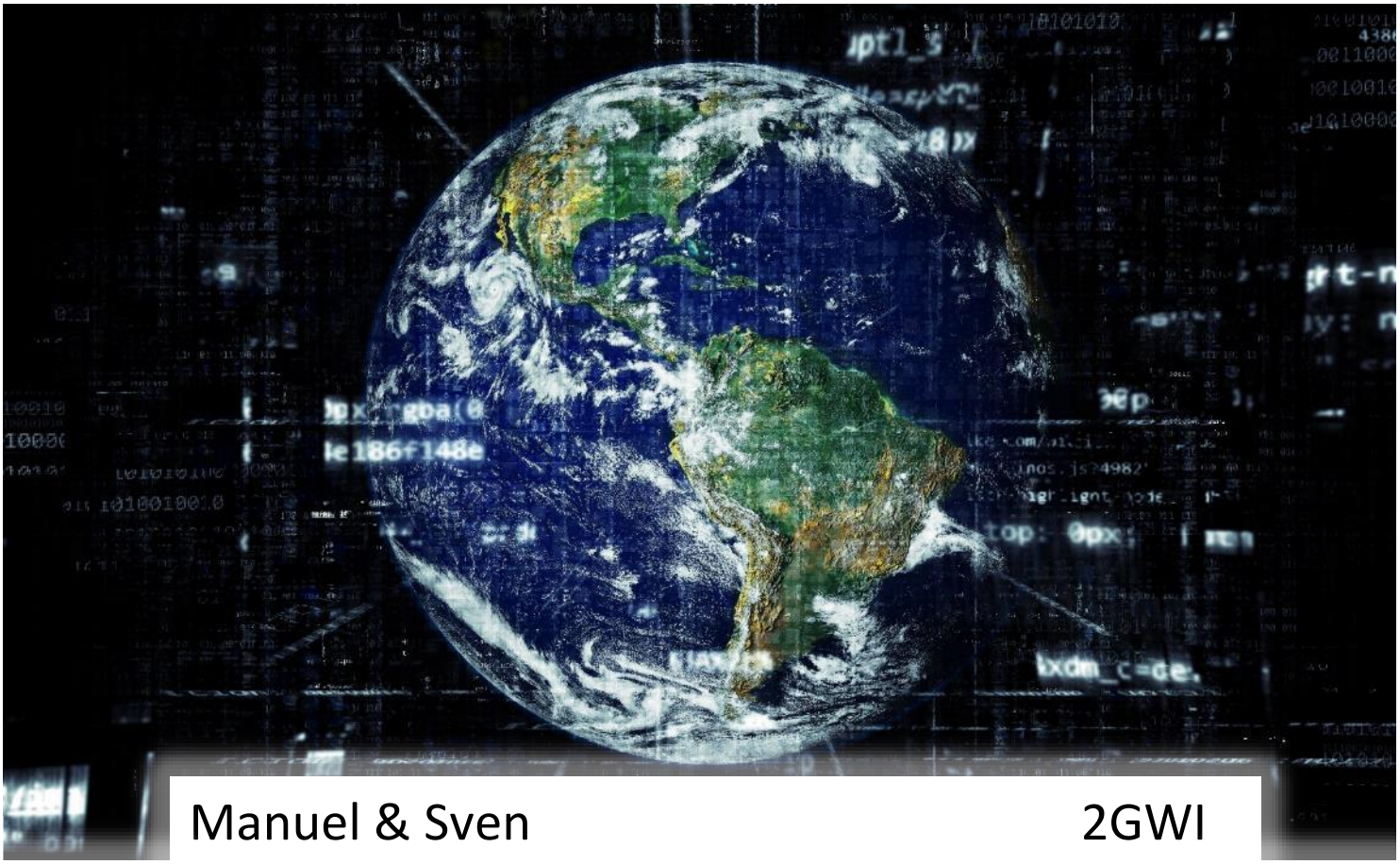


Manuel & Sven 2GWI

Snakeology 404V

04.01.2021

Inhaltsverzeichnis

1 Vorwort

1.1 Aufgabenstellung

In der Schule besuchen wir das Vertiefungsmodul 404V, in dem es um die Vertiefung der objektbasierten Programmierung mit der Programmiersprache Java geht. Wir haben den Auftrag bekommen, in Zweiergruppen eine Projektarbeit durchzuführen. In der Wahl des Projektthemas haben wir viel Spielraum bekommen. Die Bedingung ist, dass ein lauffähiges Programm mit der Programmiersprache Java erstellt und ein Filesystem integriert werden soll.

Folgende Daten sind fix festgelegt und müssen eingehalten werden.

Datum	**Ereignis**
20.11.2020	Zwischenpräsentation Projektarbeit
04.01.2021	Abgabe Java Projekt an Ueli Hagger bis 23:00 Uhr
15.01.2021	Präsentation Projektarbeit

1.2 Brainstorming und Projektauswahl

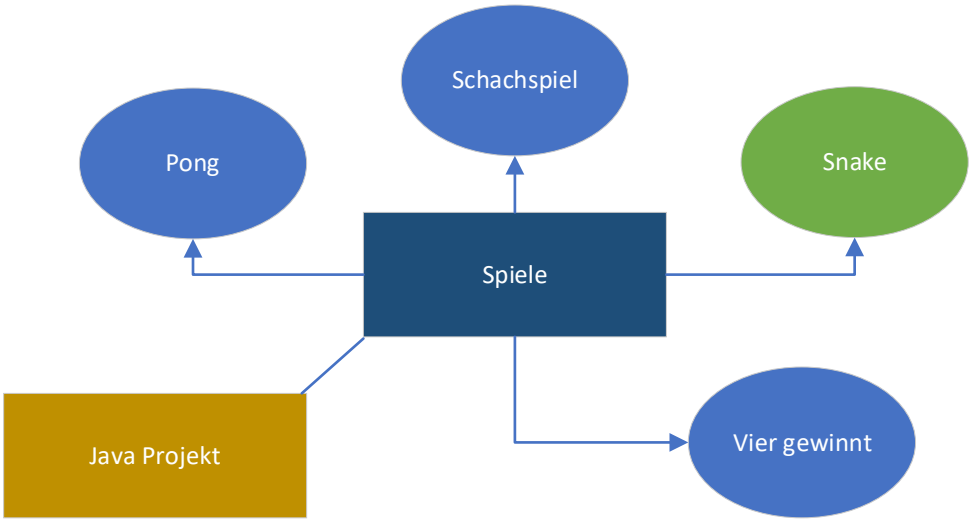


Abbildung 1: Ergebnisse Brainstorming

1.3 Motivation

Zuerst haben wir eine Weile gerätselt was wir machen sollten, da wir davon ausgingen, dass ein Spiel recht kompliziert wäre. Herr Hagger hat uns dann aber empfohlen ein Spiel zu machen. Deshalb haben uns recht schnell dafür entschieden, dass wir ein Spiel machten wollen. Danach haben wir einige Spiele aufgeschrieben, die unserer Meinung nach gut zu Programmieren und im Rahmen der Zeit sind. Diese sind: Pong, Snake, Vier gewinnt und Schach. Zuerst wollten wir unbedingt ein Schachspiel Programmieren. Nach Absprache mit unserer Lehrperson haben wir uns dann allerdings aus Zeit- und Schwierigkeitsgründen dagegen entschieden. Also haben wir bei unserem nächsten Brainstorming Wert darauf gelegt, ein Spiel auszusuchen, welches weniger Zeit zum Programmieren benötigt, aber trotzdem mit vielen Extras und Funktionen erweitert werden kann. Unsere erste Idee war Snake und da wir ein wenig unter Zeitdruck mit der Entscheidung standen und niemand etwas dagegen hatte haben wir uns dafür entschieden. Diese Entscheidung haben wir später auch nie bereut, da uns das Projekt viel Spass gemacht hat. Wir wollen versuchen, möglichst ohne Hilfe zu programmieren, da wir so umso mehr aus dem Projekt lernen können. Wenn wir auf unbekannte Funktionen oder Probleme stossen, werden wir uns natürlich im Internet darüber schlau machen und so auch viel neues Wissen ansammeln

1.4 Selbständigkeitserklärung

Hiermit bestätigen wir, Manuel Schmid und Sven Walser, dass die vorliegende Arbeit selbständig verfasst und keine anderen als die angegebenen Quellen und Hilfsmittel benutzt, sowie Zitate und gedankliche Übernahmen kenntlich gemacht wurden.

Brunnadern, 03.01.2021 

Goldach, 03.01.2021 

2 Pflichtenheft

Projektidee	**Snake-Spiel in Java implementieren**
Muss	• Spielfeld • Schlange muss mit Tastatur gesteuert werden können (Pfeile) • Schlange wird grösser beim Verzehr • Es werden zufällige Punkte verzehrt • Spielername soll eingegeben werden können. • Ein Highscore und dazugehöriger Spieler kann gespeichert werden (auch wenn das Spiel beendet wird) • Aktueller Punktestand wird angezeigt • Wenn man stirbt muss der Punktestand und die Schwanzlänge zurückgesetzt, und ein Replay-Button angezeigt werden.
Soll	• Zusätzlich steuerbar durch WASD
Kann	• Hindernisse die zufällig generiert werden • Beim in die Wand fahren soll die Schlange auf der anderen Seite herauskommen.
Aufwand	• Pro Person müssen mindestens 30 Stunden aufgewendet werden, da wir zu zweit sind, wären dies für uns 60 Stunden.

3 Planung

3.1 Anfangsplanung

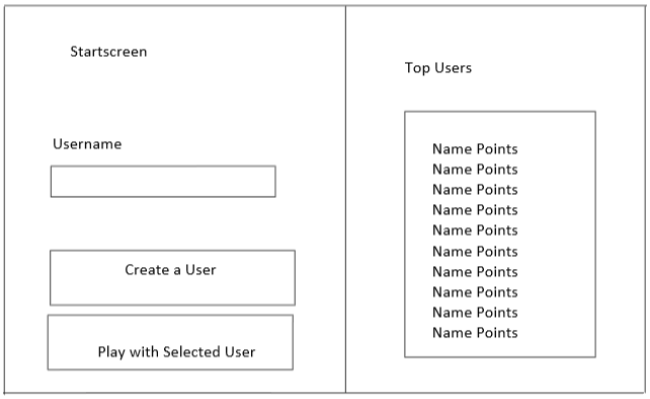
Da für uns beide die Programmierung von Spielen komplettes Neuland bedeutet, haben wir uns entschieden uns gemeinsam ein YouTube-Video zu der Erstellung eines Snake-Programms anzuschauen. Wir haben eine vierteilige Videoreihe gefunden (Siehe Anhang/Quellenverzeichnis/YouTube/ Video 1/4 bis Video 4/4), die insgesamt etwa eine Stunde lang ist und in der etappenweise erklärt wird, wie man ein Snake-Spiel programmieren könnte. Wir haben diese Videoreihe gemeinsam angeschaut und viele Notizen gemacht. Danach haben wir angefangen zu programmieren und haben sehr viel des neu Gelernten direkt umsetzen können. Anfangs sah unser Spiel dann fast identisch wie das aus dem Video aus, dies wollten wir aber noch ändern. Uns war sehr wichtig, dass unser fertiges Spiel sich sehr stark von dem aus dem Video unterscheidet und sich durch zahlreiche Verbesserungen, Änderungen, Zusatzfunktionen und andere Klassen abhebt. Ebenfalls haben wir, nachdem wir die Videos einmal angeschaut hatten, nicht

mehr darauf zugegriffen, sondern nur noch unsere Notizen und das Eingeprägte verwendet. Für die Arbeitsphasen in der Schule haben wir uns so organisiert, dass wir uns abwechseln, sodass jeweils einer an der Dokumentation schreibt, während der andere programmiert, damit wir effizienter arbeiten können.

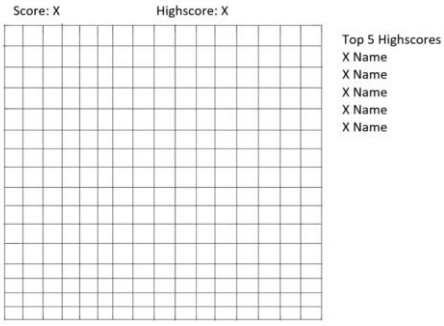
3.2 Grobentwurf

Um uns das Programm besser vorstellen zu können haben wir einige Grobentwürfe auf OneNote gezeichnet. Dies sind lediglich Entwürfe aus einem Brainstorming und müssen nicht zwingend so umgesetzt werden.

Startscreen



Spiel



Deathscreen

3.3 Feinentwurf

Unser Snake-Spiel soll ein optisch ansprechendes Spielfeld mit klar ersichtlichen Feldern haben. Der karierte Hintergrund soll mit grau getönten Feldern gefüllt werden, damit die Farben des Spiels optimal zu erkennen sind. Beim Start des Spiels muss ein Spielername eingegeben werden und der Start des Spiels soll mit einem Klick auf Start initialisiert werden. Wir wollen eine Tastatursteuerung implementieren, um die Schlange mit den Pfeiltasten navigieren zu können. Unsere Schlange soll grün sein. Es sollen quer über das Spielfeld verteilt an zufälligen Orten Punkte, also Äpfel, generiert werden, die von der Schlange verzehrt werden können. Beim Verzehr eines Apfels soll dieser verschwinden und an einer anderen Stelle ein neuer auftauchen. Für jeden verzehrten Apfel wächst der Schwanz der Schlange um ein paar Pixel und der Punktestand wird um eins erhöht. Dieser Punktestand soll zusammen mit dem Highscore am oberen Rand über dem Spielfeld angezeigt werden. Falls die Schlange stirbt wird das Spiel angehalten und der Punktestand und die Schwanzlänge werden zurückgesetzt, und ein Replay-Button angezeigt. Zudem wird die Position der Schlange auf die Startposition zurückgesetzt. Auf dem «Death-Screen» sollen die top fünf Highscores mit den dazugehörigen Spielernamen angezeigt werden. Dazu muss bei jedem Tod der Schlange die erreichte Punktzahl und der eingegebene Spielername in ein File abgespeichert werden, falls die Punktzahl grösser als einer der Highscores ist. Die Highscore-Liste soll nummeriert und übersichtlich sein. Zusätzlich neben den Pfeiltasten sollen auch noch die Tasten WASD zur Steuerung verwendet können. Wir haben uns noch ein paar zusätzliche Funktionen überlegt, die wir implementieren wollen, falls wir noch Zeit dafür finden, die aber nicht essenziell für das Projekt sind. Eines davon wäre, dass wenn die Schlange in eine Wand fährt, sie auf der gegenüberliegenden Seite des Spielfelds wieder herauskommen soll. Zusätzlich könnten wir auch noch zufällige Hindernisse erscheinen lassen, die bei Berührung den direkten Tod der Schlange bewirken. Wir haben uns entschieden, die Oberfläche des Spiels vollständig in Englisch zu gestalten.
Diese Aufgabe soll von zwei Personen bearbeitet werden und einen gemeinsamen Zeitaufwand von 60 Stunden beanspruchen.

3.4 Ressourcenplanung

Wir wollen bei unserem Projekt Eclipse als Entwicklungsumgebung verwenden da wir dieses Programm bereits in die Schule kennengelernt und verwendet haben. Um den Überblick über unsere Fortschritte bei der Programmierung des Spiels zu gewährleisten, wollen wir Git und Github verwenden, um mit Hilfe von Commits jeden Schritt einzeln abspeichern und die Änderungen überprüfen zu können. OneNote haben wir benutzt, um die Skizzen für den Grobentwurf zu zeichnen, da man dort auch gut von Hand zeichnen kann. Visio haben wir nur am Anfang verwendet, um unsere Ergebnisse des Brainstormings für das Projekt

grafisch darzustellen. Natürlich haben wir auch Word gebraucht, um eine Dokumentation für unser Projekt zu erstellen.

3.5 Terminplanung

Wir haben uns vorgenommen, die Planung möglichst ausführlich zu gestalten, damit uns die Umsetzung leichter fällt. Die Planung soll bis spätestens am 06.11.2020 abgeschlossen sein. Die erste lauffähige Version des Spiels soll spätestens eine Woche vor der Zwischenpräsentation einsatzbereit sein, um den erreichten Fortschritt vorweisen zu können. Die weitere Programmierung soll hauptsächlich im Dezember stattfinden, da wir anfangs Dezember noch wenig Prüfungen haben und Ende Dezember können wir die Ferien gut nutzen. Wir gehen davon aus, dass wir mindestens 40 Prozent des Projekts in den Weihnachtsferien erstellen. An der Dokumentation werden wir ebenfalls in den Ferien viel weiterarbeiten.

3.6 Arbeitsaufteilung

Bei der Dokumentation haben wir uns die Arbeit so aufgeteilt das jeder etwa 50 Prozent übernimmt. Beim Programmieren sind wir aber auf ein Problem gestossen, da das gemeinsame Programmieren an einem verhältnismässig eher kleinen Projekt sich als sehr umständlich erwies. Also haben wir uns dafür entschieden das der grösste Teil des Programms von einer Person programmiert wird und der andere bei ihm sitzt und versucht ihm zu helfen und ansonsten an der Dokumentation weiterarbeitet. Wir werden allerdings sehr gut darauf achten, dass nicht das ganze Projekt von einer Person programmiert wird, denn die zweite Person wird bei jeder Gelegenheit, bei der eine Klasse oder Methode unabhängig vom Programm erstellt werden kann, zum Einsatz kommen und diese voll ausprogrammieren. So können beide stark am Projekt mitarbeiten und davon profitieren. Wir werden auf den ersten Zeilen jeder Klasse den Autor festhalten.

4 Dokumentation

4.1 Durchführung

4.1.1 Projektstruktur

Wir haben von Anfang an speziell darauf geachtet, die Struktur des Codes und der entsprechenden Klassen möglichst übersichtlich zu halten. Dies hat sich mit der, im Verlauf des Projekts zunehmenden Anzahl an verschiedenen Klassen als sehr hilfreich erwiesen. Wir haben zusammengehörige Komponenten des Spiels in Packages gegliedert, was auch die gemeinsame Arbeit am Programm erleichtert hat.

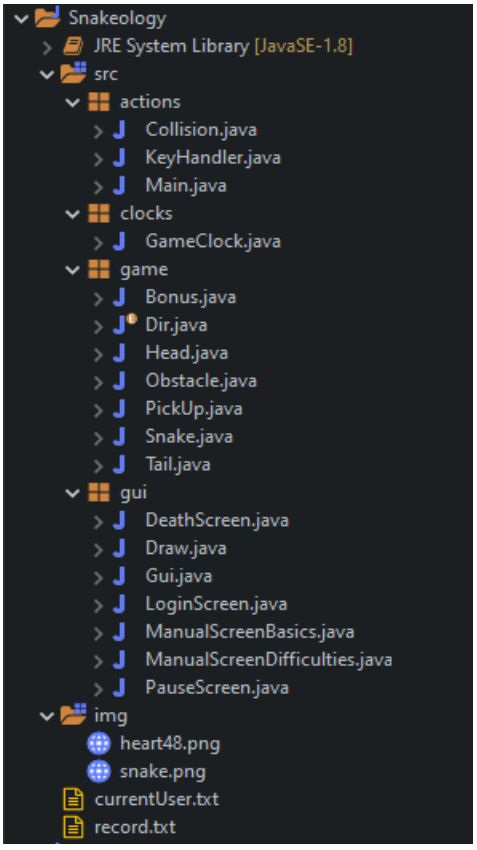


Abbildung 2: Projektstruktur des Java-Projekts

4.1.2 Sicherungsstruktur (Git)

Um unseren Projektfortschritt gleichzeitig sichern und verfolgen zu können, haben wir mit einem Github-Repository gearbeitet. Die entsprechenden Commits haben wir mit Git über die Kommandozeile durchgeführt. Sobald ein oder mehrere Arbeitsschritte abgeschlossen waren haben wir die Änderungen mit dem Master-Branch zusammengeführt. Zum Projektschluss sind wir damit auf etwa 20 Commits gekommen. Nachfolgend haben eine Abbildung der ersten neun Commits als Beispiel eingefügt. Unser vollständiges GitHub-Repository haben wir im Quellenverzeichnis unter «6.2.1, GitHub-Repository» verlinkt.

Abbildung 3: GitHub-Commits

4.2 Umsetzung GUI

4.2.1 Login-Screen

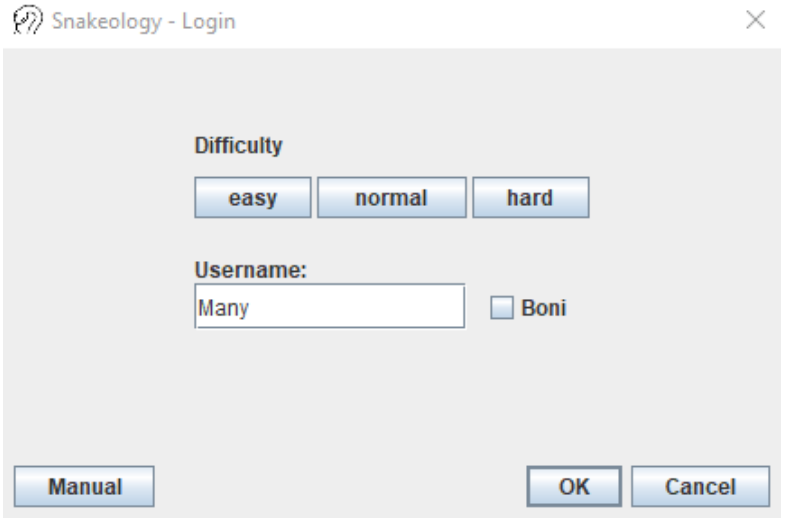


Abbildung 4: Login-Screen

Sobald man das Programm startet, wird dieser Login-Screen in der Mitte des Bildschirms angezeigt. Man kann die gewünschte Schwierigkeitsstufe anklicken, wodurch dieser `JButton` deaktiviert wird, so kann man zwischen den Schwierigkeitsstufen wechseln. Danach kann man seinen Benutzernamen in ein `JTextField` eingeben, der lokal in einer Datei abgespeichert wird. Wenn das Feld «Username» leer gelassen wird, wird beim Klick auf «Ok» eine Fehlermeldung angezeigt. Zum Schluss kann man mit einer `JCheckBox` entscheiden, ob man die Boni im Spiel aktivieren oder deaktivieren möchte, standardmässig ist das Feld deaktiviert. Mit dem Button «Manual» kann die Spielanleitung aufgerufen werden. (siehe Abbildung 8) Beim Klick auf «Cancel» wird das Programm wieder beendet und mit Ok wird man zum Spiel weitergeleitet.

4.2.2 Spieloberfläche ohne Boni

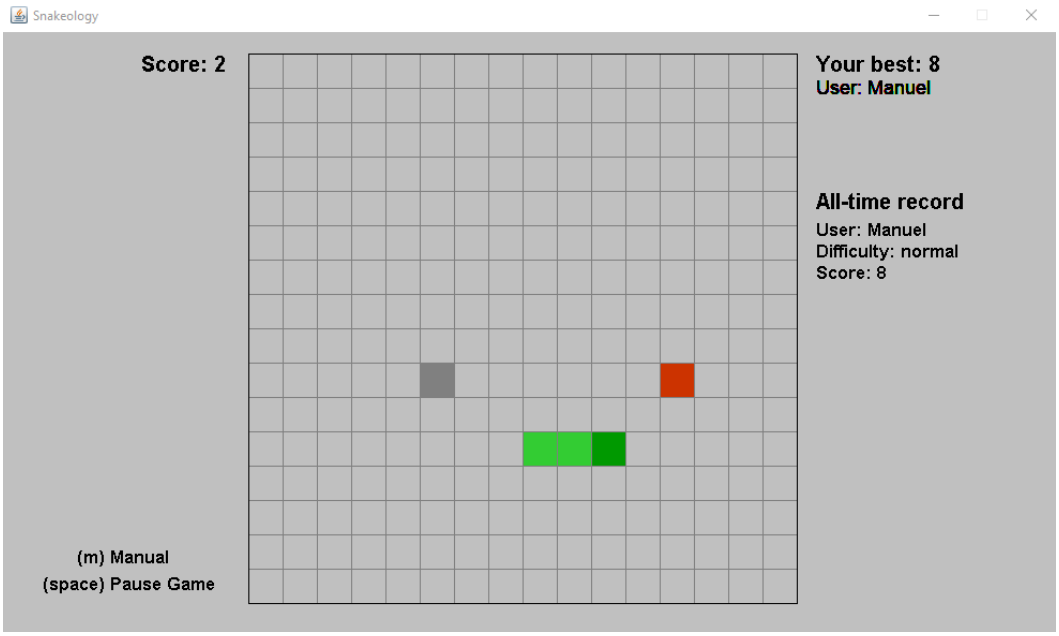


Abbildung 5: Snakeology (ohne Boni)

Wenn man weitergeleitet wird, beginnt das Spiel direkt. Wenn man die Boni deaktiviert, wird der oben abgebildete `JFrame` angezeigt. Im Spielfeld ist grün die Schlange und rot ein Apfel zu erkennen. Immer wenn die Schlange einen solchen Apfel isst, erhöht sich die Punktzahl um eins und ein neuer Apfel erscheint. Die Texte um das Spielfeld herum sind als mit `JLabel` definiert. In der Oberen linken Ecke ist die aktuelle Punktzahl zu sehen, die durch die Anzahl der hellgrünen Schwanzteile der Schlange repräsentiert wird. Oben rechts hingegen sieht man die beste Erreichte Punktzahl seit dem letzten Start des Spiels und direkt darunter wird der eingegebene Benutzername angezeigt. Ebenfalls rechts wird der «All-time record» angezeigt, dort wird der Spieler mit dem höchsten erreichten Punktestand verewigt. Gespeichert werden der Benutzername, die verwendete Schwierigkeitsstufe beim Erreichen des Rekords und die genaue Anzahl Punkte. Unten links wird erwähnt, dass man sich durch Drücken der Taste «m» eine Anleitung zum Spiel ansehen (siehe Abbildung 8) und mit der Leertaste das Spiel pausieren kann.

4.2.3 Spieloberfläche mit Boni

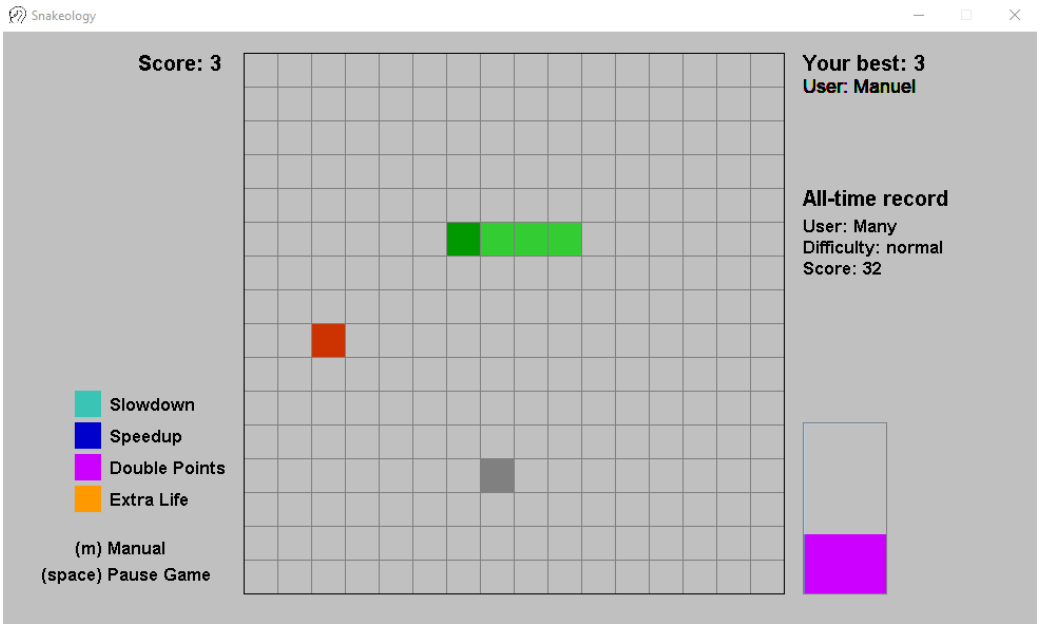


Abbildung 6: Snakeology (mit Boni)

Wenn man im Login-, oder im Death-Screen die Boni aktiviert und das Spiel beginnt wird dieser erweiterte `JFrame` angezeigt. Der `JFrame` ist der gleiche wie ohne Boni, nur das ein paar Erweiterungen eingebaut sind. Am linken Rand der Abbildung sieht man die Erklärung, was die vier verschiedenen, unterschiedlich eingefärbten Boni bewirken. Die Boni funktionieren an sich genau wie PickUps bzw. Äpfel, man fährt in einen hinein und er verschwindet und der Bonus wird aktiviert. Die Effekte der Boni sind voll ausprogrammiert und funktionsfähig. Sobald ein Bonus aktiviert wurde ist er für eine bestimmte Zeit lang aktiv, bis die Zeit abläuft und ein neuer Bonus auf dem Spielfeld erscheint. Diese verbleibende Zeit, in der der Bonuseffekt noch anhält, haben wir mit Hilfe einer `JProgressBar` in der rechten unteren Ecke visualisiert. «Die Klasse `JProgressBar` dient dazu, einen Fortschrittsbalken zu schaffen.» (Petri, 2020) Im Easy-Mode oder durch den entsprechenden Bonus kann ein Extraleben aktiviert werden, welches durch ein Bild eines Herzes dargestellt wird.

4.2.4 Pause-Screen

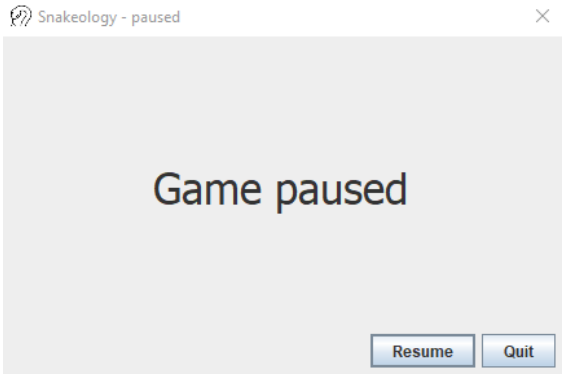


Abbildung 7: Pause-Screen

Der Pause-Screen kann, während das Spiel läuft durch Drücken der Taste «Escape» oder der Leertaste aufgerufen werden. Das Spiel wird pausiert und der oben abgebildete `JDialog` angezeigt. Nun kann man mit Klick auf «Resume» das Spiel weiterlaufen lassen und somit den `JDialog` schliessen oder mit «Quit» das ganze Spiel beenden.

4.2.5 Manual-Screen (Spielanleitung)

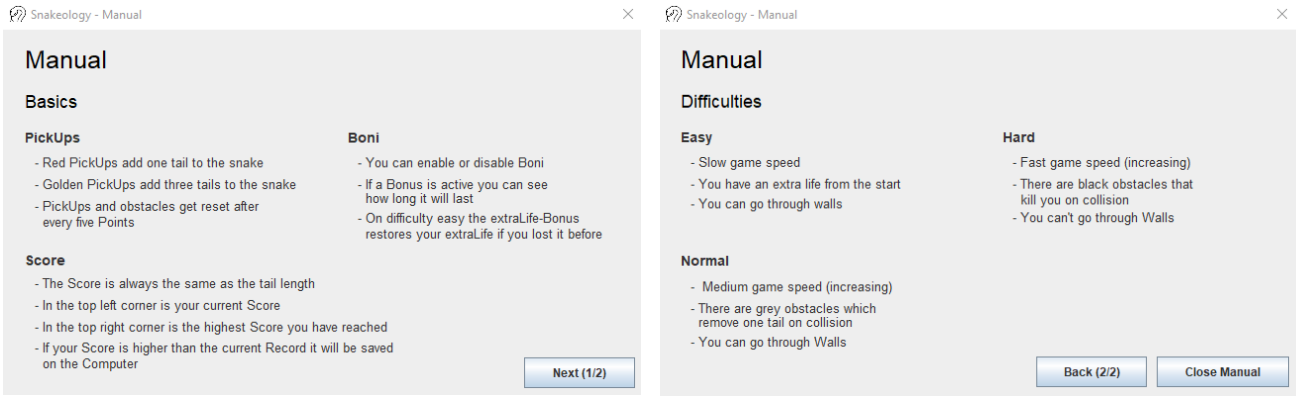


Abbildung 8: Manual-Screen 1 und 2

Diese zwei `JDialog`-Fenster haben keine programmatische Funktion, sie sind nur dazu da, dem Benutzer das Spiel und seine Grundregeln zu erklären. Diese Spielanleitung kann auf zwei verschiedene Arten aufgerufen werden. Entweder durch Knopfdruck auf «Manual» im Login-Screen oder während des Spiels durch die Taste «m». Wenn die Anleitung von Login-Screen aus gestartet wurde, sieht sie genauso aus wie in der obigen Abbildung und mit Schliessen der Anleitung kommt man wieder zurück zum Login-Screen. Wenn sie allerdings während des Spiels aufgerufen wurde findet man, anstatt dem «Close Manual»-Button, einen anderen Button an der gleichen Stelle vor mit dem Text «Continue Game», dieser schliesst die Anleitung und lässt das Spiel weiterfahren.

4.2.6 Death-Screen

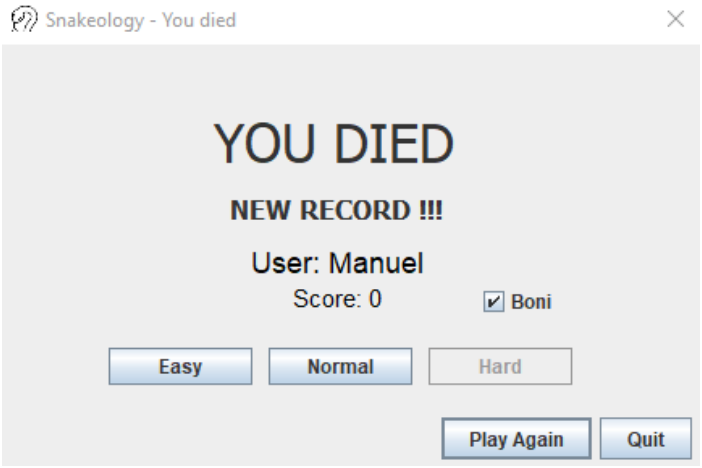


Abbildung 9: Death-Screen

Der Death-Screen wird immer dann aufgerufen, wenn die Schlange während des Spiels auf irgendeine Weise stirbt. Mit `JLabels` wird einem erklärt, dass man gestorben ist und falls in der gerade gespielten Runde eine neue Rekordpunktzahl erreicht wurde, wird das `JLabel` «New Record !!!» angezeigt. Danach kann man, falls man es nochmal versuchen möchte, erneut eine Schwierigkeitsstufe auswählen und die Boni de-, oder aktivieren. Mit einem Klick auf den `JButton` «Play Again» geht das Spiel mit den gerade angepassten Einstellungen von vorne los. Mit «Quit» wird das ganze Spiel beendet.

4.3 Wichtige Codesegmente

4.3.1 Main-Methode

Für die Main-Methode haben wir eine eigene Klasse erstellt. «Eine Methode `main()` muss jede Java-Anwendung besitzen. Sie stellt den Einstiegspunkt in die Ausführung einer Java-Anwendung dar und muss die Signatur `public static void main(String[] args)` besitzen.» (javabeginners.de, 2020) Da bei unserem Spiel als erstes das Login-Fenster erscheinen soll, wird hier ein solches erstellt, alle weiteren Funktionen verteilen sich über die anderen Klassen.

```
public class Main {

    public static void main(String[] args) throws IOException {
        // Beim Start des Programms wird der LoginScreen angezeigt.
        LoginScreen loginS = new LoginScreen();
        loginS.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
        loginS.setVisible(true);

    }
}
```

Abbildung 10: [Code] Main-Methode

4.3.2 Game-Clock: Run-Methode

In der nachfolgenden Abbildung ist ein Ausschnitt aus der Run-Methode der Game-Clock zu sehen. Das Spiel verläuft nach Ticks, mit denen jeweils der ganze Code ausgeführt wird, die Aufgabe der Game-Clock ist es zu bestimmen, wie lange diese Ticks auseinanderliegen. Diese Wartezeit wird mit dem `Sleep()` Befehl geregelt, der den aktuellen Thread anhält. (codeflow.site, 2020) Der Code ist nur ein Ausschnitt, da die Run-Methode sehr viel komplizierten Code mit Verschachtelungen beinhaltet, aber dieser Ausschnitt repräsentiert am besten den Zweck der Methode.

```
} else if (difficulty == "hard") { // Geschwingkeit im Hard-Mode
    if (Snake.score < 10) { // Schlange wird mit höherer Punktzahl schneller
        sleep(170);
    } else if (Snake.score < 15) {
        sleep(155);
    } else if (Snake.score < 20) {
        sleep(140);
    } else if (Snake.score < 25) {
        sleep(125);
    } else {
        sleep(110);
    }
}
```

Abbildung 11: [Code] Game-Clock: Run-Methode

4.3.3 Game-Clock: Collision-Handling

In der Game-Clock werden mit jedem Tick auch alle möglichen Kollisionen mit Objekten aufgerufen, die mit den gewählten Einstellungen nötig sind. Das Aufsammeln von Äpfeln und die Kollision mit sich selbst und mit Wänden muss in jedem Fall abgefragt werden. Normale und Schwarze Hindernisse gibt es allerdings nur im Normal-, und Hard-Mode, deshalb werden sie nur dann aufgerufen. Die Kollision mit Boni ist nur notwendig, wenn der Benutzer die Boni auch aktiviert hat.

```
// Alle Kollisionen werden aufgerufen
Collision.collidePickUp();
Collision.collideSelf();
Collision.collideWall();
if (difficulty == "hard" || GameClock.difficulty == "normal" ) { // Im Easy-Mode gibt es keine Hindernisse (Obstacles)
    Collision.collideNormalObstacle();
    Collision.collideBlackObstacle();
}
if (boniOn == true) { // Muss nur aufgerufen werden, wenn Boni aktiviert sind
    Collision.collideBonus();
}
```

Abbildung 12: [Code] Game-Clock: Collision-Handling

4.3.4 PickUp-Objekt (Apfel)

Wir stellen hier stellvertretend für alle auf dem Spiel erscheinenden Objekte, also Äpfel, Hindernisse und Boni, das Apfel bzw. PickUp-Objekt vor. Diese funktionieren grundsätzlich recht ähnlich, haben aber sehr verschiedene Funktionen und Eigenschaften. Sobald ein neuer Apfel erscheinen soll, wird mit einer eins zu zehn Chance festgelegt, ob der Apfel eine goldene Farbe hat. Für die Umsetzung dieser prozentualen Chance haben wir eine Zufallszahl von eins bis zehn erstellt und abgefragt, ob es die Zahl fünf geworden ist. Dann werden Zufällige Koordinaten für die X-, und Y-Achse des Apfels mit einem Wertebereich von 0 bis 15 generiert, da das Spielfeld genau 16 mal 16 Felder gross ist. In der Draw-Klasse, in der das ganze Design des Spiels geregelt ist, wird ein `Point`-Objekt (Dietrich Boles, 2009) dieses Apfels erstellt, mit einer X und Y Koordinate, um genau an dieser Stelle das Feld rot einzufärben. Dann wird durch die Schwänze der Schlange iteriert, und falls die Koordinaten eines Schwanzes mit denen des Apfels übereinstimmen wird der Apfel zurückgesetzt. Somit soll verhindert werden, dass ein Apfel in einem Schwanz erstellt wird. Danach werden ebenfalls noch die Koordinaten des Hindernis- und Bonus-Objekts abgeglichen und bei Übereinstimmung der Apfel erneut zurückgesetzt.

```
public void reset() {
    // Golden PickUp
    isGolden = false;
    int randomNum = ThreadLocalRandom.current().nextInt(1, 10 + 1); // Zufallszahl von 1-10
    if (randomNum == 5) {
        isGolden = true;
    }
    // Normal Pickup
    this.setX(ThreadLocalRandom.current().nextInt(0,15));
    this.setY(ThreadLocalRandom.current().nextInt(0,15));
    // Hier soll verhindert werden, dass ein Apfel in einem Tail, obstacle oder bonus spawnt
    for(int i = 0; i < Snake.tails.size(); i++) { // Iterieren durch Tails
        if(x == Snake.tails.get(i).getX() && y == Snake.tails.get(i).getY()) {
            this.reset();
        }
        if(x == Snake.obstacle.getX() && y == Snake.obstacle.getY()) {
            this.reset();
        }
        if(x == Snake.bonus.getX() && y == Snake.bonus.getY()) {
            this.reset();
        }
    }
}
```

Abbildung 13: [Code] PickUp-Objekt

4.3.5 Snake: Add und Remove Tail

In der Snake-Klasse haben wir die zwei Methoden `add`-, und `removeTail` erstellt, um jederzeit Tails (Schwänze) von der Schlange entfernen, oder hinzufügen zu können. Bei der `addTail`-Methode wird, falls noch kein Tail existiert, ein neues Tail an der Stelle wo der Kopf der Schlange war zur der tails-ArrayList hinzugefügt. Falls bereits Tails existieren wird jeweils ein neues an der Position des letzten Tails hinzugefügt. Da im Hard-Mode die Hindernisse bei Kollision direkt tödlich sind, würden sie nie zurückgesetzt werden, also haben wir manuell eine Funktion eingefügt, die, sobald die Anzahl Tails durch fünf teilbar ist, die Hindernisse zurücksetzt. Bei der `removeTail`-Methode wird eine Zahl übergeben, wie viele Tails entfernt werden sollen. Dann wird durch diese Zahl iteriert und jedes Mal das letzte Tail von der Schlange

```
public static void addTail() { // Fügt ein neues Tail hinzu
    if(tails.size() < 1) { // Fügt, wenn es noch kein Tail hat, ein neues an der alten Position des Kopfes ein.
        tails.add(new Tail(head.getX(), head.getY()));
    } else { // Fügt ein neues Tail an der alten Position des letzten Tails ein
        tails.add(new Tail(tails.get(tails.size()-1).x, tails.get(tails.size()-1).y));
    }
    if (tails.size() % 5 == 0) { // Immer wenn die Punktzahl durch fünf Teilbar ist werden Hindernisse zurückgesetzt.
        obstacle.reset();
    }
}

public static void removeTail(int c) { // Entfernt eine bestimmte Anzahl Tails von der Schlange
    for (int i = 0; i < c; i++) {
        if(tails.size() > 0) {
            tails.remove(tails.size() - 1);
        }
    }
}
```

Abbildung 14: [Code] Snake: Add & Remove Tail

entfernt.

4.3.6 Tastendruck Handling

Sobald eine Taste gedrückt wird, wird eine Switch-Case-Operation ausgelöst, für die der `KeyCode` der gedrückten Taste ausgelesen wird. In der unteren Abbildung sehen Sie das KeyHandling für die W, A, S und D Taste. Wenn zum Beispiel die «W» Taste gedrückt wird, wird die Richtung der Schlange, die in der «Dir» Klasse, die als `Enum` auf vier Optionen beschränkt ist, entsprechend nach oben gesetzt.

```
@Override
public void keyPressed(KeyEvent e) { // Bewegungen der Schlange in bestimmte Richtung
    switch (e.getKeyCode()) { // KeyCode der gedrückten Taste wird verwendet

    // WASD-Tasten
    case KeyEvent.VK_W:
        if(!(Snake.head.getDir() == Dir.DOWN) && !Snake.waitToMove) { // verhindert Probleme durch schnelles Tastenhämmern (hoch-runter)
            Snake.head.setDir(Dir.UP);
            Snake.waitToMove = true;
        }
        break;
    case KeyEvent.VK_A:
        if(!(Snake.head.getDir() == Dir.RIGHT) && !Snake.waitToMove) {
            Snake.head.setDir(Dir.LEFT);
            Snake.waitToMove = true;
        }
        break;
    case KeyEvent.VK_S:
        if(!(Snake.head.getDir() == Dir.UP) && !Snake.waitToMove) {
            Snake.head.setDir(Dir.DOWN);
            Snake.waitToMove = true;
        }
        break;
    case KeyEvent.VK_D:
        if(!(Snake.head.getDir() == Dir.LEFT) && !Snake.waitToMove) {
            Snake.head.setDir(Dir.RIGHT);
            Snake.waitToMove = true;
```

Abbildung 15: [Code] KeyHandler WASD

4.4 Testprotokoll

4.4.1 Testfälle

Nr.	Aktion/Eingabe	Erwartung/Ausgabe	Effektives Resultat	Erfüllt J/N	Massnahmen
1	Eingeben eines Usernames und Auf OK drücken	Name wird im Dokument currentUser.txt gespeichert	Name wird im Dokument currentUser.txt gespeichert	J	
2	Eingeben eines Anderen Usernames bei einem neuen Versuch	Name wird im Dokument currentUser.txt gespeichert und ersetzt den Alten	Name wird im Dokument currentUser.txt gespeichert und ersetzt den Alten	J	
3	Keinen Username eingeben und auf OK klicken	Fehlermeldung wird ausgegeben	Es wird mit einem leeren Username weitergemacht	N	Fehlermeldung erstellen
4	Rekord brechen	Alter Rekord wird durch den Neuen Ersetzt	Alter Rekord wird durch den Neuen ersetzt	J	
5	Rekord mit anderer Schwierigkeit brechen	Schwierigkeit wird aktualisiert	Schwierigkeit wird aktualisiert	J	
6	Spiel mit der Leertaste pausieren	Spiel wird pausiert und Pause-Screen wird aufgerufen	Spiel wird pausiert und Pause-Screen wird aufgerufen	J	
7	Spiel mit der Escape-Taste pausieren	Spiel wird pausiert und Pause-Screen wird aufgerufen	Spiel wird pausiert und Pause-Screen wird aufgerufen	J	
8	Handbuch mit "m" im Spiel öffnen	Spiel wird pausiert und Handbuch wird geöffnet	Spiel wird pausiert und Handbuch wird geöffnet	J	
9	Schwierigkeitsgrad auswählen	Der ausgewählte Schwierigkeitsgrad ist nicht mehr anwählbar	Der ausgewählte Schwierigkeitsgrad ist nicht mehr anwählbar	J	
10	gleiche Punktzahl wie der Rekord erreichen	Rekord wird bleibt und wird nicht aktualisiert	Rekord wird bleibt und wird nicht aktualisiert	J	
11	In Wand fahren in Easy-Mode	Die Schlange kommt auf der anderen Seite des Spielfelds wieder hervor	Die Schlange kommt auf der anderen Seite des Spielfelds wieder hervor	J	
12	In Wand fahren in Hard-Mode	Die Schlange stirbt, es wird der Death-Screen angezeigt	Die Schlange stirbt, es wird der Death-Screen angezeigt	J	
13	Boni aktivieren	Es erscheinen Boni auf dem Spielfeld	Es erscheinen Boni auf dem Spielfeld	J	
14	Bonus "double" wird aktiviert und Schlange sammelt Apfel auf	Es werden zwei statt einem Punkt hinzugefügt	Es werden zwei statt einem Punkt hinzugefügt	J	
15	Bonus "speedup" wird aktiviert	Die Geschwindigkeit der Schlange erhöht sich drastisch	Die Geschwindigkeit der Schlange erhöht sich drastisch	J	

16	Bonus "slow-down" wird aktiviert	Die Geschwindigkeit der Schlange verringert sich drastisch	Die Geschwindigkeit der Schlange verringert sich drastisch	J	
17	Bonus "extraLife" wird aktiviert	Bild "heart.png" wird angezeigt und bei Tod verliert man nur das Leben	Bild "heart.png" wird angezeigt und bei Tod verliert man nur das Leben	J	
18	Schwierigkeitsgrad Easy auswählen	Es erscheinen keine Hindernisse, Man hat von Anfang an ein Extraleben	Es erscheinen keine Hindernisse, Man hat von Anfang an ein Extraleben	J	
19	Schwierigkeitsgrad Normal auswählen	Es erscheinen graue Hindernisse	Es erscheinen graue Hindernisse	J	
20	Schwierigkeitsgrad Hard auswählen	Es erscheinen schwarze Hindernisse, Kollision mit Hindernissen ist tödlich	Es erscheinen schwarze Hindernisse, Kollision mit Hindernissen ist tödlich	J	
21	Bonus "extraLife" wird im Hard-Mode aktiviert, In Wand fahren	Die Schlange kommt auf der anderen Seite des Spielfelds wieder hervor	Die Schlange kommt auf der anderen Seite des Spielfelds wieder hervor	J	

4.4.2 Testbericht

Die Tests sind fast vollständig ohne Fehler durchgelaufen, dies liegt daran, dass wir während der Programmierung schon sobald wir einen Fehler gefunden hatten, diesen gleich darauf verbesserten. Deshalb ist bei unseren Github-Commits oft der Punkt «Div. Bugfixes» für Diverse Bugfixes aufgelistet, die wir nebenbei noch korrigiert haben. Ein Fehler, den wir unbedingt noch beheben wollen, ist die Reaktion auf die Eingabe eines leeren Benutzernamens. Bis jetzt kann man das Spiel normal starten, doch sobald es zur Festlegung eines neuen Rekords kommt verursacht ein leerer Benutzername fatale Fehler. Wir werden die Erstellung einer Fehlermeldung, bei leerem Benutzernamen ganz nach oben auf unsere Prioritätenliste setzen, da es keinen grossen Programmieraufwand mit sich bringt, aber dennoch sehr wichtig ist.

5 Reflexion

5.1 Soll / Ist Ressourcen Vergleich

Wir haben die Software-Ressourcen, die eingeplant waren, alle sehr gut nutzen können, besonders die Verwendung von Git hat uns sehr überzeugt und wir würden es ganz bestimmt wieder so machen bei einem ähnlichen Projekt. Einen Punkt haben wir noch vergessen aufzulisten, das wäre Microsoft Excel, dieses Programm haben wir nämlich für die Erstellung unserer Testfälle verwendet, da sich sehr gut einfache Tabellen erstellen lassen, die man leicht in die Word-Dokumentation einbinden kann. Wir haben uns nicht genau aufgeschrieben, wieviel Zeit wir für das Projekt aufgewendet haben, aber die eingeplanten 60 Stunden haben wir ganz bestimmt erreicht und wahrscheinlich sogar übertroffen. Wir haben sehr viel Zeit, (vor allem in den Weihnachtsferien), für die Programmierung eingesetzt.

5.2 Probleme

Beim Programmieren sind wir immer wieder auf Konflikte oder Fehler gestossen. Im Folgenden werden wir auf ein Paar davon eingehen und erklären, wie wir sie gelöst haben.

Der Death-Screen sollte die zwei Optionen «Play Again» und «Quit» anbieten. Das Problem war allerdings, das der «Play Again» Button jeweils einen neuen `JFrame` des Spiels über den bereits vorhandenen gelegt hat, also haben wir es so umprogrammiert, dass zuerst das komplette Spiel geschlossen wird, und danach die gleichen Operationen wie beim Start des Spiels manuell aufgerufen werden. So werden alle Daten zurückgesetzt und das Problem mit mehreren `JFrames` ist behoben. Wir haben in den einigen Dialogen einen «Quit» Button eingebaut, der neben dem Dialog auch das ganze Spiel schliesst und dachten erst gar nicht daran, dass Benutzer auch versuchen könnten, diese Fenster mit dem normalen X-Button oben rechts zu schliessen. Wenn man dies tat, beendete sich nämlich nur der Dialog und nicht das Spielfeld und da dieses noch disabled war konnte es nur durch den Task-Manager geschlossen werden. Also haben wir einen `WindowListener` für das Drücken des X-Buttons eingebaut, der auch das Spiel selbst beendet.

Uns ist erst mit dem Einbau der `JProgressBar` zur Anzeige des aktiven Bonus aufgefallen, dass wenn die Schlange stirbt und das Spiel mit dem «Play Again» Button neugestartet wird, der aktive Bonus immer noch anhält und für die verbleibende Zeit noch aktiv ist. Dies war schnell korrigiert, indem bei der Aktivierung des Death-Screens die verbleibende Bonuszeit zurückgesetzt wird. Wir wollten unbedingt vier Boni in das Spiel einbauen, waren aber lange Zeit sehr unschlüssig, was der vierte Bonus für eine Funktion haben sollte. Erst sollte es ein sogenannter «Swapper» werden, der die Richtung der Schlange umkehrt. Dies fanden wir aber zu langweilig und so kamen wir auf die Idee, einen Teleporter-Bonus einzubauen, der Die Schlange an einen zufälligen Ort auf dem Spielfeld teleportierte. Dies stellte sich aber für den Spieler als recht frustrierend heraus, da man oft direkt vor ein Hindernis, eine Wand, oder den eigenen Schwanz teleportiert wurde, und so direkt starb. Endlich kam uns die Idee, wir könnten einen Punktever-doppler erstellen, der, solange er aktiv ist, alle aufgesammelten Punkte verdoppelt. Diese Idee gefiel uns beiden gut also setzten wir sie um.

5.3 Schlusswort / Fazit

Wir sind nun am Ende unseres Projekts angekommen. Wir sind beide der Meinung, dass das Projekt uns viel Spass gemacht und uns in unseren Programmierfähigkeiten ein gutes Stück weitergebracht hat. Die Programmierung des Spiels brachte uns so manchen Frust-, aber auch viele Glücksmomente. Wir haben sämtliche Muss-, Soll- und Kann Kriterien erfüllt und haben weit darüber hinaus Features und zusätzliche Fenster mit neuen Funktionen eingebaut. Wir sind stolz auf das fertige Produkt und das neue Wissen, den wir aus der Erstellung ziehen konnten. Wie wir es uns fest vorgenommen hatten, unterscheidet sich unser Spiel sehr stark von dem, dass in unserer Hilfs-Videoreihe gezeigt wurde. Jede Klasse hat seit Beginn der

Programmierung zahlreiche Generalerneuerungen und Umgestaltungen durchlaufen. Wir hatten immer wieder neue Ideen, wie wir das Spiel erneuern könnten und haben diese auch mit Freude umgesetzt. Besonders die Umsetzung der Boni war sehr spannend, wenn auch recht kompliziert. Wir haben das Spiel auch anderen Personen vorgeführt und sie es spielen lassen, um ihren Input und ihre Rückmeldungen einbinden zu können. Die Dokumentation hat uns nicht so viel Spass gemacht wie das Spiel zu programmieren, aber dank der Energie, die wir in das Projekt gesteckt haben, viel uns das Schreiben leicht und wir kamen schnell voran. Unsere Teamarbeit hat meistens gut funktioniert, da wir die gleichen Stundenpläne haben konnte wir auch unsere Arbeitszeiten gut aufeinander abstimmen und so stets im Zeitplan bleiben.

6 Anhang

6.1 Abbildungsverzeichnis

6.2 Quellenverzeichnis

6.2.1 GitHub-Repository

Snakeology: https://github.com/Manuel-Schmid/Snakeology

6.2.2 Bilder Titelblatt

Oberes Bild: https://blockchain-hero.com/neo-ethereum-welt-digital/amp/

Unteres Bild: https://unsplash.com/photos/70Rir5vB96U

6.2.3 Bilder des Spiels

Herz Bild: https://www.pngitem.com/middle/JRoioT_snake-snake-icon-free-hd-png-download/

Snake Bild: https://iconarchive.com/show/free-valentine-heart-icons-by-designbolts/Heart-icon.html

6.2.4 Zitationsverzeichnis

codeflow.site. (15. 3 2020). *2021 codeflow.site*. Von codeflow.site: https://www.codeflow.site/de/article/java-wait-and-sleep abgerufen

Dietrich Boles, U. O. (08. 09 2009). *2020 programmierkurs-java.de*. Von programmierkurs-java.de: http://www.programmierkurs-java.de/objekttheater/API/theater/Point.html abgerufen

javabeginners.de. (16. 05 2020). *javabeginners.de 2021*. Von javabeginners.de: https://javabeginners.de/Grundlagen/main.php#:~:text=Eine%20Methode%20main()%20muss,(String%5B%5D%20args)%20besitzen. abgerufen

Petri, B. u. (30. 12 2020). *2021 - Java-Tutorial.org*. Von Java-Tutorial.org: https://www.java-tutorial.org/jprogressbar.html abgerufen

6.2.5 Internetquellen

- https://www.w3schools.com/colors/colors_picker.asp
- https://books.trinket.io/thinkjava/appendix-b.html
- https://stackoverflow.com/questions/144892/how-to-center-a-window-in-java
- https://docs.oracle.com/javase/7/docs/api/java/awt/Color.html
- https://www.geeksforgeeks.org/java-swing-jprogressbar/
- https://examples.javacodegeeks.com/desktop-java/awt/event/window-closing-event-handling/#:~:text=Basically%20all%20you%20have%20to,handle%20a%20window%20closing%20event
- https://www.javatpoint.com/java-joptionpane
- https://stackoverflow.com/questions/17132452/java-check-if-jtextfield-is-empty-or-not
- https://stackoverflow.com/questions/10336293/splitting-filenames-using-system-file-separator-symbol
- https://www.geeksforgeeks.org/split-string-java-examples/
- https://www.educative.io/edpresso/how-to-convert-an-integer-to-a-string-in-java
- https://stackoverflow.com/questions/25378611/how-to-update-jlabel-or-settext-from-another-class-in-java
- https://stackoverflow.com/questions/2380314/how-do-i-set-a-jlabels-background-color
- https://www.codeflow.site/de/article/modulo-java
- https://stackoverflow.com/questions/363681/how-do-i-generate-random-integers-within-a-specific-range-in-java

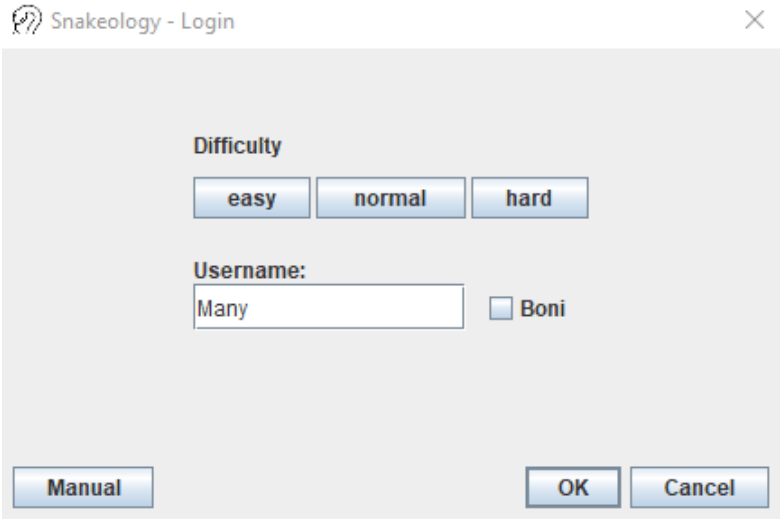
6.2.6 YouTube

- Video 1/4: https://www.youtube.com/watch?v=OZYVfVxB81s
- Video 2/4: https://www.youtube.com/watch?v=EMkrtobp1uA
- Video 3/4: https://www.youtube.com/watch?v=iGYEpF6o0QY
- Video 4/4: https://www.youtube.com/watch?v=UJ3aZ4M5joA&t

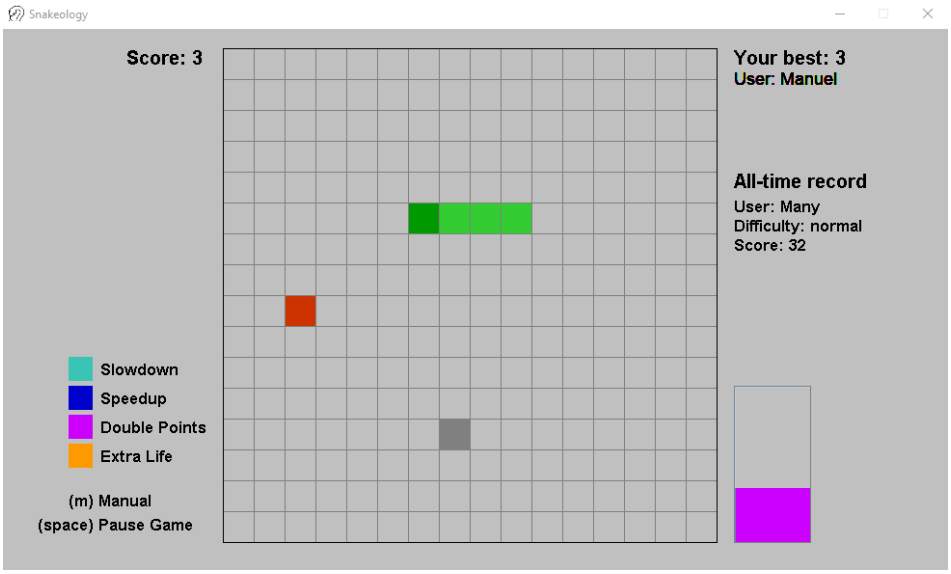
6.3 Arbeitsjournal

Datum	**Ergebnisse & Tätigkeiten**
14.08.2020	Projekteinführung und Gruppenbildung
21.08.2020	Auswahl des Projekts, Grobentwurf erstellt, Pflichtenheft erstellt, Dokumentation Grundgerüst kreiert.
28.08.2020	Vorwort: Brainstorming und Auftrag eingetragen.
11.09.2020	Definitive Aufgabenstellung fertig,
23.10.2020	Grobentwurf verbessert
06.11.2020	Ausbau Dokumentation, Erstellung GitHub Projekt, Verknüpfung mit Git
07.11.2020	Erstellung Grundstruktur und Grid
13.11.2020	Ausarbeitung Grid, Dokumentation ergänzen
14.11.2020	Erstellung weiterer Struktur, KeyHandler, Head und Tail, Push des erreichten Fortschritts
15.11.2020	**Snakeology v1.0:** Bewegung, Kollisionen und Aussehen
18.11.2020	Erstellung Zwischenbericht
20.11.2020	**Zwischenpräsentation Projekt**, Abgabe Zwischenbericht
21.11.2020	Kleine Verbesserungen, Schlange wird schneller
27.11.2020	Erstellung Login-Screen, Username in File speichern
04.12.2020	User wird angezeigt, Liste für Top-Spieler angefangen
11.12.2020	**Snakeology v2.0:** Login, Death, Record usw. vollständig überarbeitet
18.12.2020	Pause-Screen mit KeyHandlern erstellt
22.12.2020	Bugfix: Äpfel können nicht mehr in Tail spawnen
23.12.2020	Golden PickUp Handling erstellt
24.12.2020	Erstellung Schwierigkeitsstufen & Obstacles
25.12.2020	Extra Life, Durch Wall gehen, Div. Bugfixes
26.12.2020	Boni: Slowdown, SpeedUp, Teleporter
27.12.2020	Bonus: Double, Bugfixes, Difficulty abspeichern, neues Speichermanagement
28.12.2020	Legende für Boni, Progress-Bar für Boni, Div. Bugfixes
29.12.2020	**Snakeology v2.1:** Erstellung Anleitung, Designverbesserungen
30.12.2020	Kommentare hinzugefügt, Manual kann auf Login-Screen angezeigt werden
31.12.2020	App-Icon gesetzt, Grosse Teile der Dokumentation abgeschlossen
01.01.2021	Dokumentation beinahe abgeschlossen
02.01.2021	Dokumentation erneut anschauen und verbessern
03.01.2021	Dokumentation vollständig abgeschlossen

6.4 Benutzerhandbuch

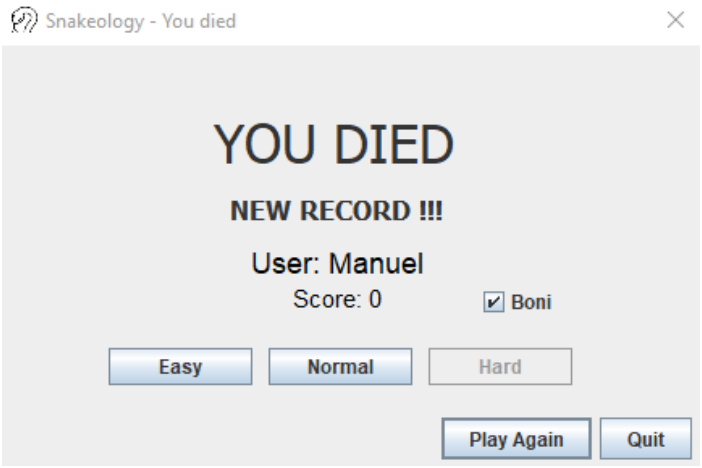


Sobald Sie das Spiel starten sollten Sie das folgende Fenster sehen. Nun können Sie Ihren Gewünschten Schwierigkeitsgrad wählen, dieser beeinflusst die Geschwindigkeit und die Gefahren, die Sie im Spiel erwarten werden. Mit einem Klick auf das kleine Feld, das mit «Boni» betitelt ist, können sie entscheiden, ob Boni mit speziellen Effekten wie zum Beispiel einem Extraleben oder Zeitlupe aktiviert oder deaktiviert sein sollen. Dann geben Sie Ihren gewünschten Benutzernamen in das Feld «Username:» ein. Wir empfehlen Ihnen, zumindest bei der ersten Runde, sich einmal die Anleitung mit den Spielregeln durchzulesen. Diese können Sie mit den Knopf «Manual» aufrufen. Die Anleitung ist, genau wie der Rest des Spiels, in Englisch geschrieben. Falls Sie das Spiel direkt starten möchten können Sie auch während des Spiels noch auf die Anleitung zugreifen. Mit Klick auf Ok geht das Spiel los.



Wenn das Spiel beginnt erwartet Sie ein Fenster, das ähnlich wie das oben abgebildete aussieht, je nachdem, ob sie die Boni aktiviert haben, werden mehr oder weniger Informationen am Rand angezeigt. Nun können Sie das Spiel mit den Pfeiltasten, oder WASD steuern. Auf die erweiterten Spielregeln werden wir

hier nicht eingehen, da Sie sich diese jederzeit während des Spiels, durch Öffnen der Anleitung mit der Taste «m», in Ruhe ansehen können. Im Grunde geht es aber darum, dass Sie die grüne Schlange steuern, und versuchen müssen, so viele rote Äpfel wie möglich zu verzehren, wobei der Schwanz der Schlange jedes Mal um ein Feld verlängert wird. Der Kopf der Schlange darf nicht mit dem eigenen Schwanz zusammenstossen, sonst ist das Spiel zu Ende. Oben Links sehen Sie, wie lange ihr Schwanz zurzeit ist und rechts die beste Punktzahl, die sie erreicht haben. Wenn Sie der erste Spieler sind, oder den vorhandenen Rekord brechen, werden die Daten Ihres Spieldurchlaufs als «All-time record» verewigt. Dieser Rekord bleibt auch vorhanden, wenn Sie das Spiel beenden. Wenn Sie während des Spiels eine Verschnaufpause benötigen, können Sie das Spiel mit der Escape-, oder der Leertaste pausieren. Wenn Sie Boni aktiviert und einen Bonus aufgesammelt haben, wird am rechten Rand ein Balken dargestellt, der Ihnen, in der Farbe des ausgesammelten Bonus, anzeigt, wie lange der Bonuseffekt noch aktiv ist.



Falls Ihre Schlange stirbt, wird Ihnen das oben dargestellte Fenster angezeigt, in dem Ihnen nochmals Ihren Benutzernamen und Ihre erreichte Punktzahl angezeigt wird. Falls Sie nochmal einen Versuch wagen wollen, können Sie Ihre Einstellungen über Schwierigkeit und Boni beliebig verändern, oder auch unverändert lassen. Sobald Sie auf «Play Again» drücken, wird das Spiel neugestartet. Wenn Sie schliesslich doch genug haben, können Sie das Spiel mit einem Klick auf «Quit» beenden. Ihr Rekord bleibt dabei trotzdem gespeichert.

6.5 Vollständiger Code

Nachfolgend haben wir den vollständigen Code aus unseren 18 Klassen eingefügt.

```
/**
 * @author Manuel
 */

package actions;

import java.io.IOException;

import javax.swing.JDialog;

import gui.LoginScreen;

public class Main {

	public static void main(String[] args) throws IOException {
	   // Beim Start des Programms wird der LoginScreen angezeigt.
	   LoginScreen loginS = new LoginScreen();
	   loginS.setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
	   loginS.setVisible(true);

	}

}
```

```
/**
 * @author Manuel
 */

package clocks;

import java.io.IOException;


import actions.Collision;
import game.Snake;
import gui.Draw;

public class GameClock extends Thread{ // Diese Klasse verwaltet die
        Funktionen des Spiels und arbeitet in Ticks

        public static boolean running = true; // Läuft das Spiel?
        public static String difficulty; // Schwierigkeitsstufe
        public static boolean extraLife = false; // Ist ein extraLife
        aktiv?
        public static boolean boniOn = false; // Sind Boni aktiviert?
        public static int bonusTimer = 40; // Timer wie lange die Boni
        andauern

        public GameClock() {}

        public void run() {
           if (difficulty == "easy") {     extraLife = true; } else {
        extraLife = false; } // Easy-Mode hat ein standardmässiges extra
        Leben
           while(running) { // Auch wenn es nicht funktioniert stürtzt
        es nicht ab, es wird nur ein StackTrace geprintet
                 try {
```

```
            if (bonusTimer == 0) { // Bonus wird neu gesetzt nachdem er abgelaufen ist.
                Snake.bonus.reset();
                Collision.activeBonus = "";
                if (difficulty == "normal" || difficulty == "hard") {extraLife = false; }
                bonusTimer = 40;
            }
            if (extraLife == true) { Draw.lblHeart.setVisible(true); } else { Draw.lblHeart.setVisible(false); } // Sichtbarkeit des Herzbildes
            if (Collision.activeBonus != "") { Draw.b.setVisible(true); } else { Draw.b.setVisible(false); } // Sichtbarkeit der ProgressBar
            // Tick Management, BonusTimer wird verringert
            if (Collision.activeBonus == "slowdown" && bonusTimer > 0) { // Geschwindigkeit mit Slowdown-Bonus
                sleep(230);
                bonusTimer--;
                Draw.b.setValue((int) (4 * bonusTimer)); // ProgressBar-Wert wird angepasst, wieviel Prozent der Zeit in der der Bonus anhält verbleibt noch?
            } else if (Collision.activeBonus == "speedup" && bonusTimer > 0){
                sleep(140);
                bonusTimer--;
                Draw.b.setValue((int) (1.666666 * bonusTimer)); // ProgressBar-Wert wird angepasst, wieviel Prozent der Zeit in der der Bonus anhält verbleibt noch?
            } else if (difficulty == "easy") { // Geschwingkeit im Easy-Mode
                sleep(215); // Schlange bleibt immer gleichschnell
            } else if (difficulty == "normal") { // Geschwingkeit im Normal-Mode
```

```
                    if (Snake.score < 10) { // Schlange wird mit
höherer Punktzahl schneller
                        sleep(185);
                    } else if (Snake.score < 15) {
                        sleep(175);
                    } else if (Snake.score < 20) {
                        sleep(165);
                    } else {
                        sleep(155);
                    }
                } else if (difficulty == "hard") { // Geschwingkeit
im Hard-Mode
                    if (Snake.score < 10) { // Schlange wird mit
höherer Punktzahl schneller
                        sleep(170);
                    } else if (Snake.score < 15) {
                        sleep(155);
                    } else if (Snake.score < 20) {
                        sleep(140);
                    } else if (Snake.score < 25) {
                        sleep(125);
                    } else {
                        sleep(110);
                    }
                }

                if (Collision.activeBonus == "extraLife" || Colli-
sion.activeBonus == "double") {
                    bonusTimer--;
                    Draw.b.setValue((int) (2.5 * bonusTimer));
// ProgressBar-Wert wird angepasst, wieviel Prozent der Zeit in
der der Bonus anhält verbleibt noch?
                }
```

```
                Snake.move(); // Schlange bewegt sich (letzte ge-
        drückte Richtung)
                Snake.waitToMove = false;
                // Alle Kollisionen werden aufgerufen
                Collision.collidePickUp();
                Collision.collideSelf();
                Collision.collideWall();
                if (difficulty == "hard" || GameClock.difficulty
        == "normal" ) { // Im Easy-Mode gibt es keine Hindernisse (Ob-
        stacles)
                     Collision.collideNormalObstacle();
                     Collision.collideBlackObstacle();
                }
                if (boniOn == true) { // Muss nur aufgerufen
        werden, wenn Boni aktiviert sind
                     Collision.collideBonus();
                }
             } catch (InterruptedException e) {
                e.printStackTrace();
             } catch (IOException e) {
                // TODO Auto-generated catch block
                e.printStackTrace();
             }
          }
        }
}
```

```
/**
 * @author Manuel
 */

package actions;

import java.awt.Color;
import java.io.IOException;
import javax.swing.JDialog;

import clocks.GameClock;
import game.Bonus;
import game.Obstacle;
import game.PickUp;
import game.Snake;
import gui.DeathScreen;
import gui.Draw;

public class Collision { // Diese Klasse ist für das Handling aller
        Arten von Kollisionen zuständig

        public static String activeBonus = ""; //Variable für den derzeit
        aktiven Bonus

        public static void collideSelf() throws IOException { // Kolli-
        sion mit einem Tail der Schlange
           for(int i = 0; i < Snake.tails.size(); i++) { // Iterieren
        durch Tails
                if(Snake.head.getX() == Snake.tails.get(i).getX() &&
        Snake.head.getY()      ==      Snake.tails.get(i).getY()      &&
        !Snake.tails.get(i).isWait()) {
                     if (GameClock.extraLife == true) {
                          GameClock.extraLife = false;
                     } else {
```

```
                    Snake.head.setX(7);
                    Snake.head.setY(7);
                    Snake.tails.clear();
                    // Spiel stoppen & DeathScreen anzeigen
                    GameClock.running = false;
                    DeathScreen deathS = new DeathScreen();
                    deathS.setDefaultCloseOperation(JDialog.DIS-
POSE_ON_CLOSE);
                    deathS.setVisible(true);
                }
            }
        }
}

public static void collideWall() throws IOException { // Kolli-
sion mit einer der Wände (Rahmen des Spielfelds)
    if (Snake.head.getX()<0 || Snake.head.getX() > 15 ||
Snake.head.getY()<0 || Snake.head.getY() > 15 ) {// Wenn der Head
oben, unten, links oder rechts in die Border fährt
        if (GameClock.difficulty == "hard") { // Beim Hardmode
stirbt man direkt
            if (GameClock.extraLife == true) { // Extraleben
abziehen und auf anderer Seite spawnen
                    GameClock.extraLife = false;
                    // Schlange kommt auf anderer Seite wieder
raus
                    if (Snake.head.getX()<0) {
                        Snake.head.setX(15);
                    } else if (Snake.head.getX()>15){
                        Snake.head.setX(0);
                    } else if (Snake.head.getY()<0){
                        Snake.head.setY(15);
                    } else if (Snake.head.getY()>15){
                        Snake.head.setY(0);
```

```
                    }
                } else {
                    Snake.head.setX(7);
                    Snake.head.setY(7);
                    Snake.tails.clear();
                    // Spiel stoppen & DeathScreen anzeigen
                    GameClock.running = false;
                    DeathScreen deathS = new DeathScreen();
                    deathS.setDefaultCloseOperation(JDialog.DIS-
POSE_ON_CLOSE);
                    deathS.setVisible(true);
                }
            // Schlange kommt auf anderer Seite wieder raus
            } else if (Snake.head.getX()<0){
                Snake.head.setX(15);
            } else if (Snake.head.getX()>15){
                Snake.head.setX(0);
            } else if (Snake.head.getY()<0){
                Snake.head.setY(15);
            } else if (Snake.head.getY()>15){
                Snake.head.setY(0);
            }
        }
    }

    public static void collidePickUp() { // Kollision mit einem Apfel
        if(Snake.head.getX() == Snake.pickup.getX() &&
Snake.head.getY() == Snake.pickup.getY()) {
            // Score erhöhen
            if (PickUp.isGolden) { // Golder Apfel && Double-Bonus
Handling
                if (Collision.activeBonus == "double") {
                    Snake.score += 6;
                    for (int i = 0; i < 6; i++) {
```

```
                        Snake.addTail();
                    }
                } else {
                    Snake.score += 3;
                    for (int i = 0; i < 3; i++) {
                        Snake.addTail();
                    }
                }
            } else {
                if (Collision.activeBonus == "double") { // Dou-
ble-Bonus Handling
                    Snake.score += 2;
                    Snake.addTail();
                    Snake.addTail();
                } else {
                    Snake.score += 1;
                    Snake.addTail();
                }
            }
            if (Snake.score > Snake.bestscore) { // Bestscore aktu-
alisieren
                Snake.bestscore = Snake.score;
            }
            Snake.pickup.reset(); // neuer Apfel erscheint
        }
    }

    public static void collideNormalObstacle() { // Kollision mit
Hindernis im Normal-Mode
        if(Obstacle.isBlack == false && Snake.head.getX() ==
Snake.obstacle.getX() && Snake.head.getY() ==
Snake.obstacle.getY()) {
            // Score verringern
            if (Snake.score > 1) {
```

```
                Snake.score -= 1;
            } else {
                Snake.score = 0;
            }
            Snake.obstacle.reset(); // Neues Hindernis erscheint
            Snake.removeTail(1); // Schwanz abziehen
        }
    }

    public static void collideBlackObstacle() throws IOException {
    // Kollision mit Hindernis im Hard-Mode
        if(Obstacle.isBlack == true && Snake.head.getX() ==
    Snake.obstacle.getX() && Snake.head.getY() ==
    Snake.obstacle.getY()) {
            if (GameClock.extraLife == true) { // ExtraLife Handling
                GameClock.extraLife = false;
                Snake.obstacle.reset(); // Neues Hindernis er-
    scheint
            } else {
                // Schlange stirbt
                Snake.head.setX(7);
                Snake.head.setY(7);
                Snake.tails.clear();
                GameClock.running = false;
                DeathScreen deathS = new DeathScreen();
                deathS.setDefaultCloseOperation(JDialog.DIS-
    POSE_ON_CLOSE);
                deathS.setVisible(true);
            }
        }
    }

    public static void collideBonus() { // Kollision mit einem Bonus
    (wird nur wenn Boni aktiviert sind von der GameClock aufgerufen)
```

```
            if(Snake.head.getX() == Snake.bonus.getX() &&
        Snake.head.getY() == Snake.bonus.getY()) {
                Snake.bonus.setX(-5);
                Snake.bonus.setY(-5);
                GameClock.bonusTimer = 40; // Timer zurücksetzen
                activeBonus = Bonus.bonus; // bonus wird aktiv
                Draw.b.setValue(100); // ProgressBar vom Bonus wird auf-
        gefüllt
                if (activeBonus == "slowdown") { // Handling der ein-
        zelnen Boni und ProgressBar wird entsprechend dem Bonus einge-
        färbt
                      GameClock.bonusTimer = 25;
                      Draw.b.setForeground(new Color(57, 196, 182));
                }
                else if (activeBonus == "extraLife") {
                      GameClock.extraLife = true;
                      Draw.b.setForeground(new Color(255, 153, 0));
                } else if (activeBonus == "speedup") {
                      GameClock.bonusTimer = 60;
                      Draw.b.setForeground(new Color(0, 0, 204));
                } else if (activeBonus == "double") {
                      Draw.b.setForeground(new Color(204, 0, 255));
                }
            }
        }
}
```

```
/**
 * @author Manuel
 */

package actions;

import java.awt.event.KeyEvent;
import java.awt.event.KeyListener;

import javax.swing.JDialog;

import clocks.GameClock;
import game.Dir;
import game.Snake;
import gui.ManualScreenBasics;
import gui.PauseScreen;

public class KeyHandler implements KeyListener{ // Diese Klasse soll das
        Handling der gedrückten Tasten während des Spiels verwalten.

        @Override
        public void keyTyped(KeyEvent e) {}

        @Override
        public void keyPressed(KeyEvent e) { // Bewegungen der Schlange
        in bestimmte Richtung
           switch (e.getKeyCode()) { // KeyCode der gedrückten Taste
        wird verwendet

           // WASD-Tasten
           case KeyEvent.VK_W:
                if(!(Snake.head.getDir() == Dir.DOWN) && !Snake.waitTo-
        Move) { // verhindert Probleme durch schnelles Tastenhämmern
        (hoch-runter)
```

```
                Snake.head.setDir(Dir.UP);
                Snake.waitToMove = true;
            }
            break;
        case KeyEvent.VK_A:
            if(!(Snake.head.getDir() == Dir.RIGHT) && !Snake.wait-
ToMove) {
                Snake.head.setDir(Dir.LEFT);
                Snake.waitToMove = true;
            }
            break;
        case KeyEvent.VK_S:
            if(!(Snake.head.getDir() == Dir.UP) && !Snake.wait-
ToMove) {
                Snake.head.setDir(Dir.DOWN);
                Snake.waitToMove = true;
            }
            break;
        case KeyEvent.VK_D:
            if(!(Snake.head.getDir() == Dir.LEFT) && !Snake.wait-
ToMove) {
                Snake.head.setDir(Dir.RIGHT);
                Snake.waitToMove = true;
            }
            break;

        // Pfeiltasten
        case KeyEvent.VK_UP:
            if(!(Snake.head.getDir() == Dir.DOWN) && !Snake.wait-
ToMove) {
                Snake.head.setDir(Dir.UP);
                Snake.waitToMove = true;
            }
            break;
```

```
   case KeyEvent.VK_LEFT:
        if(!(Snake.head.getDir() == Dir.RIGHT) && !Snake.wait-
ToMove) {
             Snake.head.setDir(Dir.LEFT);
             Snake.waitToMove = true;
        }
        break;
   case KeyEvent.VK_DOWN:
        if(!(Snake.head.getDir() == Dir.UP) && !Snake.wait-
ToMove) {
             Snake.head.setDir(Dir.DOWN);
             Snake.waitToMove = true;
        }
        break;
   case KeyEvent.VK_RIGHT:
        if(!(Snake.head.getDir() == Dir.LEFT) && !Snake.wait-
ToMove) {
             Snake.head.setDir(Dir.RIGHT);
             Snake.waitToMove = true;
        }
        break;

   // PauseScreen (wird durch "Escape" oder die Leertaste auf-
gelöst
   case KeyEvent.VK_ESCAPE:
        if(GameClock.running == true) {
             GameClock.running = false;
             PauseScreen pauseS = new PauseScreen();
             pauseS.setDefaultCloseOperation(JDialog.DIS-
POSE_ON_CLOSE);
             pauseS.setVisible(true);
        }
        break;
   case KeyEvent.VK_SPACE:
```

```
                if(GameClock.running == true) {
                     GameClock.running = false;
                     PauseScreen pauseS = new PauseScreen();
                     pauseS.setDefaultCloseOperation(JDialog.DIS-
        POSE_ON_CLOSE);
                     pauseS.setVisible(true);
                }
                break;

           // ManualScreen (wird durch die "m" Taste ausgelöst
           case KeyEvent.VK_M:
                if(GameClock.running == true) {
                     GameClock.running = false;
                     ManualScreenBasics ManualBS = new ManualScreenBa-
        sics();
                     ManualBS.setDefaultCloseOperation(JDialog.DIS-
        POSE_ON_CLOSE);
                     ManualBS.setVisible(true);
                }
                break;
           }
        }

        @Override
        public void keyReleased(KeyEvent e) {}

}
```

```
/**
 * @author Sven
 */

package game;

public enum Dir { // Keys setzen, die das Enum annehmen kann (Directions)
        UP, LEFT, RIGHT, DOWN
}
/**
 * @author Manuel
 */

package game;

import java.util.concurrent.ThreadLocalRandom;

public class Bonus { // Diese Klasse erstellt ein Bonus-Objekt
        int x;
        int y;
        public static String bonus = ""; // Hier wird der Bonus gespei-
        chert (slowdown, extraLife, speedup oder double)

        public Bonus() {
           this.setX(ThreadLocalRandom.current().nextInt(0,15));     //
        Random Koordinaten zwischen 0 und 15 für Boni für X und Y Achse
           this.setY(ThreadLocalRandom.current().nextInt(0,15));
           // Hier soll verhindert werden, dass ein Bonus in einem Tail,
        PickUp oder Hindernis spawnt
           for(int i = 0; i < Snake.tails.size(); i++) { // Iterieren
        durch Tails
                 if(x    ==    Snake.tails.get(i).getX()    &&    y    ==
        Snake.tails.get(i).getY()) {
                    this.reset();
```

```
            }
            if(x == Snake.pickup.getX() && y == Snake.pickup.getY())
{
                  this.reset();
            }
            if(x  ==  Snake.obstacle.getX()  &&  y  ==  Snake.obsta-
cle.getY()) {
                  this.reset();
            }
      }
      int randomNum = ThreadLocalRandom.current().nextInt(1, 12 +
1); // Zufallszahl von 1-10
      if (randomNum <= 3) { // Je nachdem in welchem Bereich die
Zufallszahl liegt wird ein Bonus festgelegt
            bonus = "slowdown";
      } else if (randomNum <= 6) {
            bonus = "extraLife";
      } else if (randomNum <= 9) {
            bonus = "speedup";
      } else if (randomNum <= 12) {
            bonus = "double";
      }
}

public void reset() {
      this.setX(ThreadLocalRandom.current().nextInt(0,15));     //
Random Koordinaten zwischen 0 und 15 für Boni für X und Y Achse
      this.setY(ThreadLocalRandom.current().nextInt(0,15));
      // Hier soll verhindert werden, dass ein Bonus in einem Tail,
PickUp oder Hindernis spawnt
      for(int i = 0; i < Snake.tails.size(); i++) { // Iterieren
durch Tails
            if(x    ==    Snake.tails.get(i).getX()    &&    y    ==
Snake.tails.get(i).getY()) {
                  this.reset();
```

```
            }
            if(x == Snake.pickup.getX() && y == Snake.pickup.getY())
{
                  this.reset();
            }
            if(x  ==  Snake.obstacle.getX()  &&  y  ==  Snake.obsta-
cle.getY()) {
                  this.reset();
            }
      }
      int randomNum = ThreadLocalRandom.current().nextInt(1, 12 +
1); // Zufallszahl von 1-10
      if (randomNum <= 3) { // Je nachdem in welchem Bereich die
Zufallszahl liegt wird ein Bonus festgelegt
            bonus = "slowdown";
      } else if (randomNum <= 6) {
            bonus = "extraLife";
      } else if (randomNum <= 9) {
            bonus = "speedup";
      } else if (randomNum <= 12) {
            bonus = "double";
      }
}

public int getX() {
      return x;
}

public void setX(int x) {
      this.x = x;
}

public int getY() {
      return y;
```

```
        }

        public void setY(int y) {
           this.y = y;
        }
}
```

```
/**
 * @author Sven
 */

package game;

public class Tail { // erstellt ein neues Tail-Objekt

        // Koordinaten
        int x;
        int y;
        boolean wait = true; // Warten auf Bewegung

        public Tail(int x, int y) {
           this.setX(x);
           this.setY(y);
        }

        public int getX() {
           return x;
        }

        public void setX(int x) {
           this.x = x;
        }

        public int getY() {
           return y;
        }

        public void setY(int y) {
           this.y = y;
        }
```

```
        public boolean isWait() {
           return wait;
        }

        public void setWait(boolean wait) {
           this.wait = wait;
        }


}

/**
 * @author Sven
 */

package game;

public class Head {
        Dir dir = Dir.RIGHT; // Standardmässig bewegt sich die Schlange
        nach rechts
        int x;
        int y;

        public Head(int x, int y) {
           this.setX(x);
           this.setY(y);
        }


        public Dir getDir() {
           return dir;
        }

        public void setDir(Dir dir) {
```

```
            this.dir = dir;
        }

        public int getX() {
            return x;
        }

        public void setX(int x) {
            this.x = x;
        }

        public int getY() {
            return y;
        }

        public void setY(int y) {
            this.y = y;
        }

}
```

```
/**
 * @author Manuel & Sven
 */

package game;

import java.awt.Point;
import java.util.ArrayList;

import gui.Gui;

public class Snake {

	public static int score = 0; // Aktueller Punktestand

	public static int bestscore = 0; // Beste erreichte Punktzahl

	public static boolean waitToMove = false; // Wartezeit zwischen
	Tastendruck und Bewegung

	public static Head head = new Head(7, 7); // Startposition etwa
	in Mitte

	public static ArrayList<Tail> tails = new ArrayList<Tail>(); //
	Liste der Tails (Schwänze) der Schlange

	public static PickUp pickup = new PickUp();    // erster Apfel
	wird erstellt

	public static Obstacle obstacle = new Obstacle(); // erstes Hin-
	derniss wird erstellt

	public static Bonus bonus = new Bonus(); // erster Bonus wird
	erstellt
```

```
public static void addTail() { // Fügt ein neues Tail hinzu
   if(tails.size() < 1) { // Fügt, wenn es noch kein Tail hat, ein neues an der alten Position des Kopfes ein.
         tails.add(new Tail(head.getX(), head.getY()));
   } else { // Fügt ein neues Tail an der alten Position des letzten Tails ein
         tails.add(new         Tail(tails.get(tails.size()-1).x, tails.get(tails.size()-1).y));
   }
   if (tails.size() % 5 == 0) { // Immer wenn die Punktzahl durch fünf Teilbar ist werden Hindernisse zurückgesetzt.
         obstacle.reset();
   }
}

public static void removeTail(int c) { // Entfernt eine bestimmte Anzahl Tails von der Schlange
   for (int i = 0; i < c; i++) {
         if(tails.size() > 0) {
               tails.remove(tails.size() - 1);
         }
   }
}

public static void move() {
   // Move Tails
   if(tails.size() >= 2) {
         for(int i = tails.size()-1; i>=1; i--) {
               if(tails.get(i).isWait()) { // Wenn Sie sich noch nicht bewegen kann
                     tails.get(i).setWait(false);
               } else { // Wenn Sie sich bewegen kann
                     tails.get(i).setX(tails.get(i-1).getX());
```

```
                    tails.get(i).setY(tails.get(i-1).getY());
               }
          }
     }

     // Move first Tail to Head: Setzt erstes Tail an Position des Heads bei Bewegung
     if(tails.size() >= 1) {
          if(tails.get(0).isWait()) { // Wenn Sie sich noch nicht bewegen kann
               tails.get(0).setWait(false);
          } else { // Wenn Sie sich bewegen kann
               tails.get(0).setX(head.getX());
               tails.get(0).setY(head.getY());
          }
     }

     // Move Head
     switch(head.getDir()) {
          case RIGHT:          // Wenn die Schlange nach rechts geht geht der Kopf auf der X-Achse 1 nach rechts, unten jeweils das gleiche Prinzip
               head.setX(head.getX() + 1);
               break;
          case UP:
               head.setY(head.getY() - 1);
               break;
          case LEFT:
               head.setX(head.getX() - 1);
               break;
          case DOWN:
               head.setY(head.getY() + 1);
               break;
     }
```

```
        }

        // Position zu den Koordinaten
        public static Point ptc(int x, int y) { // erstellt ein neues
        Punkt-Objekt mit den Koordinaten x und y
           Point p = new Point(0, 0);
           p.x = x * 32 + Gui.xoff;
           p.y = y * 32 + Gui.yoff;

           return p;
        }

}
```

```
/**
 * @author Manuel
 */

package game;

import java.util.concurrent.ThreadLocalRandom;

public class PickUp { // Diese Klasse erstellt ein PickUp-Objekt (Apfel)
        int x;
        int y;
        public static boolean isGolden = false; // Es gibt selten auch
        goldene Äpfel

        public PickUp() {
           isGolden = false; // Der erste Apfel kann nicht golden sein
           this.setX(ThreadLocalRandom.current().nextInt(0,15));     //
        Random Koordinaten zwischen 0 und 15 für Äpfel
           this.setY(ThreadLocalRandom.current().nextInt(0,15));
           // Hier soll verhindert werden, dass ein Apfel in einem Tail
        spawnt
           for(int i = 0; i < Snake.tails.size(); i++) { // Iterieren
        durch Tails
                 if(x    ==    Snake.tails.get(i).getX()    &&    y    ==
        Snake.tails.get(i).getY()) {
                       this.reset();
                 }
           }
        }

        public void reset() {
           // Golden PickUp
           isGolden = false;
```

```
   int randomNum = ThreadLocalRandom.current().nextInt(1, 10 +
1); // Zufallszahl von 1-10
   if (randomNum == 5) {
        isGolden = true;
   }
   // Normal Pickup
   this.setX(ThreadLocalRandom.current().nextInt(0,15));
   this.setY(ThreadLocalRandom.current().nextInt(0,15));
   // Hier soll verhindert werden, dass ein Apfel in einem Tail,
obstacle oder bonus spawnt
   for(int i = 0; i < Snake.tails.size(); i++) { // Iterieren
durch Tails
        if(x    ==    Snake.tails.get(i).getX()    &&    y    ==
Snake.tails.get(i).getY()) {
             this.reset();
        }
        if(x  ==  Snake.obstacle.getX()  &&  y  ==  Snake.obsta-
cle.getY()) {
             this.reset();
        }
        if(x == Snake.bonus.getX() && y == Snake.bonus.getY())
{
             this.reset();
        }
   }
}

public int getX() {
   return x;
}

public void setX(int x) {
   this.x = x;
}
```

```
        public int getY() {
            return y;
        }

        public void setY(int y) {
            this.y = y;
        }
}
```

```
/**
 * @author Manuel
 */

package game;

import java.util.concurrent.ThreadLocalRandom;

import clocks.GameClock;

public class Obstacle { // Diese Klasse erstellt ein Hindernis-Objekt
        int x;
        int y;
        public static boolean isBlack = false; // Hindernisse für den
        Hard-Mode

        public Obstacle() { // Hindernisse im Hardmode sind schwarz
           if (GameClock.difficulty == "hard") {
                isBlack = true;
           } else {
                isBlack = false;
           }
           this.setX(ThreadLocalRandom.current().nextInt(0,15));     //
        Random Koordinaten zwischen 0 und 15 für Hindernisse für X und
        Y Achse
           this.setY(ThreadLocalRandom.current().nextInt(0,15));
           // Hier soll verhindert werden, dass ein Obstacle in einem
        Tail oder PickUp spawnt
           for(int i = 0; i < Snake.tails.size(); i++) { // Iterieren
        durch Tails
                if(x    ==    Snake.tails.get(i).getX()    &&    y    ==
        Snake.tails.get(i).getY()) {
                     this.reset();
```

```
            }
            if(x == Snake.pickup.getX() && y == Snake.pickup.getY())
{
                  this.reset();
            }
      }
}

public void reset() {
      if (GameClock.difficulty == "hard") {   // Hindernisse im
Hardmode sind schwarz
            isBlack = true;
      } else {
            isBlack = false;
      }
      this.setX(ThreadLocalRandom.current().nextInt(0,15));
      this.setY(ThreadLocalRandom.current().nextInt(0,15));
      // Hier soll verhindert werden, dass ein Obstacle in einem
Tail, PickUp oder Bonus spawnt
      for(int i = 0; i < Snake.tails.size(); i++) { // Iterieren
durch Tails
            if(x    ==    Snake.tails.get(i).getX()    &&    y    ==
Snake.tails.get(i).getY()) {
                  this.reset();
            }
            if(x == Snake.pickup.getX() && y == Snake.pickup.getY())
{
                  this.reset();
            }
            if(x == Snake.bonus.getX() && y == Snake.bonus.getY())
{
                  this.reset();
            }
      }
```

```
        }

        public int getX() {
           return x;
        }

        public void setX(int x) {
           this.x = x;
        }

        public int getY() {
           return y;
        }

        public void setY(int y) {
           this.y = y;
        }
}
/**
 * @author Manuel
 */

package gui;

import java.awt.Image;

import javax.swing.ImageIcon;
import javax.swing.JFrame;
import actions.KeyHandler;

public class Gui { // Gui Klasse, die einen neuen JFrame erstellt und
        auf die Draw Klasse zugreift
```

```
public Gui() { // Schliesst allfällig übrig gebliebene JFrames bevor ein neuer erstellt wird
   if (jf != null) {
        jf.dispose();
   }
}

static JFrame jf; // Neuer statischer JFrame
Draw d; // Neues Draw Objekt
Image imgSnake = new ImageIcon(this.getClass().getResource("/snake.png")).getImage();

public static int width = 1000, height = 600; // Grösse des Rahmens
public static int xoff = 230, yoff = 20; // Abstand zum Oberen und seitlichen Rand des Fensters

public void create() { // Erstellt das Hauptfenster
   jf = new JFrame("Snakeology"); // Titel
   jf.setSize(width, height);
   jf.setDefaultCloseOperation(JFrame.EXIT_ON_CLOSE);
   jf.setLocationRelativeTo(null);
   jf.setLayout(null);
   jf.setResizable(false);
   jf.addKeyListener(new KeyHandler());
   jf.setIconImage(imgSnake);

   d = new Draw();
   d.setBounds(0, 0, width, height);
   d.setVisible(true);
   jf.add(d);

   jf.requestFocus();
   jf.setVisible(true);
```

```
            }

}
```

```
/**
 * @author Manuel
 */

package gui;

import java.awt.BorderLayout;
import java.awt.FlowLayout;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;

import clocks.GameClock;
import game.Snake;

import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.ActionEvent;
import javax.swing.JLabel;
import java.awt.Font;
import java.awt.Image;

import javax.swing.SwingConstants;

public class PauseScreen extends JDialog { // Dieser JDialog mit der
        Pausierung des Spiels aufgerufen und kann das Spiel beenden, oder
        weiterfahren lassen
        private static final long serialVersionUID = 1L;

        private final JPanel contentPanel = new JPanel();
```

```
public PauseScreen() {
    setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
    setTitle("Snakeology - paused");

    Gui.jf.setEnabled(false);

    setBounds(100, 100, 450, 300);
    getContentPane().setLayout(new BorderLayout());
    contentPanel.setBorder(new EmptyBorder(5, 5, 5, 5));
    getContentPane().add(contentPanel, BorderLayout.CENTER);
    contentPanel.setLayout(null);
    setLocationRelativeTo(null);
    Image imgSnake = new Image-
Icon(this.getClass().getResource("/snake.png")).getImage();
    setIconImage(imgSnake);

    JLabel lblGamePaused = new JLabel("Game paused");
    lblGamePaused.setHorizontalAlignment(SwingConstants.CENTER);
    lblGamePaused.setBounds(96, 97, 241, 40);
    lblGamePaused.setFont(new Font("Tahoma", Font.PLAIN, 33));
    contentPanel.add(lblGamePaused);

    JPanel buttonPane = new JPanel();
    buttonPane.setLayout(new FlowLayout(FlowLayout.RIGHT));
    getContentPane().add(buttonPane, BorderLayout.SOUTH);

    JButton btnResume = new JButton("Resume");

    btnResume.setActionCommand("OK");
    buttonPane.add(btnResume);
    getRootPane().setDefaultButton(btnResume);

    JButton btnQuit = new JButton("Quit");
```

```
            btnQuit.setActionCommand("Cancel");
            buttonPane.add(btnQuit);

            btnResume.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) { // Hebt die
        Pausierung des Spiels auf und es geht weiter
                        Gui.jf.setEnabled(true);
                        GameClock gc = new GameClock();
                        gc.start();
                        GameClock.running = true;
                        Snake.move();
                        Snake.waitToMove = false;
                        dispose(); // Schliesst den Pause-Screen
                  }
            });

            btnQuit.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) { // Beendet
        das Spiel
                        Gui.jf.dispose(); // Schliesst die Spieloberfläche
                        dispose(); // Schliesst den Pause-Screen
                  }
            });

            // Close whole Game on Window Close
            this.addWindowListener(new WindowAdapter() {
                  public void windowClosing(WindowEvent e) {
                        Gui.jf.dispose(); // Schliesst die Spieloberfläche
        bei Klick auf X
                  }
             });
        }
}
```

```
/**
 * @author Manuel
 */

package gui;

import java.awt.*;
import java.io.BufferedReader;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.IOException;

import javax.swing.*;
import javax.swing.border.MatteBorder;

import clocks.GameClock;
import game.Bonus;
import game.Obstacle;
import game.PickUp;
import game.Snake;

public class Draw extends JLabel { // Hier wird die Spieloberfläche mit
        Grid und allen Extras erstellt und gezeichnet
        private static final long serialVersionUID = 1L;

        Point p;
        private String crntUser; // Aktueller Benutzer
        private String recUser; // Benutzername des Rekordhalters
        private String recScore; // Punkestand des Rekordhalters
        private String recDifficulty; // Schwierigkeitsgrad des Rekord-
        halters
        private String record; // // String der in speziellem, mit Se-
        mikolon getrenntem Format die Rekorddaten abspeichert
```

```
public static JLabel lblHeart = new JLabel(""); // Label für das Herzbild (extraLife)
public static JProgressBar b = new JProgressBar(); // ProgressBar für die Anzeige wie lange der aktive Bonus noch anhält
String splitter = ";";  // Splitzeichen für den "record" String

@SuppressWarnings("resource")
protected void paintComponent(Graphics g) { // übergabe des "Graphics" Objekts

   // Current User
   try {  // liest den aktuellen Benutzer aus dem entsprechenden File aus
         BufferedReader crntUserReader = new BufferedReader(new FileReader("currentUser.txt"));
         crntUser = "";
         String username = crntUserReader.readLine();
         while(username != null) { // lesen bis keine Zeile mehr
               crntUser = username;
               username = crntUserReader.readLine();
         }
   } catch (FileNotFoundException e1) {
         e1.printStackTrace();
   } catch (IOException e1) {
         e1.printStackTrace();
   }

   // get Record

   try { // liesst den aktuellen Rekord aus dem entsprechenden File aus
         BufferedReader recReader = new BufferedReader(new FileReader("record.txt"));
         record = "";
```

```
            String rec = recReader.readLine();
            while (rec != null) {
                  record = rec;
                  rec = recReader.readLine();
            }
            if(record == "") {
                  record = " " + splitter + 0 + splitter + " ";
            }
            String[] recordArr = record.split(splitter);
            recUser = recordArr[0];
            recScore = recordArr[1];
            recDifficulty = recordArr[2];
      } catch (FileNotFoundException e1) {
            e1.printStackTrace();
      } catch (IOException e1) {
            e1.printStackTrace();
      }

      super.paintComponent(g); // Zugriff auf die paintComponent
Funktion um das Interface zu zeichnen
      Graphics2D g2d = (Graphics2D) g;
      g2d.setRenderingHint(RenderingHints.KEY_ANTIALIASING,   Ren-
deringHints.VALUE_ANTIALIAS_OFF);

      // Draw Background
      g.setColor(Color.LIGHT_GRAY);
      g.fillRect(0, 0, Gui.width, Gui.height);

      // Draw Username
      JLabel lblUser = new JLabel("User: " + crntUser);
      lblUser.setFont(new Font("Arial", Font.PLAIN, 18));
      lblUser.setBounds(760, 35, 200, 35);
      lblUser.setForeground(Color.BLACK);
      this.add(lblUser);
```

```
   // Draw Heart
   if (GameClock.running == true) {
        Image img = new Image-
Icon(this.getClass().getResource("/heart48.png")).getImage();
        lblHeart.setIcon(new ImageIcon(img));
        lblHeart.setBounds(145, 50, 48, 48);
        this.add(lblHeart);
   }

   // Draw Snake Tails
   g.setColor(new Color(51, 204, 51)); // Schlangenfarbe für
Tails
   for(int i = 0; i < Snake.tails.size(); i++) {
        p = Snake.ptc(Snake.tails.get(i).getX(),
Snake.tails.get(i).getY());
        g.fillRect(p.x, p.y, 32, 32);
   }

   // Draw Snake Head
    g.setColor(new Color(0,153,0));
    p = Snake.ptc(Snake.head.getX(), Snake.head.getY());
    g.fillRect(p.x, p.y, 32, 32);

   // Draw PickUp
    if (PickUp.isGolden) {
         g.setColor(new Color(255, 200, 0));
    } else {
         g.setColor(new Color(204,51,0));
    }
    p = Snake.ptc(Snake.pickup.getX(), Snake.pickup.getY());
    g.fillRect(p.x,  p.y, 32,  32);

    // Draw Obstacle
```

```
    if (GameClock.difficulty == "hard" || GameClock.difficulty
== "normal" ) {
          if (Obstacle.isBlack) {
               g.setColor(new Color(0,0,0));
          } else {
               g.setColor(new Color(128,128,128));
          }
          p  =  Snake.ptc(Snake.obstacle.getX(),  Snake.obsta-
cle.getY());
          g.fillRect(p.x,  p.y, 32,  32);
    }

   // Draw Bonus
   if (GameClock.boniOn) {
          if (Bonus.bonus == "slowdown") {
               g.setColor(new Color(57, 196, 182));
          } else if (Bonus.bonus == "extraLife") {
               g.setColor(new Color(255, 153, 0));
          } else if (Bonus.bonus == "speedup") {
               g.setColor(new Color(0, 0, 204));
          } else if (Bonus.bonus == "double") {
               g.setColor(new Color(204, 0, 255));
          }
          p = Snake.ptc(Snake.bonus.getX(), Snake.bonus.getY());
          g.fillRect(p.x,  p.y, 32,  32);
   }

   // Draw Grid
   g.setColor(Color.GRAY);
   for(int x = 0; x < 16; x++) {
        for (int y = 0; y < 16; y++) {
             g.drawRect(x * 32 + Gui.xoff, y * 32 + Gui.yoff,
32, 32);
        }
```

```
        }

        // Draw Border
        g.setColor(Color.BLACK);
        g.drawRect(Gui.xoff,  Gui.yoff,  512,  512);

        // Draw Score
        g.setFont(new Font("Arial", Font.BOLD, 20));
        g.drawString("Score: " + Snake.score,  130, 37);
        g.drawString("Your best: " + Snake.bestscore,  760,  37);

        // Draw Record
        g.drawString("All-time record",  760,  165);
        g.setFont(new Font("Arial", Font.BOLD, 17));
        g.drawString("User: " + recUser,  760,  190);
        g.drawString("Difficulty: " + recDifficulty,  760,  210);
        g.drawString("Score: " + recScore,  760,  230);

        // Draw Bonus ProgressBar
        b.setOrientation(SwingConstants.VERTICAL);
        b.setMinimum(0);
        b.setMaximum(100);
        b.setBackground( new Color(0, 0, 0, 0));
        b.setBounds(760, 370, 80, 163);
        b.setBorder(new MatteBorder(1, 1, 1, 1, (Color) Color.GRAY));
        this.add(b);

        // Draw Hotkeys
        g.drawString("(m) Manual",  70,  495);
        g.drawString("(space) Pause Game",  38,  520);

        // Draw Boni Legend
        if (GameClock.boniOn) { // Muss nur angezeigt werden, wenn
    die Boni aktiviert sind
```

```
                g.drawString("Slowdown",  103,  359);
                g.drawString("Speedup",  103,  389);
                g.drawString("Double Points",  103,  419);
                g.drawString("Extra Life",  103,  449);

                JLabel lblSlowdown = new JLabel("");
                lblSlowdown.setBounds(70, 340, 25, 25);
                lblSlowdown.setBackground(new Color(57, 196, 182));
                lblSlowdown.setOpaque(true);
                this.add(lblSlowdown);

                JLabel lblSpeedup = new JLabel("");
                lblSpeedup.setBounds(70, 370, 25, 25);
                lblSpeedup.setBackground(new Color(0, 0, 204));
                lblSpeedup.setOpaque(true);
                this.add(lblSpeedup);

                JLabel lblDouble = new JLabel("");
                lblDouble.setBounds(70, 400, 25, 25);
                lblDouble.setBackground(new Color(204, 0, 255));
                lblDouble.setOpaque(true);
                this.add(lblDouble);

                JLabel lblExtraLife = new JLabel("");
                lblExtraLife.setBounds(70, 430, 25, 25);
                lblExtraLife.setBackground(new Color(255, 153, 0));
                lblExtraLife.setOpaque(true);
                this.add(lblExtraLife);
            }
            repaint();
        }

}
```

```
/**
 * @author Manuel & Sven
 */

package gui;

import java.awt.BorderLayout;
import java.awt.Image;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;

import clocks.GameClock;

import javax.swing.JTextField;
import java.awt.event.ActionListener;
import java.io.BufferedWriter;
import java.io.FileWriter;
import java.io.IOException;
import java.awt.event.ActionEvent;
import javax.swing.JLabel;
import javax.swing.JOptionPane;
import javax.swing.JCheckBox;

public class LoginScreen extends JDialog { // Dieser JDialog ermöglicht
        die Auswahl einer Schwierigkeitsstufe, die Aktivierung von Boni
        und die Eingabe eines Benutzernamens
        private static final long serialVersionUID = 1L;

        private final JPanel contentPanel = new JPanel();
```

```
private JTextField tfUsername; // textField für die Eingabe des Usernames
public static boolean loginManual = false; // Wurde das Manual über den Login-Screen gestartet?

@SuppressWarnings({ "resource", "unused" })
public LoginScreen() throws IOException {
   setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
   setTitle("Snakeology - Login");
   setBounds(100, 100, 450, 300);
   getContentPane().setLayout(new BorderLayout());
   contentPanel.setBorder(new EmptyBorder(5, 5, 5, 5));
   getContentPane().add(contentPanel, BorderLayout.CENTER);
   contentPanel.setLayout(null);
   setLocationRelativeTo(null);
   Image imgSnake = new ImageIcon(this.getClass().getResource("/snake.png")).getImage();
   setIconImage(imgSnake);

   tfUsername = new JTextField(); // In dieses Feld kann der Benutzername eingetragen werden
   tfUsername.setBounds(108, 130, 151, 26);
   contentPanel.add(tfUsername);
   tfUsername.setColumns(10);

   JLabel lblNewLabel = new JLabel("Username:");
   lblNewLabel.setBounds(108, 115, 72, 14);
   contentPanel.add(lblNewLabel);

   JLabel lblNewLabel_1 = new JLabel("Difficulty");
   lblNewLabel_1.setBounds(108, 46, 72, 14);
   contentPanel.add(lblNewLabel_1);

   JButton btnEasy = new JButton("easy");
```

```
btnEasy.setBounds(108, 71, 65, 23);
contentPanel.add(btnEasy);

JButton btnNormal = new JButton("normal");

btnNormal.setBounds(176, 71, 83, 23);
contentPanel.add(btnNormal);

JButton btnHard = new JButton("hard");

btnHard.setBounds(262, 71, 65, 23);
contentPanel.add(btnHard);

JCheckBox checkBoniOn = new JCheckBox("Boni");
checkBoniOn.setBounds(268, 132, 59, 23);
contentPanel.add(checkBoniOn);

JButton btnManual = new JButton("Manual");

btnManual.setBounds(8, 231, 78, 23);
contentPanel.add(btnManual);

JButton okButton = new JButton("OK");
okButton.setActionCommand("OK");
okButton.setBounds(292, 231, 53, 23);
contentPanel.add(okButton);

JButton cancelButton = new JButton("Cancel");
cancelButton.setActionCommand("Cancel");
cancelButton.setBounds(350, 231, 77, 23);
contentPanel.add(cancelButton);
```

```
   // Erstellt, falls nicht schon vorhanden, eine neue Textdatei
"currentUser"
   FileWriter fwCrntUser = new FileWriter("currentUser.txt",
false);
   BufferedWriter userWriter = new BufferedWriter(fwCrntUser);
   // Erstellt, falls nicht schon vorhanden, eine neue Textdatei
"record"
   FileWriter fwRecord = new FileWriter("record.txt", true);
   BufferedWriter recordWriter = new BufferedWriter(fwRecord);

   okButton.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent arg0) {
             if (tfUsername.getText().isEmpty()) { // Wenn kein
Benutzername eingegeben wird, erscheint eine Fehlermeldung
                  JOptionPane.showMessageDialog(null,"Please
enter a valid Username.","Invalid username",JOptionPane.WARN-
ING_MESSAGE);
                  tfUsername.setText("");
             } else { // Standardschwierigkeitsstufe ist "nor-
mal", wenn keine ausgewählt wurde
                  if (GameClock.difficulty != "easy" && Game-
Clock.difficulty != "normal" && GameClock.difficulty != "hard")
{
                       GameClock.difficulty = "normal";
                  }
                  try {
                       // Username in BufferedReader speichern
                       userWriter.write(tfUsername.get-
Text());
                       userWriter.flush(); // daten übertragen
                       tfUsername.setText("");
                  } catch (IOException e) {
                       e.printStackTrace();
                  }
                  Gui g = new Gui();
```

```
                    GameClock gc = new GameClock();
                    g.create();
                    gc.start(); // Methode aus Thread
                    if (checkBoniOn.isSelected()) {
                         GameClock.boniOn = true;
                    } else {GameClock.boniOn = false;}
                    dispose();
               }
          }
     });

     okButton.setActionCommand("OK");
     getRootPane().setDefaultButton(okButton);

     cancelButton.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) {
               dispose(); // Schliest das Fenster
          }
     });

     cancelButton.setActionCommand("Cancel");

     btnManual.addActionListener(new ActionListener() {
          public void actionPerformed(ActionEvent e) { // Öffnet
die Bedienungsanleitung
               loginManual = true;
               ManualScreenBasics ManualBS = new ManualScreenBa-
sics();
               ManualBS.setDefaultCloseOperation(JDialog.DIS-
POSE_ON_CLOSE);
               ManualBS.setVisible(true);
          }
     });
```

```
            btnEasy.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) { // Toggle
            Button easy
                        GameClock.difficulty = "easy";
                        btnEasy.setEnabled(false);
                        btnNormal.setEnabled(true);
                        btnHard.setEnabled(true);
                  }
            });

            btnNormal.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent e) { // Toggle
            Button normal
                        GameClock.difficulty = "normal";
                        btnEasy.setEnabled(true);
                        btnNormal.setEnabled(false);
                        btnHard.setEnabled(true);
                  }
            });

            btnHard.addActionListener(new ActionListener() {
                  public void actionPerformed(ActionEvent arg0) { // Tog-
            gle Button hard
                        GameClock.difficulty = "hard";
                        btnEasy.setEnabled(true);
                        btnNormal.setEnabled(true);
                        btnHard.setEnabled(false);
                  }
            });
      }
}
```

```
/**
 * @author Manuel
 */

package gui;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.FlowLayout;
import java.awt.Font;
import java.awt.Image;
import java.io.BufferedReader;
import java.io.BufferedWriter;
import java.io.FileNotFoundException;
import java.io.FileReader;
import java.io.FileWriter;
import java.io.IOException;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;

import actions.Collision;
import clocks.GameClock;
import game.Snake;

import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.ActionEvent;
import javax.swing.JLabel;
import javax.swing.SwingConstants;
```

```
import javax.swing.JCheckBox;

public class DeathScreen extends JDialog { // Der Deathscreen ist ein
        JDialog der für die Abspeicherung der Rekorddaten und die Mög-
        lichkeit, das Spiel neuzustarten zuständig ist
        private static final long serialVersionUID = 1L;

        private final JPanel contentPanel = new JPanel();

        private String crntUser; // Aktueller Benutzername
        private String record; // String der in speziellem, mit Semikolon
        getrenntem Format die Rekorddaten abspeichert
        public boolean isNewRecord = false; // Ist der Rekord in der
        aktuellen Runde gebrochen worden?
        String splitter = ";"; // Splitzeichen für den "record" String

        @SuppressWarnings("resource")
        public DeathScreen() throws IOException {
           setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
           setTitle("Snakeology - You died");

           Gui.jf.setEnabled(false);

           // get Current User
           try { // liest den aktuellen Benutzer aus dem entsprechenden
        File aus
                 BufferedReader crntUserReader = new BufferedReader(new
        FileReader("currentUser.txt"));
                 crntUser = "";
                 String username = crntUserReader.readLine();
                 while(username != null) { // lesen bis keine Zeile mehr
                       crntUser = username;
                       username = crntUserReader.readLine();
                 }
```

```
    } catch (FileNotFoundException e1) {
        e1.printStackTrace();
    } catch (IOException e1) {
        e1.printStackTrace();
    }

    // New Record Handling
    try { // liesst den aktuellen Rekord aus dem entsprechenden File aus
        BufferedReader recReader = new BufferedReader(new File-Reader("record.txt"));
        record = "";
        String rec = recReader.readLine();
        while (rec != null) {
            record = rec;
            rec = recReader.readLine();
        }
        if(record == "") { // Handling bei leerem Rekord File
            record = " " + splitter + 0 + splitter + " ";
        }
        String[] recordArr = record.split(splitter); // Rekord wird in leserliches Format gesplitet
        int recordScore = Integer.parseInt(recordArr[1]); // Erreichte Punktzahl des Rekordes wird ausgelesen

        if (Snake.score > recordScore) { // Wenn die aktuelle Punktzahl die Rekordpunkzahl übertrumpft wird ein neuer Rekord gebildet und abgespeichert
            isNewRecord = true;
            // Record Writer
            String myScore = Integer.toString(Snake.score);
            FileWriter fwRecord = new FileWriter("record.txt", false);
```

```
                BufferedWriter recordWriter = new Buff-
eredWriter(fwRecord);
                try {
                      recordWriter.write("");
                      recordWriter.flush();
                      recordWriter.write(crntUser + splitter +
myScore + splitter + GameClock.difficulty); // Neuer Rekord wird
in speziellem Format abgespeichert
                      recordWriter.flush(); // Daten übertragen
                } catch (IOException e) {
                      e.printStackTrace();
                }
          }
                isNewRecord = true;
    } catch (FileNotFoundException e1) {
          e1.printStackTrace();
    } catch (IOException e1) {
          e1.printStackTrace();
    }

    setBounds(100, 100, 450, 300);
    getContentPane().setLayout(new BorderLayout());
    contentPanel.setBorder(new EmptyBorder(5, 5, 5, 5));
    getContentPane().add(contentPanel, BorderLayout.CENTER);
    contentPanel.setLayout(null);
    setLocationRelativeTo(null);
    Image imgSnake = new Image-
Icon(this.getClass().getResource("/snake.png")).getImage();
    setIconImage(imgSnake);

    JLabel lblNewLabel = new JLabel("YOU DIED");
    lblNewLabel.setFont(new Font("Tahoma", Font.PLAIN, 33));
    lblNewLabel.setBounds(133, 41, 162, 38);
    contentPanel.add(lblNewLabel);
```

```
JLabel lblUser = new JLabel("User: " + crntUser);
lblUser.setHorizontalAlignment(SwingConstants.CENTER);
lblUser.setFont(new Font("Arial", Font.PLAIN, 18));
lblUser.setBounds(143, 116, 133, 38);
lblUser.setForeground(Color.BLACK);
contentPanel.add(lblUser);

int usrScore = Snake.score;
JLabel label = new JLabel("Score: " + usrScore);
label.setHorizontalAlignment(SwingConstants.CENTER);
label.setForeground(Color.BLACK);
label.setFont(new Font("Arial", Font.PLAIN, 15));
label.setBounds(143, 138, 133, 38);
contentPanel.add(label);

JButton btnHard = new JButton("Hard");

btnHard.setBounds(266, 187, 89, 23);
contentPanel.add(btnHard);

JButton btnNormal = new JButton("Normal");

btnNormal.setBounds(167, 187, 89, 23);
contentPanel.add(btnNormal);

JButton btnEasy = new JButton("Easy");

btnEasy.setBounds(68, 187, 89, 23);
contentPanel.add(btnEasy);

JCheckBox checkBoniOn = new JCheckBox("Boni");
checkBoniOn.setBounds(296, 147, 59, 23);
contentPanel.add(checkBoniOn);
```

```
   if (GameClock.boniOn) { // Schaut ob Boni in der Runde akti-
viert waren und setzt das Häckchen entsprechend
         checkBoniOn.setSelected(true);
   } else {
         checkBoniOn.setSelected(false);
   }

   if (isNewRecord) { // Meldung über neuen Rekord wird nur
angezeigt, wenn ein neuer Rekord ausgelöst wurde
         JLabel lblNewRecord = new JLabel("NEW RECORD !!!");
         lblNewRecord.setFont(new   Font("Tahoma",   Font.BOLD,
16));
         lblNewRecord.setHorizontalAlignment(SwingCon-
stants.CENTER);
         lblNewRecord.setBounds(112, 83, 195, 38);
         contentPanel.add(lblNewRecord);
   }

   JPanel buttonPane = new JPanel();
   buttonPane.setLayout(new FlowLayout(FlowLayout.RIGHT));
   getContentPane().add(buttonPane, BorderLayout.SOUTH);

   JButton btnAgain = new JButton("Play Again");

   btnAgain.setActionCommand("OK");
   buttonPane.add(btnAgain);
   getRootPane().setDefaultButton(btnAgain);

   JButton btnQuit = new JButton("Quit");

   btnQuit.setActionCommand("Cancel");
   buttonPane.add(btnQuit);
```

```
   btnAgain.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) { // Alles
wird wieder zurückgesetzt und das Spiel wird mit dem gleichen
Benutzer neugestartet
            // Score zurücksetzen
            Snake.score = 0;
            Gui g = new Gui();
            GameClock gc = new GameClock();
            g.create();
            gc.start(); // Methode aus Thread zum starten der
Ticks
            Snake.pickup.reset();
            Snake.obstacle.reset();
            GameClock.running = true;
            if (checkBoniOn.isSelected()) { // Boni Handling
                GameClock.boniOn = true;
                Collision.activeBonus = "";
                Snake.bonus.reset();
            } else {GameClock.boniOn = false;}
            dispose();
        }
   });

   btnQuit.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) { // Beendet
den DeathScreen und das Spiel
            Gui.jf.dispose();
            dispose();
        }
   });

   if (GameClock.difficulty == "easy") { // Toggle Buttons für
Schwierigkeitsauswahl
        btnEasy.setEnabled(false);
```

```
            btnNormal.setEnabled(true);
            btnHard.setEnabled(true);
        } else if (GameClock.difficulty == "normal") {
            btnEasy.setEnabled(true);
            btnNormal.setEnabled(false);
            btnHard.setEnabled(true);
        } else if (GameClock.difficulty == "hard") {
            btnEasy.setEnabled(true);
            btnNormal.setEnabled(true);
            btnHard.setEnabled(false);
        }

        btnEasy.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) { // Toggle
Button "easy"
                GameClock.difficulty = "easy";
                btnEasy.setEnabled(false);
                btnNormal.setEnabled(true);
                btnHard.setEnabled(true);
            }
        });

        btnNormal.addActionListener(new ActionListener() {
            public void actionPerformed(ActionEvent e) { // Toggle
Button "normal"
                GameClock.difficulty = "normal";
                btnEasy.setEnabled(true);
                btnNormal.setEnabled(false);
                btnHard.setEnabled(true);
            }
        });

        btnHard.addActionListener(new ActionListener() {
```

```
                public void actionPerformed(ActionEvent arg0) { // Tog-
        gle Button "hard"
                    GameClock.difficulty = "hard";
                    btnEasy.setEnabled(true);
                    btnNormal.setEnabled(true);
                    btnHard.setEnabled(false);
                }
            });

            // Close whole Game on Window Close
            this.addWindowListener(new WindowAdapter() {
                public void windowClosing(WindowEvent e) { // Schliesst
        auch das Spielfeld wenn das X gedrückt wird
                    Gui.jf.dispose();
                }
             });
        }
}
```

```
/**
 * @author Sven
 */

package gui;

import java.awt.BorderLayout;
import java.awt.Color;
import java.awt.Font;
import java.awt.Image;
import java.awt.event.ActionEvent;
import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JLabel;
import javax.swing.JPanel;
import javax.swing.SwingConstants;
import javax.swing.border.EmptyBorder;

public class ManualScreenBasics extends JDialog { // Dieser JDialog ist
        die erste von zwei Seiten der Bedienungsanleitung und dient nur
        zur Darstellung von Information, hat also keine mathematische
        Funktion
        private static final long serialVersionUID = 1L;

        private final JPanel contentPanel = new JPanel();

        public ManualScreenBasics() {
           setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);
```

```
        if (LoginScreen.loginManual == false) {
              Gui.jf.setEnabled(false);
        }

        setTitle("Snakeology - Manual");
        setBounds(100, 100, 663, 410);
        getContentPane().setLayout(new BorderLayout());
        contentPanel.setBorder(new EmptyBorder(5, 5, 5, 5));
        getContentPane().add(contentPanel, BorderLayout.CENTER);
        contentPanel.setLayout(null);
        Image imgSnake = new Image-
    Icon(this.getClass().getResource("/snake.png")).getImage();
        setIconImage(imgSnake);

        JButton btnNext = new JButton("Next (1/2)");

        btnNext.setActionCommand("OK");
        btnNext.setBounds(527, 331, 112, 31);
        contentPanel.add(btnNext);

        JLabel lblManual = new JLabel("Manual");
        lblManual.setHorizontalAlignment(SwingConstants.LEFT);
        lblManual.setForeground(Color.BLACK);
        lblManual.setFont(new Font("Arial", Font.PLAIN, 25));
        lblManual.setBounds(23, 11, 126, 40);
        contentPanel.add(lblManual);

        JLabel lblBasics = new JLabel("Basics");
        lblBasics.setHorizontalAlignment(SwingConstants.LEFT);
        lblBasics.setForeground(Color.BLACK);
        lblBasics.setFont(new Font("Arial", Font.PLAIN, 18));
        lblBasics.setBounds(23, 62, 112, 22);
        contentPanel.add(lblBasics);
```

```
    JLabel lblNewLabel = new JLabel("PickUps");
    lblNewLabel.setFont(new Font("Arial", Font.BOLD, 14));
    lblNewLabel.setBounds(23, 98, 232, 23);
    contentPanel.add(lblNewLabel);

    JLabel lblYouHave = new JLabel("- Red PickUps add one tail to
the snake");
    lblYouHave.setFont(new Font("Arial", Font.PLAIN, 13));
    lblYouHave.setBounds(33, 124, 276, 23);
    contentPanel.add(lblYouHave);

    JLabel lblYouHave_1 = new JLabel("- Golden PickUps add three
tails to the snake");
    lblYouHave_1.setFont(new Font("Arial", Font.PLAIN, 13));
    lblYouHave_1.setBounds(33, 146, 276, 23);
    contentPanel.add(lblYouHave_1);

    JLabel lblNormal = new JLabel("Score");
    lblNormal.setFont(new Font("Arial", Font.BOLD, 14));
    lblNormal.setBounds(23, 222, 232, 23);
    contentPanel.add(lblNormal);

    JLabel lblGameSpeed = new JLabel("- The Score is always the
same as the tail length");
    lblGameSpeed.setFont(new Font("Arial", Font.PLAIN, 13));
    lblGameSpeed.setBounds(33, 246, 285, 23);
    contentPanel.add(lblGameSpeed);

    JLabel lblThereAre = new JLabel("- In the top left corner is
your current Score");
    lblThereAre.setFont(new Font("Arial", Font.PLAIN, 13));
    lblThereAre.setBounds(33, 268, 276, 23);
    contentPanel.add(lblThereAre);
```

```
    JLabel lblHard = new JLabel("Boni");
    lblHard.setFont(new Font("Arial", Font.BOLD, 14));
    lblHard.setBounds(349, 98, 232, 23);
    contentPanel.add(lblHard);

    JLabel lblFastGame = new JLabel("- You can enable or disable
Boni");
    lblFastGame.setFont(new Font("Arial", Font.PLAIN, 13));
    lblFastGame.setBounds(359, 124, 250, 23);
    contentPanel.add(lblFastGame);

    JLabel lblThereAre_1 = new JLabel("- If a Bonus is active you
can see ");
    lblThereAre_1.setFont(new Font("Arial", Font.PLAIN, 13));
    lblThereAre_1.setBounds(359, 146, 238, 23);
    contentPanel.add(lblThereAre_1);

    JLabel lblKillYouOn = new JLabel("  how long it will last");
    lblKillYouOn.setFont(new Font("Arial", Font.PLAIN, 13));
    lblKillYouOn.setBounds(359, 160, 238, 23);
    contentPanel.add(lblKillYouOn);

    JLabel lblPickupsAnd = new JLabel("- PickUps and obstacles
get reset after ");
    lblPickupsAnd.setFont(new Font("Arial", Font.PLAIN, 13));
    lblPickupsAnd.setBounds(33, 168, 276, 23);
    contentPanel.add(lblPickupsAnd);

    JLabel lblEveryFivePoints = new JLabel("  every five Points");
    lblEveryFivePoints.setFont(new   Font("Arial",   Font.PLAIN,
13));
    lblEveryFivePoints.setBounds(33, 184, 238, 23);
    contentPanel.add(lblEveryFivePoints);
```

```
   JLabel lblInThe = new JLabel("- In the top right corner is
the highest Score you have reached");
   lblInThe.setFont(new Font("Arial", Font.PLAIN, 13));
   lblInThe.setBounds(33, 290, 361, 23);
   contentPanel.add(lblInThe);

   JLabel lblIfYour = new JLabel("- If your Score is higher than
the current Record it will be saved ");
   lblIfYour.setFont(new Font("Arial", Font.PLAIN, 13));
   lblIfYour.setBounds(33, 311, 373, 23);
   contentPanel.add(lblIfYour);

   JLabel lblOnTheComputer = new JLabel("  on the Computer");
   lblOnTheComputer.setFont(new Font("Arial", Font.PLAIN, 13));
   lblOnTheComputer.setBounds(33, 327, 452, 23);
   contentPanel.add(lblOnTheComputer);

   JLabel lblOnDifficulty = new JLabel("- On difficulty easy the
extraLife-Bonus");
   lblOnDifficulty.setFont(new Font("Arial", Font.PLAIN, 13));
   lblOnDifficulty.setBounds(359, 180, 238, 23);
   contentPanel.add(lblOnDifficulty);

   JLabel lblRestoresYourExtralife = new JLabel("  restores your
extraLife if you lost it before");
   lblRestoresYourExtralife.setFont(new          Font("Arial",
Font.PLAIN, 13));
   lblRestoresYourExtralife.setBounds(359, 196, 260, 23);
   contentPanel.add(lblRestoresYourExtralife);
   setLocationRelativeTo(null);

   btnNext.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) { // Leitet
weiter auf Seite zwei der Bedienungsanleitung
```

```
                ManualScreenDifficulties manualDS = new Manu-
        alScreenDifficulties();
                manualDS.setDefaultCloseOperation(JDialog.DIS-
        POSE_ON_CLOSE);
                manualDS.setVisible(true);
                dispose(); // Schliesst das Fenster
             }
        });

        // Close whole Game on Window Close
        this.addWindowListener(new WindowAdapter() {
             public void windowClosing(WindowEvent e) {
                if (LoginScreen.loginManual == false) {
                Gui.jf.dispose(); // Schliesst das ganze Spiel
        falls es bereits offen ist
                }
             }
         });
        }

}
```

```
/**
 * @author Sven
 */

package gui;

import java.awt.BorderLayout;

import javax.swing.ImageIcon;
import javax.swing.JButton;
import javax.swing.JDialog;
import javax.swing.JPanel;
import javax.swing.border.EmptyBorder;

import clocks.GameClock;
import game.Snake;

import java.awt.event.ActionListener;
import java.awt.event.WindowAdapter;
import java.awt.event.WindowEvent;
import java.awt.event.ActionEvent;
import javax.swing.JLabel;
import java.awt.Font;
import java.awt.Image;
import java.awt.Color;
import javax.swing.SwingConstants;

public class ManualScreenDifficulties extends JDialog { // Dieser JDia-
        log ist die zweite von zwei Seiten der Bedienungsanleitung und
        dient nur zur Darstellung von Information, hat also keine mathe-
        matische Funktion
        private static final long serialVersionUID = 1L;

        private final JPanel contentPanel = new JPanel();
```

```
public ManualScreenDifficulties() {
   setDefaultCloseOperation(JDialog.DISPOSE_ON_CLOSE);

   if (LoginScreen.loginManual == false) {
        Gui.jf.setEnabled(false);
   }

   setTitle("Snakeology - Manual");
   setBounds(100, 100, 663, 410);
   getContentPane().setLayout(new BorderLayout());
   contentPanel.setBorder(new EmptyBorder(5, 5, 5, 5));
   getContentPane().add(contentPanel, BorderLayout.CENTER);
   contentPanel.setLayout(null);
   setLocationRelativeTo(null);
   Image imgSnake = new Image-
Icon(this.getClass().getResource("/snake.png")).getImage();
   setIconImage(imgSnake);

   JLabel lblManual = new JLabel("Manual");
   lblManual.setHorizontalAlignment(SwingConstants.LEFT);
   lblManual.setForeground(Color.BLACK);
   lblManual.setFont(new Font("Arial", Font.PLAIN, 25));
   lblManual.setBounds(23, 11, 126, 40);
   contentPanel.add(lblManual);

   JLabel lblDifficulties = new JLabel("Difficulties");
   lblDifficulties.setHorizontalAlignment(SwingConstants.LEFT);
   lblDifficulties.setForeground(Color.BLACK);
   lblDifficulties.setFont(new Font("Arial", Font.PLAIN, 18));
   lblDifficulties.setBounds(23, 62, 112, 22);
   contentPanel.add(lblDifficulties);

   JLabel lblNewLabel = new JLabel("Easy");
```

```
        lblNewLabel.setFont(new Font("Arial", Font.BOLD, 14));
        lblNewLabel.setBounds(23, 98, 232, 23);
        contentPanel.add(lblNewLabel);

        JLabel lblYouHave = new JLabel("- Slow game speed");
        lblYouHave.setFont(new Font("Arial", Font.PLAIN, 13));
        lblYouHave.setBounds(33, 124, 250, 23);
        contentPanel.add(lblYouHave);

        JLabel lblYouHave_1 = new JLabel("- You have an extra life
    from the start");
        lblYouHave_1.setFont(new Font("Arial", Font.PLAIN, 13));
        lblYouHave_1.setBounds(33, 146, 238, 23);
        contentPanel.add(lblYouHave_1);

        JLabel lblYouCan = new JLabel("- You can go through walls");
        lblYouCan.setFont(new Font("Arial", Font.PLAIN, 13));
        lblYouCan.setBounds(33, 166, 238, 23);
        contentPanel.add(lblYouCan);

        JLabel lblNormal = new JLabel("Normal");
        lblNormal.setFont(new Font("Arial", Font.BOLD, 14));
        lblNormal.setBounds(23, 223, 232, 23);
        contentPanel.add(lblNormal);

        JLabel lblGameSpeed = new JLabel("-  Medium game speed (in-
    creasing)");
        lblGameSpeed.setFont(new Font("Arial", Font.PLAIN, 13));
        lblGameSpeed.setBounds(33, 250, 250, 23);
        contentPanel.add(lblGameSpeed);

        JLabel lblThereAre = new JLabel("- There are grey obstacles
    which");
        lblThereAre.setFont(new Font("Arial", Font.PLAIN, 13));
```

```
        lblThereAre.setBounds(33, 272, 238, 23);
        contentPanel.add(lblThereAre);

        JLabel label_3 = new JLabel("- You can go through Walls");
        label_3.setFont(new Font("Arial", Font.PLAIN, 13));
        label_3.setBounds(33, 306, 238, 23);
        contentPanel.add(label_3);

        JLabel lblRemoveOneTail = new JLabel("  remove one tail on
collision");
        lblRemoveOneTail.setFont(new Font("Arial", Font.PLAIN, 13));
        lblRemoveOneTail.setBounds(33, 286, 238, 23);
        contentPanel.add(lblRemoveOneTail);

        JLabel lblHard = new JLabel("Hard");
        lblHard.setFont(new Font("Arial", Font.BOLD, 14));
        lblHard.setBounds(349, 98, 232, 23);
        contentPanel.add(lblHard);

        JLabel lblFastGame = new JLabel("- Fast game speed (increas-
ing)");
        lblFastGame.setFont(new Font("Arial", Font.PLAIN, 13));
        lblFastGame.setBounds(359, 124, 250, 23);
        contentPanel.add(lblFastGame);

        JLabel lblThereAre_1 = new JLabel("- There are black obstacles
that");
        lblThereAre_1.setFont(new Font("Arial", Font.PLAIN, 13));
        lblThereAre_1.setBounds(359, 146, 238, 23);
        contentPanel.add(lblThereAre_1);

        JLabel lblYouCant = new JLabel("- You can't go through
Walls");
        lblYouCant.setFont(new Font("Arial", Font.PLAIN, 13));
```

```
        lblYouCant.setBounds(359, 180, 238, 23);
        contentPanel.add(lblYouCant);

        JLabel lblKillYouOn = new JLabel("  kill you on collision");
        lblKillYouOn.setFont(new Font("Arial", Font.PLAIN, 13));
        lblKillYouOn.setBounds(359, 162, 238, 23);
        contentPanel.add(lblKillYouOn);

        JButton btnBack = new JButton("Back (2/2)");

        btnBack.setActionCommand("OK");
        btnBack.setBounds(383, 331, 112, 31);
        contentPanel.add(btnBack);

        JButton btnClose = new JButton("Close Manual");
        btnClose.setActionCommand("OK");
        btnClose.setBounds(505, 331, 134, 31);
        contentPanel.add(btnClose);

        JButton btnContinue = new JButton("Continue Game");
        btnContinue.setActionCommand("OK");
        btnContinue.setBounds(505, 331, 134, 31);
        contentPanel.add(btnContinue);

        if (LoginScreen.loginManual) { // Wenn die Anleitung vom
    Login-Screen aus gestartet wurde wird ein anderer Button an-
    gezeigt, als wenn es mitten im Spiel geöffnet wurde
            btnClose.setVisible(true);
            btnContinue.setVisible(false);
        } else {
            btnClose.setVisible(false);
            btnContinue.setVisible(true);
        }
```

```
   btnBack.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) { // Leitet
wieder zurück auf Seite eins der Bedienungsanleitung
            ManualScreenBasics manualBS = new ManualScreenBa-
sics();
            manualBS.setDefaultCloseOperation(JDialog.DIS-
POSE_ON_CLOSE);
            manualBS.setVisible(true);
            dispose();
        }
   });

   btnContinue.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) { // Hebt die
Pausierung des Spiels auf und es geht weiter
            Gui.jf.setEnabled(true);
            GameClock gc = new GameClock();
            gc.start();
            GameClock.running = true;
            Snake.move();
            Snake.waitToMove = false;
            dispose(); // Schliesst die Anleitung
        }
   });

   btnClose.addActionListener(new ActionListener() {
        public void actionPerformed(ActionEvent e) {
            LoginScreen.loginManual = false;
            dispose(); // Schliesst die Anleitung
        }
   });

   // Close whole Game on Window Close
   this.addWindowListener(new WindowAdapter() {
```

```
                public void windowClosing(WindowEvent e) {
                      if (LoginScreen.loginManual == false) {
                      Gui.jf.dispose();   // Schliesst das ganze Spiel
        falls es bereits offen ist
                      }
                      LoginScreen.loginManual = false;
                }
            });
        }
}
```