse getestet haben. Schauen wir die Tests im Detail an sehen wir, dass die ConnectFour Klasse eine sehr gute Testabdeckung erlangt hat:

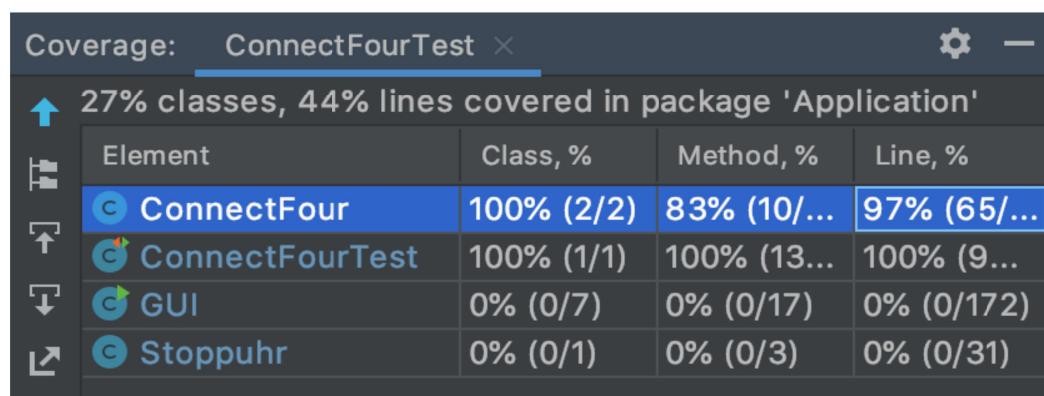


Abb 12: Coverage Übersicht aus IntelliJ

Tests für das GUI wurden keine erstellt, da dies eher das Layout und die Designelemente überprüft. Des weiteren müsste ein weiteres Framework für JavaFX Tests eingebunden werden, und zwar TestFX. Da wir dies jedoch noch nicht weiter behandelt haben, haben wir dies hier nicht angewendet.

3.5 Optionale Funktionen

Soundeffekte:

Die Soundeffekte bei einem drop sowie bei einem gewonnenen Spiel konnten leicht mit der Klasse AudioClip von JavaFx umgesetzt werden. Diese werden abgespielt sobald ein Button gewählt wird und prüft vorgängig, ob es sich um einen gewöhnlichen Drop handelt oder ob es bereits einen Gewinner hat.

Datenbank:

Auf dem featurebranch Vinkja wurden die ersten Schritte für eine Einbindung einer Datenbank realisiert. Die Gewinnerinformationen können mit einzugebenden Namen auf die Datenbank übermittelt werden. Ebenfalls kann die Datenbank mittels TableView auf einer separaten Scene angesehen werden.

Da die Funktionen noch nicht ausgereift sind, wurde auf einen Merge auf dem Master verzichtet.

4 Fazit und Schlussfolgerung

Zielerreichung

Unsere gesetzten Ziele konnten wir vollständig realisieren. Ebenfalls konnten wir diverse optionale Ziele umsetzen, wobei die Zeit für die vollständige Implementation nicht ausgereicht hat. So haben wir eine Stoppuhr, sowie die Aufzählung der Spielzüge realisieren können. Diese sind Teil der Highscore Datenbank. Dieser konnte zwar funktional auf dem Working Directory umgesetzt werden, sollte jedoch so umgebaut werden, dass es auf allen ausführenden Clients funktioniert. Zudem hat uns während der Arbeit am Code vom GUI der repetitive Text für das Styling nicht gefallen. Wir haben daraufhin das Styling auf ein separates CSS-File ausgelagert.

Herausforderungen

Mit den bescheidenen Vorkenntnissen war es zu Beginn schwierig den Aufbau und die Struktur vom Endprodukt vorzustellen sowie einen möglichen Lösungsansatz zu wählen. Die Aufteilung auf verschiedene Klassen und ihr Zusammenspiel wurde eher durch Versuche eruiert und folgte keinem vorgängig definierten Ansatz. Auch das Auslagern auf eine CSS-Datei kam so zustande. Die Versionsverwaltung mit Github stellte uns Herausforderungen in der Anwendung. Das Verständnis von Github müsste bei allen Gruppenmitgliedern besser sein um Fehler zu vermeiden.

Verbesserungsansätze

Kommentare

Wenn mehrere Personen an einem Projekt arbeiten ist es wichtig Code-Teile zu kommentieren. Aber auch für einen selbst ist dies sehr hilfreich.

Konventionen

Am Anfang des Projekts wurden Code-Teile teils Englisch teils Deutsch benannt. Dies sollte von Anfang klar festgelegt werden. Bezuglich Github sollten ebenfalls von Beginn an klare Vorgehensweisen definiert werden.

Schlussfolgerung

Die Zusammenarbeit im Team verlief dank den klaren User-Stories sehr gut. Die Sprint-Plannings wurden grossteils erfolgreich und gemäss Terminplanung abgearbeitet. Durch diese Arbeit wurde unser Wissen nochmals erweitert und vertieft. Wir können uns gut vorstellen das Spiel zukünftig weiterzuentwickeln. Zum Beispiel wäre die Realisierung von Multiplayer oder die Umsetzung in ein Browserfähiges Spiel spannend.

5 Abbildungsverzeichnis

Abb.Nr.	Beschrieb	Quelle	Seite
1	Beispielbild Vier Gewinnt	https://de.wikipedia.org/wiki/Vier_gewinnt#/media/Datei:Connect4_Wins.PNG	3
2	Beispielbild Vier Gewinnt	https://de.wikipedia.org/wiki/Vier_gewinnt#/media/Datei:Connect-four.jpg	3
3	UI Skizze Start-/Spielseite	Erstellt durch Projektteam	5
4	UI Skizze Spielanleitung	Erstellt durch Projektteam	5
5	UI Skizze optionale Setup-Scene	Erstellt durch Projektteam	9
6	UI Skizze optionale Top 10 Liste aus Datenbank	Erstellt durch Projektteam	9
7	Skizze JavaFX Layout Aufbau des Star-/Spielseite	Erstellt durch Projektteam	11
8	Skizze JavaFX Layout Aufbau des Spielanleitungsseite	Erstellt durch Projektteam	12
9	Finales Vier Gewinnt Spielfenster	Erstellt durch Projektteam	12
10	Finales Vier Gewinnt Spielanleitungenfenster	Erstellt durch Projektteam	13
11	Coverage Übersicht aus IntelliJ	Erstellt durch Projektteam	15
12	Coverage Übersicht aus IntelliJ	Erstellt durch Projektteam	15