fhwedel
UNIVERSITY OF APPLIED SCIENCES

PROGRAMMIERPRAKTIKUM

Flood Pipe

Name: Al Bittar
Vorname: Abdulrahman
Fachrichtung: Informatik
Matrikelnummer: 103869
Fachsemester: 9

fhwedel
UNIVERSITY OF APPLIED SCIENCES

1 Inhaltsverzeichnis:

2 Benutzerhandbuch

2.1 Ablaufbedingungen

Für den Betrieb der Software wird neben dem Programm selbst nur ein Java Runtime Environment (JRE) benötigt. Dieses sollte mindestens in Version 14 vorhanden sein.

2.2 Programminstallation/Programmstart

Das Verzeichnis “pp_FloodPipe_Al_Bittar.zip” ist in einen Ordner mit ausreichenden Lese- und Schreibrechten zu entpacken. Anschließend muss für den Spielstart die sich im Verzeichnis befindliche “.jar”-Datei ausgeführt werden (auf Windows-Rechnern einfach per Doppelklick).

2.3 Bedienungsanleitung:

2.3.1 Ziel des Spiels

Das Ziel des Spiels „Flood Pipe“ ist es, alle vorhandenen Rohrstücke auf dem Spielfeld so zu positionieren und zu verbinden, dass ein durchgängiger Flüssigkeitsfluss von der Quelle aus möglich ist, ohne dass es irgendwo zu offenen Enden kommt. Das Rohrstücke müssen so wenig wie möglich gedreht werden, um eine hohe Punktanzahl zu erreichen.

2.3.2 Hilfe im Spiel

Beim Hover auf eine Taste, wir eine Hilfsnachricht auftauchen, die dem Benutzer dabei hilft, was diese Taste genau macht.

Wenn der Benutzer eine Aktion macht, die zu Datenverlust führt, wird eine Bestätigung von ihm per Dialoge angefordert. Solche Aktionen können zum Beispiel aktuelles Feld löschen beim Editormodus.

Außerdem wird der Benutzer durch einen Dialog informiert, wenn irgendein Fehler auftritt. Dieser Dialog beinhaltet den Fehlercode.

Wenn der Benutzer etwas wissen muss, wird er auch entsprechend durch einen Dialog informiert.

2.3.3 Bedienung der Startseite

Nachdem die “.jar”-Datei ausgeführt wird, wird ein Fenster angezeigt. Ganz links auf dem Fenster stehen einige Einstellungen für den Benutzer.

Der Benutzer kann definieren, wie groß das Spielfeld sein sollte. Er kann die Anzahl der Reihen sowie der Spalten auswählen. Diese müssen zwischen 2 und 15 sein. Nicht-Quadratische Felder sind auch erlaubt.

Außerdem kann der Benutzer entscheiden, ob das Überlaufmodus aktiviert oder deaktiviert sein soll. Wenn der Überlaufmodus aktiviert ist, bedeutet das, dass das Spielfeld als zirkuläres Feld betrachtet wird. Das bedeutet, dass ein Rohrstück, das an einem Rand des Spielfelds liegt, sich mit dem Rohrstück, das am entgegengesetzten Rand liegt, verbinden, so dass ein Flüssigkeitsfluss durch den Rand hindurch weiter fließen kann, anstatt an einer Wand zu enden.

Das macht das Spiel interessanter und erhöht die Herausforderung, da der Spieler den Flüssigkeitsfluss in allen Richtungen berücksichtigen muss.

Das macht das Spiel interessanter und erhöht die Herausforderung, da der Spieler den Flüssigkeitsfluss in allen Richtungen berücksichtigen muss.

Das macht das Spiel interessanter und erhöht die Herausforderung, da der Spieler den Flüssigkeitsfluss in allen Richtungen berücksichtigen muss.Wenn der Benutzer die Taste „neues Spiel starten“ drückt, wird das Spielmodus aktiviert und ein ungelöstes Spielfeld angezeigt. Dieses Spielfeld sieht in jedem neuen Spiel anders aussehen. Die ausgewählten Einstellungen werden auch umgesetzt.

Wenn der Benutzer die Taste „altes Spiel laden“ drückt, taucht den Explorer auf. Dabei muss der Benutzer eine JSON-Datei auswählen, die schon aus vorherigem Spiel gespeichert wurde. Wenn die JSON-Datei gültig ist, wird das gespeicherte Spielfeld angezeigt und den Spielmodus aktiviert.

Wenn der Benutzer die Taste „Spiel beenden“ drückt, wird die Applikation geschlossen.

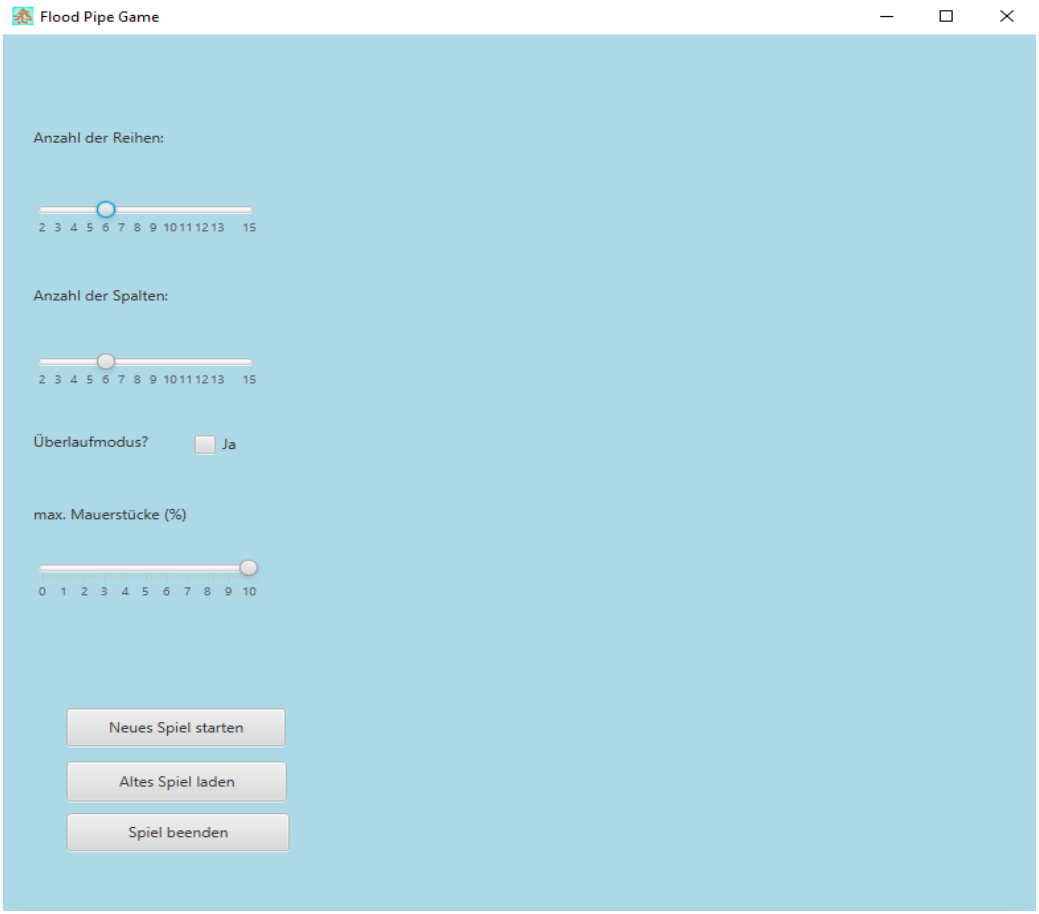

Abbildung 1: Startseite des Spiels

2.3.4 Bedienung des Spielfeldes

Nachdem ein Spiel gestartet oder geladen wurde, kann der Benutzer anfangen zu spielen. Das Feld hat mehrere Zellen. Jede Zelle kann entweder ein Rohr- oder ein Mauerstück sein. Der Benutzer kann mit einem linken bzw. rechten Mausklick die geklickte Zelle nach links bzw. nach rechts drehen, so dass am Ende das obenstehende Spielziel erfüllt wird. Ein Mauerstück kann zwar gedreht werden, wird aber dem Benutzer nicht sichtbar sein. Das heißt, ein Klick auf einem Mauerstück wird gezählt.

Als Rohrstücke existieren Geraden, Kurven, T-Verzweigungen und Endstücke (Sackgasse).

Jedes Rohrstück hat einige Öffnungsrichtungen (Übergänge), wie es in den Abbildungen zu sehen ist. Ein Mauerstück hat keine Öffnungsrichtungen. Diese Stücke werden dann in den sogenannten Editormodus verwendet, deswegen ist es gut zu wissen, was für Stücke sie sind.

Die Wasserquelle ist so animiert, dass dem Benutzer sofort einfallen kann, woher er anfangen soll. Die zufällig verbundenen Rohrstücke sind in blau, die noch nicht verbundenen hingegen in rot gefärbt.

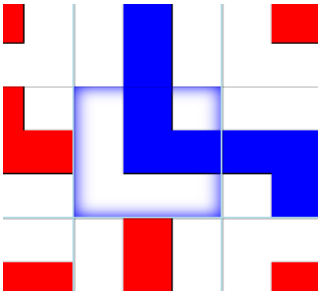

Abbildung 2: Wasserquelle

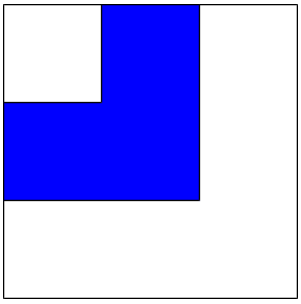

Abbildung 3: Kurve

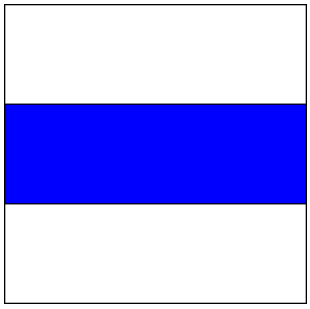

Abbildung 4: Gerade

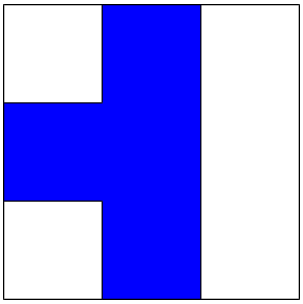

Abbildung 5 T-Verzweigung

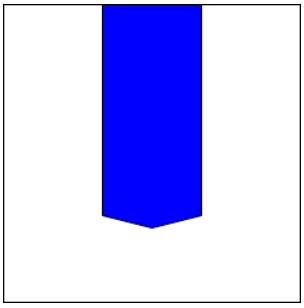

Abbildung 6 Sackgasse

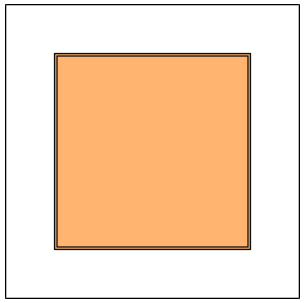

Abbildung 7 Mauerstück

Das Resultat steht auch unten mit einem Zähler für die bisherige Anzahl der Klicks, die der Benutzer benötigt hat, um zu aktuellem Zustand des Spielfeldes zu kommen.

Währen des Spiels hat der Benutzer die Möglichkeit, das aktuelle Spielfeld zu speichern. Das geht, wenn man auf der Navigationsmenü „Spiel“ drückt, und „Spiel speichern“ auswählt. Der Benutzer wird dann nach dem Pfad und Namen der Zieldatei durch den Explorer gefragt. Das soll eine JSON-Datei sein. Ist keine Datei

vorhanden, kann der Benutzer einfach einen Namen schreiben. Dadurch wird eine neue Datei automatisch angelegt und die Daten werden drin gespeichert. Wenn Fehler dabei auftreten, wird der Benutzer entsprechen informiert.

Außerdem kann der Benutzer im selben Navigationsmenü „Spiel“ ein neues Spiel anfordern oder das ganze Spiel beenden. Wenn dies der Fall ist, wird ein Hinweisdialog auftauchen, der besagt, dass ungesicherte Daten bei dieser Aktion verloren gehen. Der Benutzer kann entweder die Aktion bestätigen oder abbrechen.

Die Füllung der Rohrstücke wird im Laufe des Spiels animiert. Die Animationsgeschwindigkeit ist jederzeit änderbar. Um die Animationsgeschwindigkeit zu beschleunigen bzw. zu entschleunigen kann der Benutzer auf das Navigationsmenü „Animation“ drücken und die Animationsgeschwindigkeit ändern. Nachdem der Benutzer sie geändert hat, wird das ohne Bestätigung sofort umgesetzt.

Im dritten und letzten Navigationsmenü „Editormodus“ kann der Benutzer den Editormodus aktivieren. Das wird aber im nächsten Abschnitt erklärt und detaillierter besprochen.

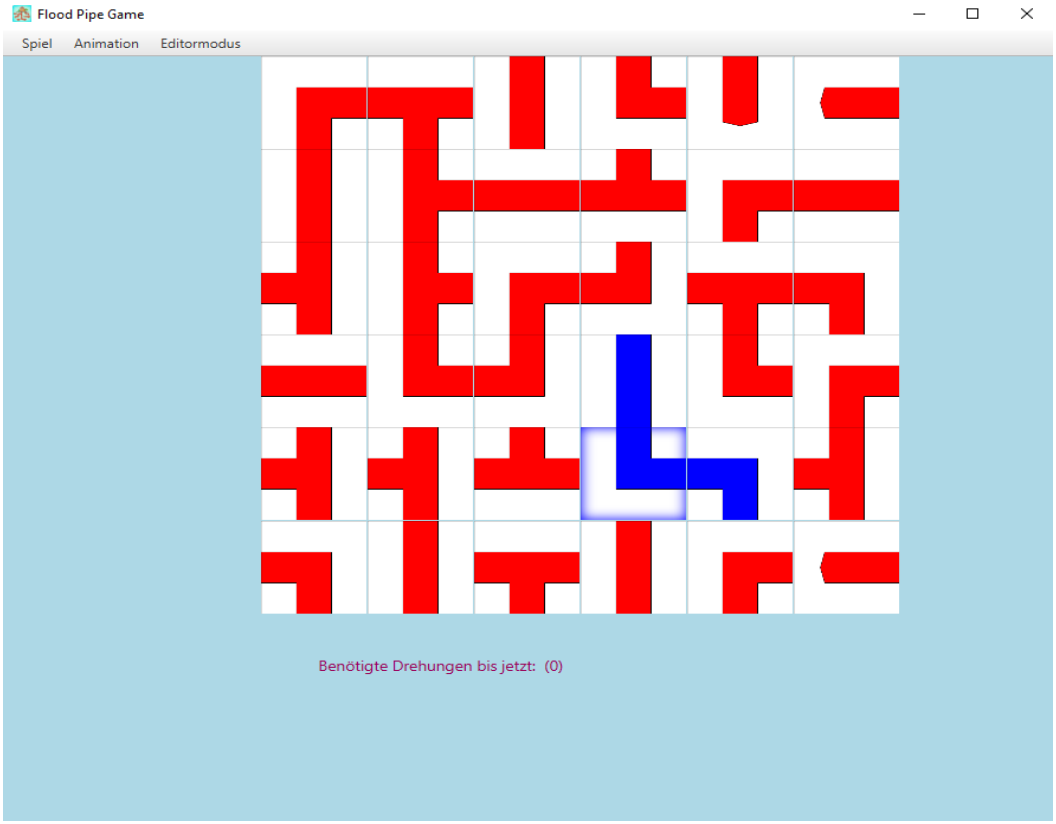

Abbildung 8: ungelöstes Feld am Anfang des Spiels

2.3.5 Bedienung des Editormodus

Im Editormodus kann der Benutzer selbst ein Spielfeld erstellen oder ein vorhandenes Spielfeld bearbeiten. Es stehen oben rechts alle Typen von Rohrstücken, ein Mauerstück und eine Wasserquelle zur Verfügung, die der Benutzer per Drag & Drop auf das Spielfeld ziehen können. Nachdem es in den Editormodus geschaltet wird, werden die aktuelle Belegung des Spielfeldes übernommen.

Der Benutzer kann jederzeit die Größe des Spielfeldes anpassen oder auch den Überlaufmodus de- bzw. aktivieren. Die Größe muss aber in derselben Schranke wie beim Start des Spieles bleiben, also zwischen 2 und 15 für jeweils für Reihen und Spalten. Die neuen Einstellungen werden erst umgesetzt, wenn der Benutzer die Taste „Einstellungen umsetzten“ drückt. Bei Verkürzung der Anzahl der Reihen oder Spalten, wird die aktuelle Belegung der verbleibenden Zellen bestehen bleiben. Im Gegenteil bei Vergrößerung der Anzahl der Reihen oder Spalten, werden neue Zellen mit Mauerstücken belegt.

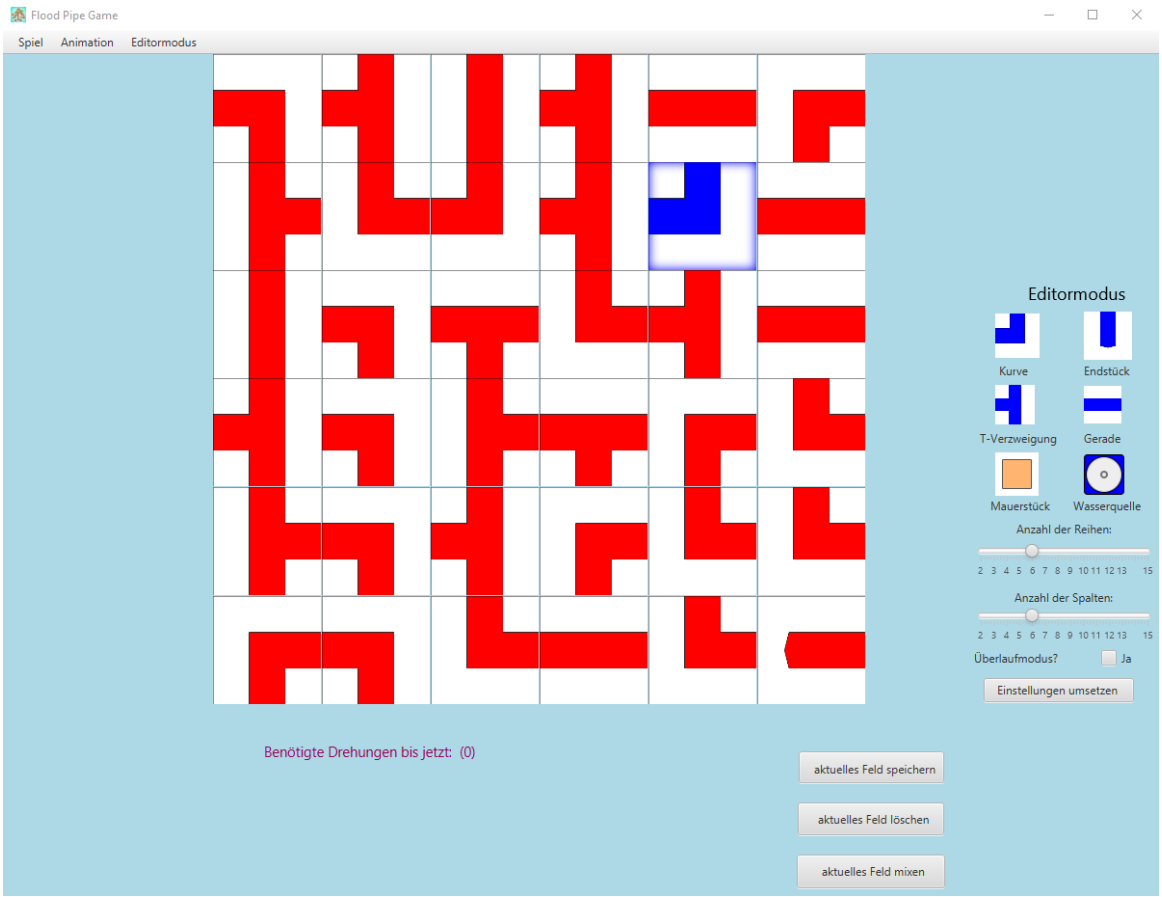

Abbildung 9: GUI bei Editormodus

Es gibt außerdem eine Taste „aktuelles Feld löschen“, falls der Benutzer ein neues leeres Spielfeld gestalten möchte. Die Taste löscht die aktuelle Belegung der Zellen und belegt alle Zellen mit Mauerstücken.

Der Benutzer kann auch die aktuellen Zellen des Spielfeldes mixen. Damit wird jede Zelle um eine zufällige Anzahl von Drehungen rotiert werden. Das passiert, wenn der Benutzer die Taste „aktuelles Feld mixen“ drückt.

Mit Drag&Drop der Benutzer Rohrstücke, Mauerstücke und eine Quelle auf das Spielfeld ziehen. Die Belegung der Zelle wird überschrieben. Mit einem Klick auf eine belegte Zelle wird das enthaltene Rohrstück gedreht. Eine Quelle kann nur auf ein durch ein Rohrstück belegtes Feld gesetzt werden. Ist schon eine Quelle vorhanden, wird diese gleichzeitig entfernt und die Quelle an der neuen Position gesetzt. Wenn ein Mauerstück auf die Quelle gezogen wird, wird die Quelle entfernt.

Sobald eine Quelle platziert ist, füllen sich die gefüllten Rohre automatisch. Ist das aktuelle Spielfeld gelöst, wird unten einen Hinweisdialog auftauchen und dem Benutzer darüber informieren, dass das aktuelle Spielfeld gelöst ist.

Das Spielfeld kann jederzeit gespeichert werden, auch wenn das Feld nicht gelöst ist. Die einzige Bedingung für das Speichern, dass es eine Quelle auf dem Spielfeld vorhanden ist. Das wird gemacht, wenn der Benutzer auf die Taste „aktuelles Feld speichern“ drückt. Wenn diese Taste gedrückt wird, wird es in den sogenannten Spielmodus zurückgeschaltet. Das heißt der Benutzer kann weiter versuchen, das Feld zu lösen.

2.3.6 Spielende

Wenn der Benutzer alle Rohrstücke so dreht, dass der obengenannten Ziel erfüllt ist, wird ein Glückwunsch-Dialog auftauchen. Der Dialog enthält die Anzahl der benötigten Klicks, die der Benutzer gebraucht hat, um das Spielfeld zu lösen.

Dabei kann er entweder ein neues Spiel starten oder das Spiel bzw. die Applikation beenden.

2.4 Fehlermeldungen

Bei bestimmten Vorgängen im Spiel können Fehlermeldungen erscheinen. Die häufigsten Gründe dafür sind die Manipulation gespeicherter Spielzustände durch eine unkundige Person. Aus diesen Gründen ist an dieser Stelle zu raten, die JSON-Dateien der Spielzustände überhaupt nicht zu manipulieren, damit die höchstmögliche Stabilität des Programms nicht beeinträchtigt wird.

Wenn irgendeinen Fehler während der Benutzung des Spieles auftritt, wird der Benutzer durch einen Dialog über den entsprechenden Fehler informiert. So ein Dialog sieht wie folgt aus.

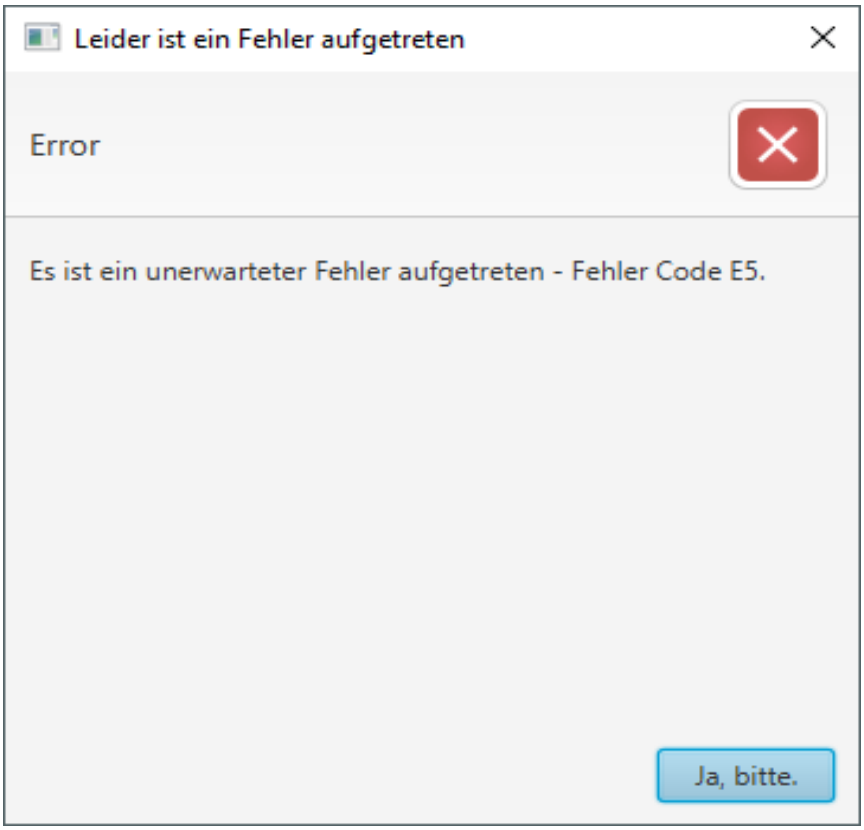

Abbildung 10 Fehlerdialog

In diesem Abschnitt werden alle Fehlercodes mit deren Ursache und Behebungsmaßnahme aufgelistet. Diese Behebungsmaßnahmen gelten als eine Lösung für den entsprechenden Fehler.

2.4.1 Beim Laden/Speichern

Fehlercode	Fehlermeldung	Ursache	Behebungsmaßnahme
E1	Die JSON-Datei ist nicht korrekt formatiert.	Die ausgewählte JSON-Datei beinhaltet eine Falsche Syntax.	Eine JSON-Datei mit korrekter Syntax auswählen
E2	Unzulässiger Zustandsfehler. Die Objekte der JSON-Datei enthalten unzulässige Zustände.	Die ausgewählte JSON-Datei beinhaltet unzulässige Zustände. Das kann z. B. ein Double-Wert statt einem Integer-Wert sein.	Eine JSON-Datei mit zulässigen Zuständen auswählen
E3	Fehler beim Lesen der Datei. Die Leseberichtigung bitte überprüfen.	Der Benutzer hat keine Berechtigung, auf die ausgewählte Datei zuzugreifen.	Die Zugriffberechtigung ändern oder eine andere Datei auswählen.
E4	Die Datei konnte unter dem angegebenen Pfad nicht gefunden werden.	Der angegeben Pfad ist falsch eingegeben.	Den Pfad der JSON-Datei korrekt angeben.
E5	Es ist ein unerwarteter Fehler aufgetreten.	Das passiert während des Ladens eines Spieles aus einem unbekannten Grund.	Unsere Help-Service anrufen oder daran eine E-Mail mit möglichst detaillierter Beschreibung der Situation, in der der Fehler aufgetreten ist, schicken. Das hilft uns dabei, unsere Software-Produkte auf einer hohen Qualität bleiben zu lassen.

Fehlercode	Fehlermeldung	Ursache	Behebungsmaßnahme
E9	Beim Schreiben in eine Datei ist eine nicht unterstützte Kodierungsfehler aufgetreten.	Die ausgewählte JSON-Datei benutzt ein nicht unterstütztes Kodierungsschema. Die Kodierung der JSON-Datei muss das UTF-8 unterstützen.	Das Kodierungsschema der ausgewählten JSON-Datei ändern oder eine andere JSON-Datei auswählen.
E10	Fehler beim Schreiben in die Datei. Die Schreibberichtigung bitte überprüfen.	Der Benutzer hat keine Berechtigung, auf die ausgewählte Datei zuzugreifen.	Die Zugriffberechtigung ändern oder eine andere Datei auswählen.
E11	Ein unerwarteter Fehler ist beim Speichern der Spielstände aufgetreten.	Das passiert während des Speicherns eines Spieles aus einem unbekannten Grund.	Unsere Help-Service anrufen oder daran eine E-Mail mit möglichst detaillierter Beschreibung der Situation, in der der Fehler aufgetreten ist, schicken. Das hilft uns dabei, unsere Software-Produkte auf einer hohen Qualität bleiben zu lassen.

2.4.2 Im Spielverlauf

Fehlercode	Fehlermeldung	Ursache	Behebungsmaßnahme
E6	Beim Starten eines neuen Spiels ist ein Fehler aufgetreten	Das Spiel kann aus technischen Gründen nicht gestartet werden.	Die Applikation schließen und wieder öffnen.
E7	Ein unerwarteter Button wurde angeklickt	Wenn ein Dialog auftaucht, wurde eine unerwartete Taste angeklickt.	Nur die Optionen anklicken, die innerhalb des Dialoges stehen. Beim Abbruch der gewünschten Aktion, die Aktion nochmal durchführen.
E12	Beim Neustart des Spiels wurde ein Input-Output-Fehler ausgelöst.	Das Spiel kann aus technischen Gründen nicht neugestartet werden.	Die Applikation schließen und wieder öffnen.

3 Programmiererhandbuch

Dieser Teil der Softwaredokumentation ist nur für Entwickler gedacht, deswegen wird an einigen Stellen ein Verständnis von höheren Programmierkonzepten sowie mathematischen Theorien vorausgesetzt.

Im Allgemeinen muss der Entwickler mit Java, JavaFX, OOP, Graphentheorie und Wahrscheinlichkeitstheorie vertraut sein.

3.1 Entwicklungskonfiguration

Komponente	Details
Prozessor	Intel Core i7-1165G7
Hauptspeicher	48,0 GB
Grafikprozessor	Intel(R) Iris(R) Xe Graphics NIVIDIA T500
Betriebssystem	Windows 10 Pro
Entwicklungsumgebung	IntelliJ IDEA Community Edition 2021.3.3
Java Development Kit	Version 17.0.2
JavaFX	Version 11.0.2
JUnit	Version 4.12
Gson	Version 2.8.8

Komponente	Details
Quellencodierung	UTF-8
JavaFX Scene Builder	Version 17.0.0
Draw.io Diagrams	diagrams.net 20.8.10

3.2 Problemanalyse und Realisation

3.2.1 Realisierung der Datenstruktur für Rohr- und Mauerstücke (Zellen)

Problemanalyse

Das Spielfeld ist eine zweidimensionale Collection. Diese Collection besteht aus Rohr- und Mauerstücken. Als Rohrstücke existieren Geraden, Kurven, T-Verzweigungen und Endstücke. Benötigt ist, eine Datenstruktur im Package „logic“, die diese Anforderungen erfüllt.

Realisationsanalyse

Erster Ansatz:

Alle Typen der Stücke in einer Klasse implementieren

Diese Klasse muss 2 primäre Klassenattribute haben. Ein enum-Attribut, das der Typ des Stücks ermittelt, und eine Collection, die die Öffnungsrichtungen enthält. Diese sind final, also können nur per Konstruktor zugewiesen werden. In dieser Klasse werden alle notwendige Methoden (z. B. Rotate()…) implementiert.

Eine andere Klasse (z. B. PlayingField genannt) muss ein 2D-Array vom Typ der ersten Klasse enthalten, wobei jeder Zelle diese Arrays ein Stück repräsentiert. Bei jeder Erzeugung eines dieser Stücke muss den Typ sowie ihre Öffnungsrichtungen eingegeben.

Pro- und Kontraargumente dieses Ansatzes

Proargumente sind

1- Die Datenstruktur ist einfach gehalten, denn sie besteht aus nur 2 Klassen.
2- Arbeiten mit „instancof“-Operator wird vermieden.

Kontraargumente sind

1- Die Skalierung dieses Ansatzes ist schwierig oder halt bei einigen Fällen gar nicht möglich. Wenn das Spiel weiterentwickelt wird, um zum Beispiel einen Abfluss zu realisieren, der alle an ihm verbundenen Rohre entleert. Wie wird so eine Anforderung mit dieser Datenstruktur implementiert? Wenn eine Lösung gefunden ist, ist die Datenstruktur nicht mehr objektorientiert.

2- Diese Datenstruktur ist rein theoretisch nicht objektorientiert, weil ein Mauerstück und ein Rohrstück an sich 2 unterschiedliche Objekte der Realität sind.

Zweiter Ansatz:

Jeder Typ der Stücke in einer eigenen Klasse implementieren

Dazu soll eine abstrakte Superklasse (*Cell* genannt) implementiert, die eine Zelle des Spielfeldes repräsentiert. Davon werden eine Klasse für Mauerstücke und eine für Rohrstücke erben.

Die gemeinsamen Methoden werden in der Superklasse implementiert. Dann werden die Subklassen die Superklasse mit eigenen Methoden und Attributen spezialisieren.

Eine andere Klasse (z. B. *PlayingField* genannt) muss ein 2D-Array vom Typ der Superklasse enthalten, wobei jeder Zelle diese Arrays ein Stück repräsentiert. Bei jeder Erzeugung eines dieser Stücke muss den Konstruktor der Subklassen aufgerufen.

Pro- und Kontraargumente dieses Ansatzes

Proargumente sind

1- Die Skalierung dieses Ansatzes ist einfacher.
2- Diese Datenstruktur ist objektorientiert, weil ein Mauerstück und ein Rohrstück an sich 2 unterschiedliche Objekte der Realität sind, und jedes dieser Objekte seine eigene Klasse hat.

Kontraargumente sind

1- Die Datenstruktur ist komplexer, denn sie besteht aus mehreren Klassen mit Vererbungen.
2- Arbeiten mit „instancof“-Operator kann nicht vermieden werden.

Realisationsbeschreibung

Jede Zelle des Spielfeldes wird durch eine abstrakte Superklasse repräsentiert. In jeder Zeile befindet sich je nach Algorithmus entweder ein Mauerstückt oder ein Rohr der Rohrstücke. Denn jeder Typ der Rohre hat seine eigene Klasse. Das wird dadurch realisiert, dass es eine abstrakte Klasse (*Pipe* genannt) implementiert wird, die die Superklasse aller Rohre repräsentiert. Damit wird auf die Aufzählung (enum-Attribut) der Rohrtypen verzichtet.

Für die Realisierung der Öffnungsrichtungen wird eine Aufzählung implementiert, die die Richtung einer Rohröffnung repräsentiert. Jede Klasse soll eine Collection haben, die seine Öffnungsrichtungen speichert. Eine perfekte Möglichkeit für das Speicher von Enum-Attributen ist die sogenannte Collection ***EnumSet***. Jetzt kann jedes Rohr in seinem Konstruktor überprüfen, ob die übergebenen Öffnungsrichtungen mit seinen möglichen Zuständen übereinstimmen. Was die Fehlerträchtigkeit von Erzeugung falscher Rohre verbietet und sofort erkennt.

Die Klassen können bestimmte Methoden implementieren. Manche davon können in den Elternklassen implementiert und manche erst in den Subklassen, je nachdem was für eine Operation soll die Methode implementieren. Beispielweise kann die Methode isOpenFromTop() in der abstrakte Superklasse (*Pipe*) implementiert und nicht mehr überschrieben werden, während die Methode rotate() muss erst in den jeweiligen Subklassen implementiert werden.

Implementiert wird folgende Klassen:

- ***Cell*** (abstrakt): Superklasse der Datenstruktur, repräsentiert eine Zelle im Spielfeld.
- ***Wall*** (konkret): Erbt von der Klasse *Cell*, repräsentiert die Mauerstücke im Spielfeld.
- ***Pipe*** (abstrakt): Erbt von der Klasse *Cell*, repräsentiert ein abstraktes Rohr im Spielfeld.
- ***CurvePipe*** (konkret): Erbt von der Klasse *Pipe*, repräsentiert eine Kurve im Spielfeld.
- ***StraightPipe*** (konkret): Erbt von der Klasse *Pipe*, repräsentiert eine Gerade im Spielfeld.

- ***T_Pipe*** (konkret): Erbt von der Klasse *Pipe*, repräsentiert eine T-Verzeigung im Spielfeld.
- ***EndPipe*** (konkret): Erbt von der Klasse *Pipe,* repräsentiert eine Sackgasse im Spielfeld.
- ***OpeningDirection*** (Aufzählungstyp): repräsentiert die Öffnungsrichtungen (Übergänge) der Zellen.

3.2.2 Rotation der Zellen in der Logik

Problemanalyse

Jede Zelle im Spielfeld muss nach rechts und links drehbar sein, um Verbindungen zwischen den Zellen herzustellen.

Realisationsanalyse

Da die Drehrichtungen fest vorgegeben sind, ist ein Aufzählungstyp dafür gut geeignet. Damit kann man statt 2 Methoden für die Drehung implementieren, nur eine Methode implementieren und die Richtung als Parameter eingeben.

Realisationsbeschreibung

Ein Aufzählungstyp ***Rotation*** für *LEFT* und *RIGHT* deklarieren und dann in der abstrakten Klasse *Pipe* eine abstrakte Methode *rotate(Rotation)* definieren. Diese Methode wird dann in jeder konkreten Rohrklasse implementiert.

Je nachdem, was der aktuelle Zustand des Rohres und in welche Richtung muss das Rohr gedreht, werden die Öffnungsrichtungen entsprechend angepasst.

3.2.3 Automatische Erzeugung eines gelösten Spielfeldes

Problemanalyse

Bei jedem Spielstart wird durch das Programm ein gelöstes Spielfeld erstellt (aber so noch nicht dargestellt). Dafür werden Geraden, Kurven und T-Verzweigungen zunächst mit gleicher Wahrscheinlichkeit eingesetzt. Das bedeutet nicht, dass im fertiggestellten Spielfeld die Rohrstücke in jeweils gleicher Anzahl vertreten sind. Endstücke werden nur verwendet, wenn an dieser Stelle kein anderes Element passt. Sind weniger Zellen als oder gleich viele Zellen wie Mauerstücke erlaubt sind frei geblieben, so können diese in den freien Zellen platziert werden.

Realisationsanalyse & Realisationsbeschreibung

Mehr detaillierte Informationen stehen in den Kapital (3.3.1).

3.2.4 Speichern der Spielzustände

Problemanalyse

Wie jedes mit Dateien arbeitende Programm muss dieses Programm die Möglichkeit anbieten, das aktuelle Spiel (Spielfeldgröße und -belegung) jederzeit in einer Datei zu speichern.

Die Datei enthält dafür im JSON-Format folgende Informationen:

- die Koordinaten der Quelle,
- eine boolesche Information über den Überlaufmodus,
- die Belegung des Spielfeldes.

Die Belegung des Spielfeldes muss je nach Öffnungsrichtungen des Rohrstücks in entsprechende int-Werte umgewandelt werden.

Die Zahlen für die Rohrstücke entstehen hierbei durch einfache Addition:

Öffnung nach	**Links**	**Unten**	**Rechts**	**Oben**
Bit	4	3	2	1
Dezimal	8	4	2	1

Bei einem Rohr mit Öffnungen nach oben, unten und links, ergibt sich die Zahl 1 + 4 + 8 = 13. Ein Mauerstück wird demzufolge mit der Zahl 0 angegeben, da keine Öffnungen bestehen.

Realisationsanalyse

Es muss eine Methode geben, die die Öffnungen der Zellen in einer binären Darstellung umwandelt. Die Position der Quelle könnte mithilfe von den Indizes der 2D-Array in der Klasse *PlayingField* besorgt. Der Überlaufmodus ist sowieso als Member-Attribut in der Klasse *PlayingField* gespeichert.

Um diese Daten in eine JSON-Datei zu exportieren, kann das von Google entwickelte Framework GSON verwendet.

Da die GSON-Methode *toJson*() ein Objekt übergeben erhält, sollen die Daten am Besten in ein Objekt einer Klasse ***GameData*** gespeichert und dann das Objekt der *toJson(GameData)* übergeben.

Ein typischer Fehler ist dabei, ein falscher Pfad für die Zieldatei zu übergeben. Um diesen Fehler zu vermeiden, wird die Klasse ***FileChooser*** von JavaFx benutzt.

Realisationsbeschreibung

Für die Umwandlung der Öffnungen wurde in jeder konkreten Klasse die Öffnungsrichtungen überprüft und den Wert hart kodiert zurückgegeben. Dieser Ansatz war zum Test entwickelt, wurde aber später nicht geändert.

Der richtige Ansatz soll mit shift-Operatoren entwickelt sein. Die Methode getOpeningBinaryValue() sollte in der Klasse *Pipe* wie folgt implementiert werden.

```
public String getOpeningBinaryValue() {
    int binarySum = 0;
    for (OpeningDirection direction : this.openingDirections) {
        binarySum |= 1 << direction.getDirectionID();
    }
    return Integer.toBinaryString(binarySum);
}
```

Abbildung 11: Die bessere Implementierung von getOpeningBinaryValue()

In der *Wall* Klasse sollte hart kodiert bleiben, da sie keine Collection ***EnumSet<OpeningDirection> openingDirections*** hat.

Alle Implementierungen von getOpeningBinaryValue() in den konkreten Klassen sollten gelöscht werden.

Um die Daten in einem Objekt zu speichern, wurde die benötigten Attribute als Member-Attribute in der Klasse ***GameData*** deklariert, und nur per Konstruktor zuweisbar gemacht.

In der Klasse ***GameData*** wurde eine statische Methode *toJSON(path:String, gameData: GameData)* implementiert, die das GSON Framework benutzt um Daten zu einem JSON-Objekt umzuwandeln.

In der FXML-Datei wird ein Button für das Speichern der Spielzustände gestellt. Das geht ganz einfach mithilfe von dem Scene Builder. Dazu muss im Abschnitt Code eine Id angegeben und unter „On Action“ einen Namen für eine Handler-Methode eingegeben.

Eine Handler-Methode ist eine Methode in JavaFX, die ein Ereignis behandelt, das durch ein bestimmtes Steuerelement oder eine Aktion ausgelöst wurde. Diese Methode definiert die Aktionen, die ausgeführt werden sollen, wenn ein bestimmtes Ereignis eintritt. Ein Handler wird oft mit einer Steuerelementaktion wie einem Mausklick oder einem Tastendruck verknüpft, um eine Reaktion auf das Ereignis auszulösen. Hier ist die Aktion das Button anklicken.

Dann im Controller in der Handler-Methode wird eine Instanz von ***FileChooser*** deklariert, um einen richtigen Pfad zur JSON-Datei zu sichern. Außerdem wird ein Objekt von *GameData* deklariert und zugewiesen. Dann wird die statische Methode *toJSON(path:String, gameData: GameData)* von der Klasse ***GameData*** aufgerufen und Fehler sorgfältig behandelt. Dann ist der Prozess fertig.

3.2.5 Laden der Spielzustände

Problemanalyse

Die gespeicherten Daten können zu einem späteren Zeitpunkt wieder geladen werden, so dass das Spiel weitergespielt werden kann.

Realisationsanalyse

Beim Lesen der Daten muss einen Parser geben, der die Objekte in der JSON-Datei iteriert und zu entsprechenden Objekten umwandelt.

Es muss eine Methode geben, die die int-Werte des gespeicherten Spielfeldes zu binärer String-Darstellung umwandelt. Diese binäre String-Darstellung sollen zu einem Konstruktor in der Klasse ***PlayingField*** übergeben, so dass der Konstruktor ein vernünftiges Spielfeld erzeugt.

Diese Daten werden wiederrum in einem Objekt gespeichert, um ein neues Spiel zu erzeugen.

Ein typischer Fehler ist hier auch, ein falscher Pfad für die Zieldatei zu übergeben. Um diesen Fehler zu vermeiden, wird die Klasse ***FileChooser*** von *JavaFx* benutzt.

Realisationsbeschreibung

Da die benötigten Attribute als Member-Attribute in der Klasse ***GameData*** deklariert sind, können sie als Platzhalter für die eingelesenen Daten sein.

In der Klasse ***GameData*** wurde eine statische Methode *FromJSON(path:String)* implementiert, die das GSON Framework benutzt um Daten zu von einer JSON-Datei

tu einem Java-Objekt umzuwandeln. Dabei funktioniert die Klasse ***JsonObject*** von *Gson* sehr gut als JSON-Parser. Dieser Parser parst die Objekte in der JSON-Datei, die dann zu einem Objekt vom Typ ***GameData*** Objekt umgewandelt werden.

In der FXML-Datei wird ein Button für das Laden eines neuen Spieles gestellt und eine Handler-Methode deklariert. Das geht wie beim Button vom Speichern der Spielzustände im vorherigen Abschnitt.

In der Handler-Methode in der Controller-Klasse wird dann **FileChooser** benutzt, um einen korrekten Pfad zu sichern. Dann wird die Methode ***fromJson*** von ***GameData*** aufgerufen und den Pfad übergeben. Damit ist ein Objekt von ***GameData*** da, das alle benötigte daten enthält.

In der Klasse ***Game*** soll ein Konstruktor geben, der dieses Objekt übergeben bekommt, der in seiner Rolle die Daten zur Klasse ***PlyingField*** weiterleitet, um ein Spielfeld zu erzeugen.

Die Klasse ***PlyingField*** hat einen Konstruktor, der die Daten der JSON-Datei bekommt und sie entsprechend zu einem Spielfeld umwandelt. In diesem Konstruktor müssen die int-Werte zu binärer String-Darstellung umgewandelt werdet. Das macht er mit der Hilfsmethode *convertIntegerCellsTOBinary*(), die das übergebene 2D-Array traversiert und mithilfe von Java-methode *Integer*.*toBinaryString*(int) den übergebenen int-Wert zu binärer String-Darstellung umwandelt.

Jetzt sind alle benötigte Objekte zu dem Erzeugen eines Spiels vorhanden und können sie aufgerufen werden, um ein neues Spiel zu erzeugen. Dazu mehr in der in dem Abschnitt (3.2.7).

3.2.6 GUI – Malen von kreativen Bildern für die einzelnen Zellen

Problemanalyse

Die Rohre müssen als solche erkennbar sein und die Übergänge zueinander passen. Mauerstücke sollen auch dargestellt werden. Eine ansprechende Darstellung der Stücke ist angefordert.

Realisationsanalyse

Die Bilder sollen entweder vom Internet besorgt oder selbst gemalt werden.

Realisationsbeschreibung

Im Internet gab es keine ansprechenden Darstellungen, die der Anforderungen erfüllen, daher wurden die Bilder mit Hilfe von Draw.io selbst gemalt.

Alle benutzten Symbole waren glücklicherweise in den Symbolliste von Draw.io schon vorhanden.

Damit die Übergänge zueinander passen können, wurde ein weißes Viereck in der Mitte gezogen, seine Größe angepasst und über ihm ein Symbol gezogen und losgelassen. Die Übergänge bzw. die Größe des Symbols wurden angepasst, das Symbol gefärbt und das Bild als .png-Datei exportiert.

Das wurde für alle notwendigen Bilder wiederholt, bis alle Bilder gemalt sind.

3.2.7 GUI – Konstruktion der Game-Instanz

Problemanalyse

Nach dem Start der Applikation hat der Benutzer mehrere Möglichkeiten, um den Spielablauf an seine Bedürfnisse anzupassen. Zu den Optionen gehören z.B. die Erzeugung eines Spielfeldes im Editormodus, neues Spiel starten, während im Moment ein aktuelles Spiel abläuft und Anpassung der Zeilen- und Spaltenanzahl des Spielfeldes auf eigene Wünsche. Diese Werte müssen der Spiellogik beim Start des Spiels übergeben werden, damit der Zustandsverlust von Objekten in den Klassen ***JavaFXGUI***, ***UserInterfaceController*** und ***Game*** vermieden wird.

Realisationsanalyse

Ein möglicher Ansatz wäre das Deklarieren von statischen Game-Instanzen in den relevanten Klassen und sie immer bei jeder Änderung bündig durch Konstruktionen und Getter machen.

Vorteile dieses Ansatzes sind, dass kein Zustandsverlust von Objekten durch null-Werte entstanden wird, Denn statt einer neuen Instanz von z. B. dem Controller zu erstellen, um die Game-Instanz zu aktualisieren, was Zustandsverlust von Objekten verursacht, einfach per Klassenname die Game-Instanzen aufgerufen und aktualisiert werden.

Realisationsbeschreibung

In den relevanten Klassen ***JavaFXGUI*** und ***UserInterfaceController*** eine Instanz von Game deklarieren und sie immer bei jeder Änderung bündig durch Konstruktionen und Getter machen. In diesem Ansatz werden die zu ändernde Objekte aktualisiert, ohne Zustandsverlust von anderen Objekten zu verursachen.

3.2.8 GUI – Darstellung der Wasserquelle

Problemanalyse

Die Quelle liegt an einer zufälligen Position auf dem Spielfeld. Sie kann auf jedem der Rohrstücke platziert sein. Die Quelle darf beliebig symbolisiert werden, muss aber eindeutig identifizierbar sein und das unterliegende Rohrstück erkennbar lassen.

Realisationsanalyse

Ein möglicher Ansatz wäre, das Bild der Quelle und des Rohres durch Code zusammenzufügen. Das Problem dabei ist, dass das nicht Standard in Java ist, was zu einem nicht schönen Bild führt.

Der weitere Ansatz wäre, für jedes mögliche Bild ein separates Bild mit Draw.io zu malen und im IMAGE-Array zu speichern. Das ist eigentlich eine gute Lösung, macht der Code aber komplexer, weil bei jeder Zuweisung einer Zelle des Spielfeldes eine Überprüfung durchgeführt wird, ob Zelle eine Quelle ist, und wenn Ja was für eine Rohrstück da ist, um ein entsprechendes Bild anzuzeigen. Außerdem ist das Löschen einer Quelle im Editormodus noch komplexer.

Der letzte Ansatz ist, statt ein Bild zuzuweisen, einen Effekt über das *ImageView* der Quelle hinzufügen, und bei jeder Änderung den Effekt einfach löschen. Dieser Ansatz ist am einfachsten, weil das *JavaFx* Standard ist.

Realisationsbeschreibung

In der Klasse ***JavaFXGUI*** wurde eine Methode *setEffectToWaterSource*() implementiert, die einfach dem *ImageView* der Wasserquelle einen schönen Effekt hinzufügt. Der Effekt setzt einen blauen Rahmen durch das *ImageView*.

Zum Löschen des Effektes wird die Anweisung *setEffect*(null) von dem *ImageView* der Wasserquelle aufgerufen.

3.2.9 GUI - Rotation der Zellen in der View

Problemanalyse

Jede Zelle im Spielfeld muss nach rechts und links drehbar sein, um Verbindungen zwischen den Zellen herzustellen, das muss dem Spieler sichtbar sein.

Realisationsanalyse

Es muss einen Event-Handler in jeder Zelle des Spielfeldes geben, die dieses Ereignis behandelt. Wobei ein Event Handler eine Methode in JavaFX ist, die ein Ereignis behandelt, das durch ein bestimmtes Steuerelement oder eine Aktion ausgelöst wurde. Ein Event Handler registriert einen Hörer (Listener) für ein bestimmtes Ereignis und definiert die Aktionen, die ausgeführt werden sollen, wenn das Ereignis eintritt. Beispiele für Ereignisse in JavaFX können sein, wenn ein Button geklickt wird, eine TextBox den Fokus verliert oder wenn die Größe eines Fensters geändert wird.

Realisationsbeschreibung

Das Hinzufügen des Event-Handlers erfolgt bei der initialen Erzeugung der ImageViews. Das wird von JavaFx mit der Methode *addEventhandler*() unterstützt.

In der methode selbst wird geprüft, ob das das ImageView mit einer linken bzw. rechtem Mausklick angeklickt ist und wird entsprechend mit der Methode setRotate() gedreht.

3.2.10 Eitormodus – Mechanismus von Drag&Drop

Problemanalyse

Im Editormodus werden neben dem Spielfeld die einsetzbaren Rohrstücke, ein Mauerstück und eine Quelle angezeigt.

Per Drag&Drop können diese Elemente auf das Feld gezogen werden. Die Belegung der Zelle wird durch den Drop überschrieben.

Realisationsanalyse

In JavaFx erfolgt das Drog&Drop per Litner-Methoden. Das ist Standard unterstützt, muss nur angewendet werden.

Ein Listener in JavaFX ist eine Art von Event-Handler, der auf Ereignisse reagieren kann, die auf einer Benutzeroberfläche oder einem JavaFX-Element auftreten. Ein Listener kann eingerichtet werden, um auf bestimmte Benutzeraktionen wie Mausklicks oder Tasteneingaben zu reagieren und entsprechende Aktionen auszuführen. Es handelt sich hierbei um ein wichtiges Konzept bei der Erstellung von interaktiven JavaFX-Anwendungen.

Realisationsbeschreibung

In der FXML-Datei muss für jedes Stück eine eigene Listener-Methode vom Typ setOnDragDetected() hinzugefügt weden.

Dieser JavaFX-Code startet einen Drag-and-Drop-Vorgang. Hierbei wird ein Dragboard-Objekt erzeugt, das mithilfe der Methode "startDragAndDrop" aktiviert wird. Hierbei wird ein TransferMode-Parameter angegeben, welcher angibt, welche Art von Übertragung (Copy, Move, Link, etc.) zulässig ist. Anschließend wird ein ClipboardContent-Objekt erstellt, welches die Daten für die Übertragung enthält. Hierbei werden mit den Methoden "putImage" und "putString" ein Bild und eine Zeichenkette hinzugefügt, um im Ziel-ImageView erkannt wird, was für ein Image das ist. Schließlich werden die Inhalte des ClipboardContent-Objekts dem Dragboard-Objekt hinzugefügt. Zuletzt wird das MouseEvent-Objekt mit der Methode "consume" konsumiert, um anzuzeigen, dass das Event verarbeitet wurde und nicht weiterverarbeitet werden soll.

In dieser Art und Weise ist der Source-ImageView eingestellt und muss das „Loslassen“ im Ziel-ImageView eingestellt.

Bei der Initialisierung des 2D-Array des Spielfeldes von ImageView muss 2 Event-Handler für jedes ImageView hinzugefügt.

1- Event-Handler vom Typ onDragOver. Diese Methode zwei Argumente entgegen: aktuelle ImageView und seine Position.
Die Methode fügt einen Event-Handler zu dem übergebenen ImageView hinzu, um das Drag-and-Drop-Verhalten beim Überschreiben zu behandeln. Innerhalb des Event-Handlers wird überprüft, ob die Quelle des Drag-and-Drop-Vorgangs nicht die gleiche ist wie das Ziel (iv) und ob das Dragboard ein Bild enthält. Wenn diese Bedingungen erfüllt sind, wird überprüft, ob es sich bei dem String im Dragboard um den String

Constants.WATER_SOURCE_STR handelt. In diesem Fall muss überprüft werden, ob die aktuelle Zelle auf dem Spielfeld eine Wand ist. Wenn dies der Fall ist, werden keine Übertragungsmodi akzeptiert. Wenn nicht, werden alle Übertragungsmodi akzeptiert. Am Ende wird das Ereignis mit event.consume() verarbeitet, um sicherzustellen, dass es nicht weiterverarbeitet wird.

2- Event-Handler vom Typ OnDragDroppt. Diese Methode dient dazu, ein Ereignis (Drag and Drop) auf einem ImageView-Element in JavaFX zu verarbeiten. Wenn ein Drag-Drop-Ereignis auf dem ImageView-Element stattfindet, wird es von der Methode handleOnDragDropped behandelt. Zunächst wird das Dragboard-Objekt aus dem DragEvent abgerufen. Dann wird überprüft, ob das Dragboard ein Bild enthält. Wenn ja, wird ausgegeben, dass das Bild erfolgreich gedroppt wurde. Anschließend wird der Typ des gedroppten Elements mithilfe des Strings aus dem Dragboard bestimmt und je nach Typ eine entsprechende Aktion durchgeführt. Dies kann beispielsweise das Setzen eines neuen Elements auf dem Spielfeld oder das Ändern des Wasser-Quell-Elements sein. Schließlich wird das Ergebnis des Drop-Ereignisses auf success gesetzt und das Ereignis wird konsumiert.

3.2.11 Animation von der Füllung der Rohre

Problemanalyse

Die Füllung der Rohre soll animiert erfolgen. Dafür soll von der Quelle ausgehend je ein füllbares Rohrstück die Farbe ändern, bis keine weiteren Rohrstücke gefüllt werden können. Bereits im vorigen Schritt gefüllte Rohrstücke sollen allerdings nicht zur Verzögerung der Anzeige führen.

Realisationsanalyse

Man muss die neu verbundenen Rohre erkennen und ihre Farbe schrittweise mit Zeitabstand ändern.

Für die Erkennung aller verbundenen Rohre wird der Tiefensuche-Algorithmus angewandt. Genauer erklärt ist dieser Algorithmus im Abschnitt (3.3.2), seine Vor- und Nachteile werden im selben Abschnitt aufgeführt.

Damit die schon animierten Rohre vor der Drehung nicht nochmal animiert werden, werden ihre Positionen in einer Liste in der Klasse Game gespeichert. Die neu verbundenen Rohre findet man dadurch heraus, dass man die aktuell nach der

Drehung alle verbundenen Rohre mit der Liste von schon vor der Drehung verbundenen Rohre vergleicht. Die in beiden Listen stehenden Rohre, werden gefiltert.

Vorteile dieses Ansatzes:

1- Die Operationen, die durchgeführt werden, sind von java collections unterstützt.
2- Leicht zu verstehen.

Nachteile dieses Ansatzes:

1- Vielleicht ist das nicht der bestmögliche Ansatz.

Diese gefilterte Liste, die Rohre zur Animation hat, muss an eine Methode eingegeben, die die Animation durchführt.

Man kann diese Methode mithilfe der Klasse ***Timer*** oder ***TimeLine*** implementieren. Es hängt davon ab, welche Art von Animation Sie benötigen.

Was ist besser ***Timer*** oder ***TimeLine***?

Ein ***Timer*** eignet sich besser für regelmäßige Wiederholungen von Aktionen, z. B. das ständige Aktualisieren eines Fortschrittsbalkens. Timer sind einfach zu verwenden und erfordern wenig Code.

Eine ***TimeLine*** ist hingegen besser für eine komplexere Animation mit mehreren Schritten, unterschiedlichen Geschwindigkeiten und einer definierten Länge. Mit TimeLine kann man eine Zeitachse erstellen, die bestimmte Änderungen an einer bestimmten Zeit vornimmt, und man hat mehr Kontrolle über den Ablauf und die Eigenschaften der Animation.

Daher sollte man abhängig von den Anforderungen an die Animation die passende Methode wählen.

Deswegen wird diese Animation in dieser Software mit ***TimeLine*** implementiert.

Realisationsbeschreibung

In der Klasse ***JavaFxGUI*** wird die Methode ***animatePipeFilling()*** implementiert. Die Methode verwendet die JavaFX Animation API, um eine Animation der gefüllten Rohre in einem Spiel darzustellen.

Zunächst wird eine Liste der Rohre abgerufen, die gefüllt werden müssen, indem die Methode ***game.getSortedConnectedPipes()*** verwendet wird.

Wenn es keine Rohre zu füllen gibt, wird sofort das Spielfeld auf der GUI aktualisiert. Andernfalls wird eine Timeline für die Animation erstellt.

Ein ***AtomicInteger*** wird verwendet, um die Anzahl der verbleibenden Rohre zu verfolgen, die gefüllt werden müssen. Dies ist wichtig, da es in einer multithreaded Umgebung möglich ist, dass mehrere Threads gleichzeitig auf die Anzahl zugreifen und aktualisieren.

Ein ***KeyFrame*** wird der Timeline hinzugefügt, der die GUI aktualisiert. In jedem Frame wird das nächste Rohr aus der Liste der zu füllenden Rohre entfernt und sein Bild aktualisiert, um es als gefüllt anzuzeigen.

Ein ***KeyFrame*** ist ein Zeitpunkt in einer JavaFX Timeline, an dem eine Aktion ausgeführt wird. Ein ***KeyFrame*** definiert den Zeitpunkt, zu dem eine bestimmte Aktion ausgeführt werden soll, sowie die Aktion selbst. In JavaFX wird die Dauer einer Timeline in Zeitintervallen gemessen, und jeder ***KeyFrame*** wird an einem bestimmten Zeitpunkt innerhalb dieser Intervalle ausgeführt. Die Klasse KeyFrame repräsentiert einen ***KeyFrame*** in JavaFX.

Wenn alle Rohre gefüllt wurden, wird die Animation gestoppt und das Spielbrett auf der GUI aktualisiert.

Die CycleCount der Timeline wird auf die Anzahl der Rohre gesetzt, die gefüllt werden müssen. Zum Schluss wird die Timeline gestartet, um die Animation zu starten.

3.2.12 Echtzeitumsetzung der eingestellten Animationsgeschwindigkeit

Problemanalyse

Die Geschwindigkeit der Animation soll für den Benutzer in einem sinnvollen Rahmen über das Optionsmenü änderbar sein.

Realisationsanalyse

Eine Möglichkeit, um die Geschwindigkeit der Animation zu ändern, ist das Konfigurieren einer anpassbaren Zeitdauer für den ***KeyFrame***. Diese Zeitdauer kann als eine Option im Optionsmenü bereitgestellt werden.

Um dies zu tun, kann man eine globale Variable (z.B. ***animationDuration***) für die Dauer der Animation in Millisekunden definieren. Diese Variable kann über das Optionsmenü aktualisiert werden, wenn der Benutzer die Geschwindigkeit der Animation ändern möchte. Dann, anstelle einer festen Zeitdauer für den ***KeyFrame*** zu verwenden, kann man ***animationDuration*** als Argument für den Konstruktor des ***KeyFrames*** übergeben:

```
fillTimeline.getKeyFrames().add(
        new KeyFrame(animationDuration, event -> { ... }));
```

Auf diese Weise wird die Geschwindigkeit der Animation durch die Änderung der Wert der animationDuration-Variable geändert, die von dem Benutzer über das Optionsmenü eingestellt werden kann.

Realisationsbeschreibung

Eine Listner-Method wurde zum Slider im Menü Animation hinzugefügt, die die statische Variable ***animationDuration*** sofort umsetzt, falls der Benutzer das ändert.

3.3 Algorithmen

3.3.1 Solve-Algorithmus vom Spielfeld

Der Algorithmus traversiert das 2D-Array einfach von links nach rechts und von oben nach unten durch 2 einfache for-Schleifen, und überprüft durch die Indizes je nach Zellenposition alle möglichen Fälle und Szenarien, die an dieser Position kommen können. Weil der Algorithmus schon weiß, wo er ist und was für Stücke seine Nachbarn haben, sind die Fällen an jeder Zelle reduziert. Am Anfang sind alle Zellen des Arrays mit null zugewiesen. Das Ganze wird an einem Beispiel deutlicher.

Stellen wir mal und vor, wir haben ein **4*4 Spielfeld**. Der **Überlaufmodus** ist **nicht aktiviert**.

- In der ersten Zelle [0][0] kann nur der *CurvePipe(Right, Bottom)* passen.
- An den restlichen Zellen in der ersten Reihe von [0][1] bis [0][2] sind nur zu überprüfen, ob die vorletzte Zelle von rechts geöffnet ist. Wenn das der Fall ist, dann sollte ein zufälliges Rohr mit mindestes einer Öffnung nach links und keine Öffnung nach oben (weil der Überlaufmodus deaktiviert ist) eingesetzt.
- An der Zelle [0][3] wird überprüft, ob die vorletzte Zelle von rechts geöffnet ist. Wenn das der Fall ist, dann sollte ein zufälliges Rohr mit mindestes einer Öffnung nach links und keine Öffnung nach oben (weil der Überlaufmodus deaktiviert ist) oder rechts (weil das die letzte Zelle in dieser Reihe ist und der Überlaufmodus deaktiviert) eingesetzt.
- An den Zellen der ersten Spalte [*][0] ist nur obige Zelle relevant. Wenn sie nach unten geöffnet ist, dann wird eine Zelle mit Öffnung nach oben und keine Öffnung nach links eingesetzt. Das Gleiche aber umgekehrt gilt für die Zellen der letzten Spalte [0][3].
- Für die Zelle an der Position [3][0] (letzte Zeile, erste Spalte) ist wie bei Zellen in [*][0] aber es muss das zufällige Rohr keine Öffnung nach unten haben.
- An den restlichen Zellen in der letzten Reihe, also von [3][1] bis [3][2], sind nur zu überprüfen, ob die vorletzte Zelle von rechts geöffnet ist, und ob die obige Zelle von unten geöffnet ist. Wenn das der Fall ist, dann sollte ein zufälliges Rohr mit mindestes Öffnungen nach links und oben und keine Öffnung nach unten (weil der Überlaufmodus deaktiviert ist) eingesetzt.
- An der Zelle [3][3] wird überprüft, ob die vorletzte Zelle von rechts geöffnet ist und ob die obige Zelle von unten geöffnet ist. Wenn das der Fall ist, dann sollte ein zufälliges Rohr mit Öffnungen nach links und oben und keine Öffnung nach unten oder rechts (weil das die letzte Zelle in dieser Reihe ist und der Überlaufmodus deaktiviert) eingesetzt. In solchem Fall Passt nur ein *CurvePipe(Left,TOP).*
- An den restlichen Zellen (die nicht am Rande sind) ist dann das vorletzte und obige Rohr zu überprüfen und entsprechend dem Fall ein zufälliges Stück erzeugen.

Anmerkung: Um ein zufälliges Rohr zu bekommen, wurden einige Hilfsmethoden implementiert, die je nach Fall und Position der Zelle die möglichen Rohre überprüfen. Wenn an einer Stelle eine Öffnung gewünscht bzw. nicht gewünscht ist, spricht der Algorithmus von Wanted bzw. Unwanted. Diese beiden Collections werden überprüft und entsprechend dessen ein Rohr falls möglich zurückgeben, ansonsten Null, der später zu einem Mauerstück umgewandelt ist, falls keine Sackgasse an der Position passt.

Um ein Zufälliges Rohr zu generieren, wird die vordefinierte Java-Klasse ***Random*** benutzt, um Zahlen zwischen bestimmter Schranke zu generieren und je nach dieser Zahl, wird das mögliche Rohr ausgewählt.

Dasselbe gilt für den Fall, dass der Überlaufmodus aktiviert ist. Ein Überlaufmodus in einem 2D-Array bezieht sich auf eine Technik, bei der Werte, die über die Grenzen (Ränder) des Arrays hinausgehen, wieder zum Anfang des Arrays zurückgeführt werden. Dies bedeutet, dass sie über den Rand des Arrays "überlaufen" und an der anderen Seite wieder auftauchen. Das macht das 2D-Array eine „Zirkuläres 2D-Array“.

Das bedeutet die Zellen an den Rändern müssen zusätzliche Überprüfungen durchführen, damit Zellen am an der anderen Seite des jeweiligen Randes betrachtet werden.

Ein weiterer Fall muss berücksichtigt werden, wenn der Überlaufmodus aktiviert ist. Ist es vermieden werden muss 4 Öffnungen an der letzten Zelle in der letzten Zeile und in der letzten Spalte (An [3][3]) vom Beispiel) stehende Rohr zu verbinden. Das kann überprüft werden, wenn die Schleifen an der Position [0][3] ist. Dort muss überprüft werden, ob es schon 3 Öffnungen von den restlichen Nachbarzellen Zellen an der Zelle [3][3] verbunden sind. In diesem Fall muss das Rohr an der Zell [3][0] keine Öffnung nach rechts haben, damit keine 4 Öffnungen an der letzten Zelle verbinden werden. Dieser Fall wurden in einer Hilfsmethode ***avoidBlocker()*** überprüft.

Zusammenfassung des Algorithmus

Der Algorithmus verwendet zwei for-Schleifen, um durch ein 2D-Array von links nach rechts und von oben nach unten zu traversieren. An jeder Zelle werden die Nachbarzellen überprüft und basierend auf ihren Zuständen (Öffnungen) und der Position der Zelle werden mögliche Szenarien bestimmt. Alle Zellen werden zu Beginn auf null initialisiert. Die möglichen Fälle sind abhängig von der Position der Zelle und dem Überlaufmodus (deaktiviert). Zufällige Rohre können mit

Hilfsmethoden generiert werden, die mögliche Optionen abhängig vom Fall und der Position überprüfen.

Anmerkung: Das ist ein intelligenter Algorithmus und wird in der Literatur als RbR (steht für rule based reasoning) klassifiziert.

Vorteile von dem Algorithmus:

1- **Einfachheit (nicht zeitaufwendig)**: Dieser Algorithmus ist einfach zu verstehen und zu implementieren, da er auf zwei einfachen for-Schleifen und Überprüfungen der Indizes basiert.
2- **Flexibilität**: Dieser Algorithmus ist flexibel und kann leicht angepasst werden, um unterschiedliche Spielfelder und Überlaufmodi zu generieren.
3- **Leistung**: Dieser Algorithmus ist schnell und effizient, da er einfach durch das Spielfeld geht und für jede Zelle nur die relevanten Nachbarn überprüft.

Nachteile von dem Algorithmus:

1- **Fixe Regeln**: Dieser Algorithmus kann nur eine begrenzte Anzahl von Regeln und Fällen handhaben, die bereits im Voraus festgelegt sind.
2- **Zufälligkeit**: Dieser Algorithmus basiert auf zufälligen Entscheidungen, die manchmal zu unerwarteten oder ungünstigen Resultaten führen können.
3- **Keine Lernfähigkeit**: Dieser Algorithmus hat keine Möglichkeit, aus seinen Fehlern zu lernen oder seine Leistung zu verbessern.

Gibt es ein besserer Algorithmus für das gleiche Problem?

Es wird viel Recherche gemacht, um einen besseren Algorithmus zu finden, gab es aber aus Sicht des Entwicklers keine. Ein Algorithmus mithilfe von Backtracking wäre eine mögliche Alternative, das löst das Nachteile des angewandten Algorithmus nicht, weil sowieso werden abhängig von der aktuellen Zelle und Position Überprüfungen durchgeführt, um ein zufälliges Stück zu generieren.

Was ist Backtracking im Allgemein?

Backtracking ist eine Methode, um Probleme zu lösen, indem mögliche Lösungen schrittweise ausprobiert und ausgeschlossen werden, wenn sie nicht funktionieren. Dies ist ein sehr allgemeiner Ansatz und kann für eine Vielzahl von Problemen verwendet werden

Vorteile von Backtracking:

1- Es ist einfach zu verstehen und zu implementieren.
2- Es ist eine flexible Methode, die für viele verschiedene Probleme verwendet werden kann.
3- Es kann schnell eine Lösung finden, wenn die Anfangslösung nahe der endgültigen Lösung ist.

Nachteile von Backtracking:

1- Es kann sehr langsam sein, wenn es viele mögliche Lösungen gibt oder wenn die Lösung sehr tief in der Suchbaumstruktur versteckt ist.
2- Es benötigt viel Speicher, da es die gesamte Suchbaumstruktur aufbauen muss.
3- Es ist schwer, eine optimale Lösung zu garantieren, da es eine uninformierte Suche ist.

Da es mal sehr langsam mal zu schnell ist, wird der Algorithmus für das gelöste Spielfeld nicht verwendet.

3.3.2 Suchalgorithmus von den verbundenen Rohren

3.3.3

Für die Suche auf verbundene Rohre wird der Algorithmus „Tiefensuche“ angewandt.

Die Tiefensuche ist ein Suchalgorithmus für Graphen, bei dem man jeden Knoten durchläuft, indem man zunächst einen Kinderknoten auswählt und dann rekursiv dessen Kinderknoten durchläuft, bis alle Kinder besucht wurden. Dann kehrt man zurück und besucht den nächsten Kinderknoten. Dieser Prozess wiederholt sich, bis alle Knoten des Graphen besucht wurden.

Die Methode dazu befindet sich in der Klasse Game unter dem Namen dfs(). Diese Methode durchsucht rekursiv ein Gitter aus Zellen des Spielfeldes und findet verbundene Rohre, die durch Öffnungen in den Zellen miteinander verbunden sind. Die Methode speichert die gefundenen verbundenen Rohre in einer Liste (connectedPipes).

Die Methode nimmt fünf Argumente entgegen:

- ***currentPosition***: Die aktuelle Position, die untersucht wird.
- ***board***: Das 2D-Array von Zellen des Spielfeldes, das durchsucht wird.
- ***visited***: Ein 2D-Array von Boolean-Werten, das angibt, ob eine Zelle bereits besucht wurde.
- ***connectedPipes***: Eine Liste von Positionen, die die verbundenen Zellen enthält.
- ***overflow***: Ein Boolean-Wert, der angibt, ob das Gitter überläuft (torusförmig ist).

Die Methode überprüft jede Zelle und ihre Nachbarzellen auf Verbindungen (Öffnungen) und führt einen rekursiven Aufruf aus, wenn eine Verbindung vorliegt und die Nachbarzelle noch nicht besucht wurde.

Die Methode beendet sich selbst, wenn die aktuelle Position null ist. Am Ende der Methode wird die aktuelle Position als besucht markiert und zur Liste der verbundenen Zellen hinzugefügt.

Vorteile der Tiefensuche:

1- **Einfachheit**: Die tiefensuche-Methode ist einfach zu verstehen und zu implementieren.
2- **Effizienz**: Für Verbindungen, die tief im Gitter vorhanden sind, ist die DFS sehr effizient, da sie nicht den gesamten Speicher durchsucht, sondern nur den Speicher, der unmittelbar erreichbar ist.
3- **Unterstützung von rekursiven Lösungen**: DFS ist gut geeignet für Probleme, die sich gut auf eine rekursive Lösung unterstützen lassen.

Nachteile der Tiefensuche:

1- **Speichernutzung**: DFS verwendet eine rekursive Implementation, die den Speicher aufgrund von Rekursionsaufrufen nutzen kann.
2- **Zeitaufwendigkeit**: Für große Probleme kann die DFS in Bezug auf Zeitaufwendigkeit ineffizient sein, da sie möglicherweise eine sehr große Anzahl von Rekursionsaufrufen ausführt.
3- **Möglicherweise nicht die beste Lösung:** DFS ist möglicherweise nicht die beste Lösung für alle Probleme, und es kann bessere Algorithmen geben, die auf bestimmte Probleme abgestimmt sind.

3.4 Programmorganisationsplan

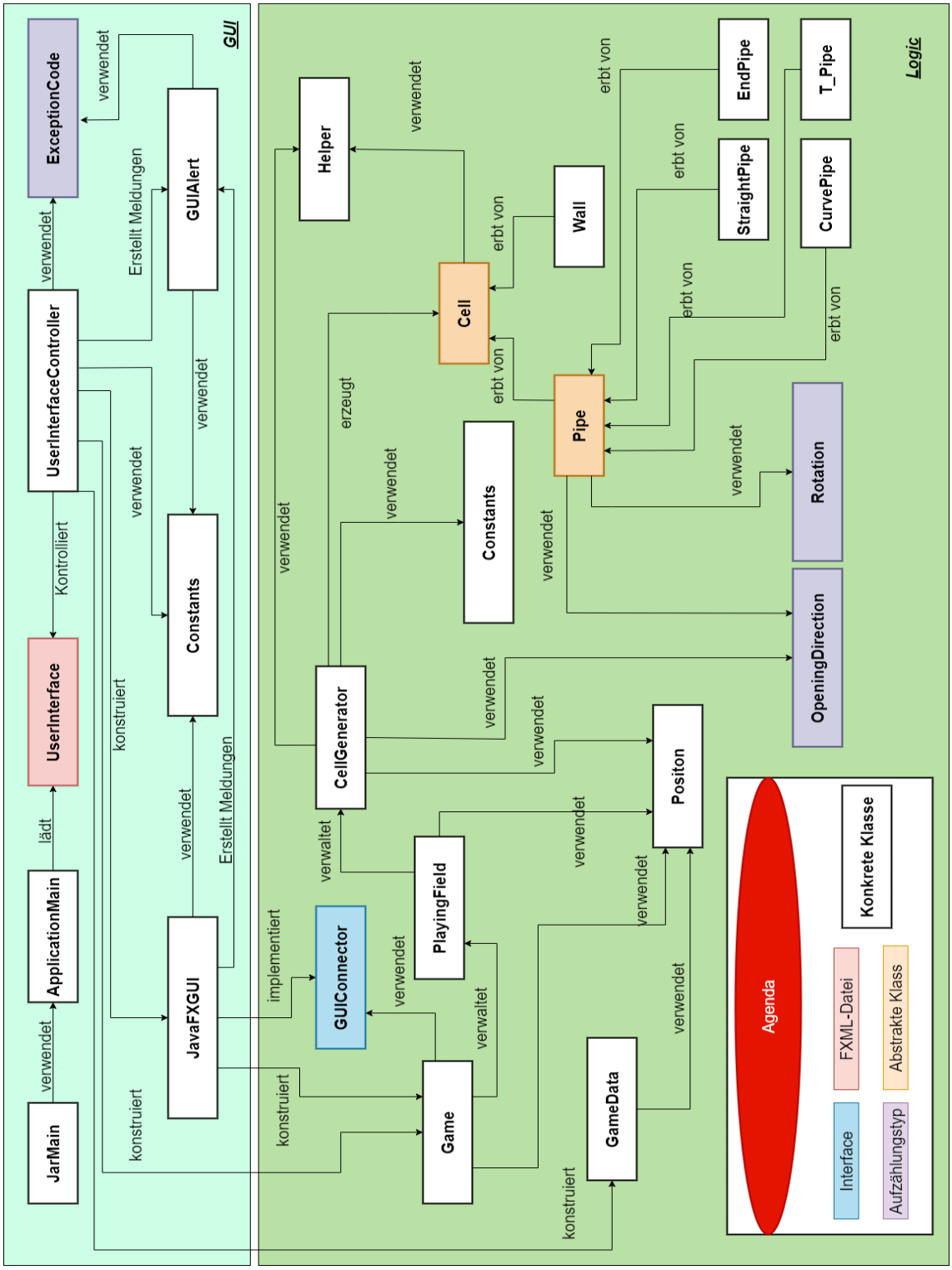

Abbildung 12: Grafische Darstellung der Klassen (Programmorganisationsplan)

Der Programmorganisationsplan wurde manuell mithilfe der Software “draw.io” erzeugt.

Der Programmorganisationsplan ist in zwei Bereiche unterteilt. Die farbliche Kennzeichnung ermöglicht hier eine einfache Zuordnung der Klassen zu den Paketen. Durch die Beschriftung der Pfeile lässt sich schnell nachvollziehen, in welchem Zusammenhang die Klassen stehen.

1- GUI

Dieser Bereich beinhaltet dabei die Klassen und FXML-Dateien, die zur Anzeige der grafischen Oberfläche (View) notwendig sind. Die ***JarMain*** Klasse ist nichts anders als einen Treiber, der das Spiel startet. Die ***ApplikationMain*** lädt die grafischen Komponenten, die in der FXNL-Datei ***UserInterface*** zu finden ist. Diese ***UserInterface*** wird dann kontrolliert von der Klasse ***UserInterfaceController***. Der Kontroller erstellt Meldungen mithilfe von der Klasse ***GUIAlert,*** die Dialoge zum Benutzer anzeigt. Der Kontroller konstruiert die GUI von der Klasse ***JavaFxGUI***, sowie das Spiel von der Klasse Game (nur beim Laden eines Spiels von einer externen JSON-Datei), und konstruiert eine Instanz von der Klasse ***GameData***, um die Daten von einer externen JSON-Datei zu extrahieren, um ein neues Spiel zu starten.

Die Klasse ***JavaFxGUI*** implementiert die Schnittstelle (Interface) GUIConector vom Paket „*logic*“. Damit kann die View mit dem Spiel der Klasse ***Game*** über das gemeinsame ***GUIConnector*** kommunizieren. Diese Klasse kann aber auch Meldungen mithilfe von der Klasse ***GUIAlert*** erstellen.

Die Klasse Constants ist dafür gedacht, Konstanten zu speichern. Sie enthält nur Strings, die von anderen Klassen verwendet werden.

1- Logic

Der Bereich “*Logic*” beinhaltet Klassen und Strukturen, die für das logische Verarbeiten von Spielvorgängen im Backend notwendig sind.

Die Klasse Game verwaltet die Klasse ***PlayingField***, der in seiner Rolle die Klasse ***CellGenerator*** verwaltet. Die Klasse ***CellGenerator*** erzeugt ein gelöstes Spielfeld von der Superklasse ***Cell***, die von den darunter stehenden Klassen geerbt wird.

Die Klassen Position und Helper sind Hilfsklassen, die von anderen Klassen verwendet werden, um möglichst die Codeverdopplung vermieden zu werden.

Die Klasse ***GameData*** ist für das Lesen und Schreiben von externen Dateien implementiert, die von der Klasse ***UserInterfaceController*** konstruiert wird.

3.5 Dateien

In dieser Software wird bisher nur mit JSON-Dateien bearbeitet, die Spielzustände enthalten. Damit Spiele gespeichert werden können, um später wieder gespielt zu werden. Das passiert, weil das Spiel kann wiedergeladen werden, um weiter gespielt zu werden.

Die Datei enthält dafür folgende Informationen bzw. Spielzustände:

- "source": Dieser Abschnitt beschreibt die Koordinaten der Quelle des Spielfeldes.
    - "x" repräsentiert die horizontale Koordinate und
    - "y" repräsentiert die vertikale Koordinate.

- "overflow": Dies ist ein Boolean-Wert, der angibt, ob das Spielfeld überläuft oder nicht.
    - "false" bedeutet, dass das Spielbrett nicht überläuft,
    - "true" bedeutet, dass es überläuft.

- "board": Dies ist eine 2D-Liste, die das eigentliche Spielfeld darstellt.
    - Jede Liste innerhalb dieser Liste repräsentiert eine Spalte des Spielfeldes und
    - jeder Wert innerhalb dieser Listen repräsentiert den Wert einer bestimmten Zelle,
    - daher repräsentiert die Länge einer Liste im zweiten Ebene die Anzahl der Zeilen des Spielfeldes und
    - die Anzahl von Listen innerhalb des ersten Ebenen repräsentiert die Anzahl der Spalten des Spielfeldes.

Da die Werte der Liste „board“ vom Datentyp int sind, erfolgt eine Umwandlung zu Konkreten Zellen mit Öffnungsrichtungen. Diese Umwandlung wurde schon im Abschnitt (3.2.4) erläutert.

So eine JSON-Datei sieht wie folgt aus:

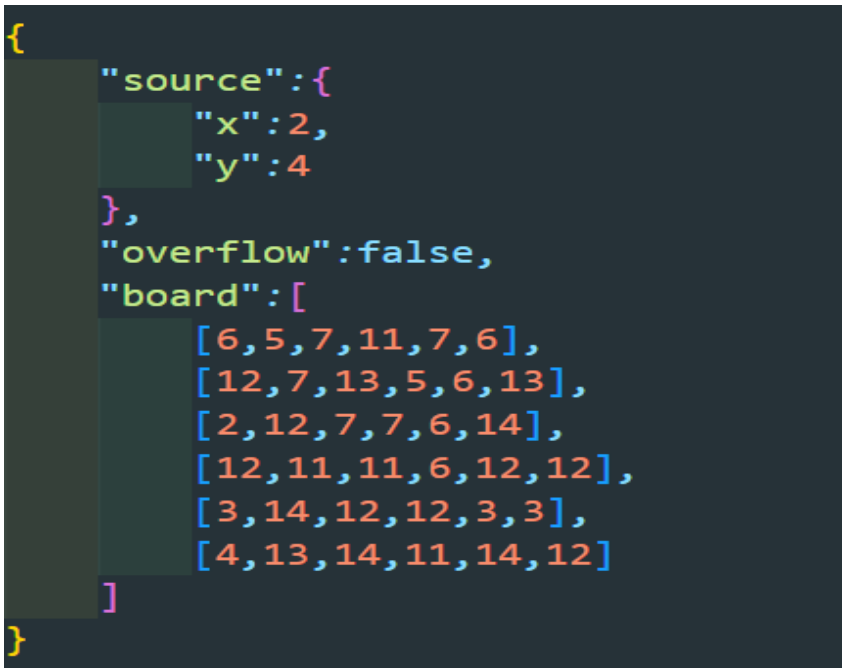

Abbildung 13: Gültige JSON-Datei für die Spielzustände

3.6 Programmtests

TestNr	Testfall	Ergebnis
1	Öffne die Applikation >> drücke neues Spiel Starten.	Neues Spiel mit Standard-Einstellungen erfolgreich gestartet und kann reibungslos gespielt werden.
2	Öffne die Applikation >> setze die Anzahl der Reihen auf 15 >> setze die Anzahl der Spalten auf 15 >> selektiere den Überlaufmodus >> setze max. Mauerstücke auf 0 >> drücke neues Spiel Starten.	Neues Spiel mit ausgewählten Einstellungen (ohne Mauerstücke) erfolgreich gestartet und kann reibungslos gespielt werden.
3	Führe die Testnummer 1 durch >> drücke das Menü „Spiel“ >> drücke neues Spiel starten >> drücke ja.	Der Benutzer wird gewarnt, dass dadurch Datenverlust entsteht. Nachdem der Benutzer „ja“ gedrückt, wird es zum ersten Fenster zurückgekehrt, um ein neues Spiel mit der Möglichkeit neue Einstellungen auszuwählen.
4	Führe die Testnummer 1 durch >> drücke das Menü „Spiel“ >> drücke neues Spiel starten >> drücke nein.	Der Benutzer wird gewarnt, dass dadurch Datenverlust entsteht. Nachdem der Benutzer „nein“ gedrückt, wird es im selben Fenster bleiben. Das Spiel kann weitergespielt werden.
5	Führe die Testnummer 1 durch >> drücke das Menü „Spiel“ >> drücke Spiel beenden >> drücke ja.	Der Benutzer wird gewarnt, dass dadurch Datenverlust entsteht. Nachdem der Benutzer „ja“ gedrückt, wird die Applikation geschlossen.
6	Führe die Testnummer 1 durch >> drücke das Menü „Spiel“ >> drücke Spiel beenden >> drücke nein.	Der Benutzer wird gewarnt, dass dadurch Datenverlust entsteht. Nachdem der Benutzer „nein“ gedrückt, wird es im selben Fenster bleiben. Das Spiel kann weitergespielt werden.

TestNr	Testfall	Ergebnis
7	Führe die Testnummer 1 durch >> drücke das Menü „Animation“ >> versuche einige Verbindungen zwischen den Rohren zu bilden >> drehe die Wasserquelle so, dass die verbundenen Rohre nicht mehr verbunden sind >> drehe die Quelle damit sie mit den anderen Rohren verbunden ist >> merke die Animationsgeschwindigkeit im Auge.	Die Rohre werden mit dem Standard-Animationsgeschwindigkeit reibungslos wie es im Abschnitt (3.2.11) angefordert animiert
8	Führe die Testnummer 7 durch >> Animationsgeschwindigkeit schneller (nach links schieben).	Die Rohre werden schneller animiert
9	Führe die Testnummer 7 durch >> Animationsgeschwindigkeit langsamer (nach rechts schieben).	Die Rohre werden langsamer animiert
10	Führe die Testnummer 1 durch >> drücke das Menü „Editormodus“ >> selektiere Editormodus.	Die Tools vom Editormodus werden angezeigt. Die aktuelle Spielfeldbelegung wird beim Wechsel in den Editormodus übernommen. Das Resultat ist weg.
11	Führe die Testnummer 10 durch >> ziehe ein Stück auf eine Zelle des Spielfeldes und lasse es los.	Die Zelle ist erfolgreich mit dem losgelassenen Stück belegt.
12	Führe die Testnummer 11 durch >> drehe die Zelle, die mit dem losgelassenen Stück belegt ist.	Die Zelle wird erfolgsreich gedreht, und auch zur Quelle verbunden.
13	Führe die Testnummer 10 durch >> verkürze die Reihen- bzw. Spaltenanzahl, so dass die Wasserquelle erhalten bleibt >> drücke Einstellungen umsetzen.	Die Reihen- bzw. die Spaltenanzahl werden erfolgreich verkürzt. Die Wasserquelle bleibt an derselben Position erhalten.

TestNr	Testfall	Ergebnis
14	Führe die Testnummer 13 durch >> drehe die Zellen, so dass Verbindungen mit der Wasserquelle gebildet werden.	Die Zellen können reibungslos verbunden werden.
15	Führe die Testnummer 10 durch >> verkürze die Reihen- bzw. Spaltenanzahl, so dass die Wasserquelle weg ist >> drücke Einstellungen umsetzen.	Die Reihen- bzw. die Spaltenanzahl werden erfolgreich verkürzt. Die Wasserquelle ist gelöscht. Keine Rohre sind verbunden.
16	Führe die Testnummer 15 durch >> ziehe eine Wasserquelle auf eine Zelle des Spielfeldes, die mit einem Rohr belegt ist und lasse sie los.	Die Wasserquelle wird an der richtigen Position belegt. Die Rohre können verbunden werden.
17	Führe die Testnummer 16 durch >> ziehe eine Wasserquelle auf eine Zelle des Spielfeldes, die mit einem Rohr belegt und keine Wasserquelle ist und lasse sie los.	Die alte Wasserquelle wird gelöscht. Die neue Wasserquelle wird an der richtigen Position belegt. Die Rohre können ausschließlich mit der neuen Wasserquelle verbunden werden.
18	Führe die Testnummer 17 durch >> ziehe ein Mauerstück auf Wasserquelle auf eine Zelle des Spielfeldes, die mit einem Rohr belegt und Wasserquelle ist.	Das Mauerstück wird an der richtigen Position auftauchen. Die Wasserquelle wird gelöscht. Keine Rohre sind verbunden.
19	Führe die Testnummer 13 durch >> selektiere den Überlaufmodus >> drücke Einstellungen umsetzen.	Das Spielfeld überläuft reibungslos durch die Ränder
20	Führe die Testnummer 10 durch >> vergrößere die Reihen- bzw. Spaltenanzahl >> drücke Einstellungen umsetzten.	Das Spielfeld wird vergrößert. Neue Zellen werden mit Mauerstücken belegt.

TestNr	Testfall	Ergebnis
21	Führe die Testnummer 20 durch >> erstelle ein Spielfeld auf eigenen Wunsch >> drücke aktuelles Feld speichern >> drücke nein.	Ein Warnungsdialog taucht auf. Nachdem der Benutzer „nein“ gedrückt, ist der Dialog weg und der Benutzer kann weiter mit dem Editormodus arbeiten.
22	Führe die Testnummer 20 durch >> erstelle ein Spielfeld auf eigenen Wunsch >> drücke aktuelles Feld speichern >> drücke ja.	Ein Warnungsdialog taucht auf. Nachdem der Benutzer „ja“ gedrückt, Sind die Tools vom Editormodus weg. **BUG**: Das Resultat ist weggeblieben. Lösung: >> lblResultat.setVisible(true) beim Handler einfügen.
23	Führe die Testnummer 10 durch >> drücke aktuelles Feld löschen >> drücke nein.	Ein Warnungsdialog taucht auf. Nachdem der Benutzer „nein“ gedrückt, ist der Dialog weg und der Benutzer kann weiter mit dem Editormodus arbeiten.
24	Führe die Testnummer 10 durch >> drücke aktuelles Feld löschen >> drücke ja.	Ein Warnungsdialog taucht auf. Nachdem der Benutzer „ja“ gedrückt, sind die Zellen des Spielfeldes gelöscht und mit Mauerstücken ersetzt.
25	Führe die Testnummer 24 durch >> ziehe ein paar Stücke und lasse sie auf die Zellen des Spielfeldes >> drehe die Zellen nach rechts und links auf eigenen Wunsch >> ziehe eine Quelle und lasse sie auf einem Rohr los.	Das Spielfeld arbeitet normal und reibungslos.
26	Führe die Testnummer 10 durch >> drücke die taste aktuelles Feld mixen.	Die Zellen des Spielfeldes werden zufällig gedreht.

TestNr	Testfall	Ergebnis
27	Führe die Testnummer 25 durch >> erstelle ein gelöstes Spielfeld (Alle Rohre mit ihren Übergängen sind verbunden).	Ein Informationsdialog taucht auf und informiert, dass das Spielfeld gelöst ist.
28	Führe die Testnummer 27 durch >> ziehe ein Mauerstück und lasse es auf die Wasserquelle los.	Die Wasserquelle ist gelöscht. Keine Rohre sind verbunden.
29	Führe die Testnummer 28 durch >> drücke die Taste aktuelles Feld speichern.	Ein Warnungsdialog taucht auf und sagt, dass es eine Wasserquelle eingesetzt werden soll, bevor das Spielfeld gespeichert werden darf.
30	Führe die Testnummer 27 durch >> drücke die Taste aktuelles Feld speichern.	Ein Warnungsdialog taucht auf. Nachdem der Benutzer „ja“ gedrückt, Sind die Tools vom Editormodus weg. **BUG**: Das Resultat ist weggeblieben. Lösung: >> lblResultat.setVisible(true) beim Handler einfügen.
31	Öffne die Applikation >> drücke altes Spiel laden >> navigiere zur beigefügten Datei „test_korrekt.json“ >> selektiere die Datei >> drücke öffnen.	Das Spielfeld wird angezeigt. Die Belegung des Spielfeldes ist korrekt (vergleiche sie mit der JSON-Datei). Das Spiel kann weitergespielt.
32	Führe die Testnummer 31 durch >> drücke das Menü „Spiel“ >> drücke Spiel speichern >> gebe einen Dateinamen ein >> drücke speichern >> drücke in Ordnung >> mache ein Screenshot vom Spielfeld.	Ein Informationsdialog taucht auf und informiert, dass die Spielzustände erfolgreich gespeichert geworden sind. Das Spiel kann weitergespielt werden.

TestNr	Testfall	Ergebnis
33	Öffne die Applikation >> drücke altes Spiel laden >> navigiere zur beigefügten Datei, die in der Testnummer 32 gespeichert wurde >> selektiere die Datei >> drücke öffnen.	Das Spielfeld wird angezeigt. Die Belegung des Spielfeldes ist korrekt (vergleiche sie mit Screenshot vom Test 32). Das Spiel kann weitergespielt.
34	Öffne die Applikation >> drücke altes Spiel laden >> navigiere zur beigefügten Datei „test_falsch_Json_Format.json“ >> selektiere die Datei >> drücke öffnen.	Die Datei wird nicht geladen, Fehlerdialoge tauchen auf mit entsprechenden Fehlercodes.
35	Öffne die Applikation >> drücke altes Spiel laden >> navigiere zur beigefügten Datei „test_falsch_illegalState.json“ >> selektiere die Datei >> drücke öffnen.	Die Datei wird nicht geladen, Fehlerdialoge tauchen auf mit entsprechenden Fehlercodes.
36	Öffne die Applikation >> drücke Spiel beenden.	Die Applikation wird geschlossen.

Anmerkungen:

1- Die Datei „test_falsch_Json_Format.json“ enthält ein falsches JSON-Format.Es gibt kein Komma nach dem Objekt „overflow“.

2- Die Datei „test_falsch_illegalState.json“ enthält eine falsche Zelle. Die erste Liste im „board“ ist gleich 40. Diese Zahl ist keinem Stück zuzuordnen.

4 Abbildungsverzeichnis: