Überblick und Hinweise für den Korrektor

Im Folgenden soll ein kleiner Überblick über die wesentlichen Inhalte des Programms gegeben werden. Zum schnelleren und besseren Verständnis für Außenstehende werden die Anforderungen mit dem Grad ihrer Erfüllung, die wesentlichen zusätzlichen Features, die wichtigsten Klassen und Methoden mit ihren Aufgaben, ausgewählte Funktionen sowie einige Schwachstellen in der Programmierung unter Angabe von Gründen für ihre Entstehung aufgeführt und kurz erklärt.

Anforderungen

ANF 1: Das Programm verwaltet mindestens die folgenden Daten:

-Spieler (inkl. Computergegner) jeweils mit Name, Spielfarbe, Spielzeit und Spielzüge als Grundlage für einen Highscore-Mechanismus

-Spielbrett, d.h. die aktuellen Positionen aller Spielfiguren

Das Programm verwaltet all diese Daten inklusive einer Statistik für jeden Spieler, eines Spielnamens und den spielspezifischen Einstellungen.

ANF 2: Das Programm muss das Spielbrett als Teil der grafischen Benutzeroberfläche zur Verfügung stellen.

Das Spielfeld wird wie gefordert mit allen Figuren und allen benötigten Steuerelementen, sowie optionalen grafischen Hilfestellungen dargestellt.

ANF 3: Das Programm muss einen Regelsatz zur Verfügung stellen.

Es können nur Felder gezogen werden, die vorher durch die Methode getKorrekteFelder() der abstrakten Klasse Figur ermittelt worden sind. Die genauen und individuellen Zugregeln wurden in der Methode getMoeglicheFelder() jeder Figur spezifiziert.

ANF 4: Das Programm muss Ausgaben zum Spielstatus zur Verfügung stellen.

Das Programm stellt Ausgaben des Spielstatus während des Spiels in Form von Labels und am Ende des Spiels in Form von Hinweisfenstern zur Verfügung. Eine Highscore-Liste ist unter dem Menü-Punkt „Highscore“ abzurufen. Dort werden die Spieler nach ihrem Punktestand sortiert aufgelistet. Der Punktestand berechnet sich anhand verschiedener statistischer Daten in der Methode getScore() der Klasse Statistik.

ANF 5: Das Programm muss dem Benutzer das Aufgeben anbieten.

Das Programm bietet einen Aufgeben-Button an, mit dem ein Spiel gegen einen menschlichen oder einen Computergegner aufgegeben werden kann. In diesem Fall wird dieses Spiel für den aufgebenden Spieler als verloren gewertet.

Das Programm bietet weiterhin die Möglichkeit an, sich mit einem menschlichen Spieler auf ein Unentschieden zu einigen. Dies geschieht ebenfalls per Knopfdruck und der andere Spieler kann dieses Angebot in einem Bestätigungsdialog annehmen oder ablehnen.

Auch gegen einen Computergegner kann ein Unentschieden angefragt werden. Die genauen Voraussetzungen dafür, dass dieser das Angebot annimmt, sind in der Benutzerdokumentation unter dem Punkt „Remis gegen den Computer“ zu finden.

Die offiziell geltende 50-Züge-Regel ist ebenfalls implementiert. Tritt diese ein, kann der Spieler auf den Unentschieden-Button klicken und das Spiel endet ohne Einfluss des Gegners unentschieden. Ein Computergegner reklamiert beim Eintreten sofort die 50-Züge-Regel und beendet das Spiel damit automatisch.

Patt und Remis werden gleich gewertet.

ANF 6: Das Programm soll dem Benutzer eine Speichern/Laden-Funktion zur Verfügung stellen.

Ein Spiel lässt sich jederzeit mit dem Button „Spiel speichern“ abspeichern. Dabei werden der Name des Spiels und der aktuelle Zeitpunkt der Speicherung als Erkennungsmerkmal für ein späteres Laden gespeichert. Es werden alle Figuren und ihre Position auf dem Spielfeld, sowie die Namen und die Farben der Spieler, der aktuelle Spieler, die spielspezifischen Einstellungen und die Zugliste in Form einer vereinfachten Schachnotation in Textdateien (Spielname.txt) im Unterordner Spiele des Ordners settings gespeichert.

Es ist möglich, mehrere Spiele gleichzeitig zu spielen und zu speichern, es muss also nicht beim erneuten Programmstart das zuletzt gespielte Spiel geladen werden.

Um einen Datenverlust durch Programmabsturz oder vorschnelles Beenden durch den Benutzer zu verhindern, wird das aktuelle Spiel bei jedem Zug automatisch gespeichert. Zu erkennen ist dies an dem Zusatz „(autosave)“ hinter dem Spielnamen. Wird dieses Spiel beim erneuten Starten des Programms nicht geladen, wird dieses Spiel gelöscht. Es dient somit nur der unmittelbaren Datenwiederherstellung.

ANF 7: Das Programm soll optional „Profispielregeln“ zur Verfügung stellen.

Unter dem Menüpunkt „Einstellungen“ können neben grafischen Hilfestellungen wie dem Anzeigen von möglichen und bedrohten Feldern auch folgende Profiregeln aktiviert und deaktiviert werden:

Rochade, En-Passant-Schlagen, Schachmeldung, Einbeziehen in die Statistik und die Dauer der Bedenkzeit

Ist eine Zugzeitbeschränkung eingestellt, wird das Spiel automatisch nach Ablauf dieser Zugzeit abgebrochen und der Spieler, der nicht rechtzeitig gezogen hat, verliert dieses Spiel. Ein Ablauf der Zeit wird wie ein Matt gewertet.
Sollte mit Bedenkzeit die gesamte Zeit des Spielers für alle Züge gemeint gewesen sein, so war die Aufgabenstellung an dieser Stelle nicht eindeutig genug formuliert und wir bitten unsere falsche Implementierung zu entschuldigen. Die erforderlichen Änderungen wären jedoch nicht sehr aufwendig, da bereits alle benötigten Methoden vorhanden sind.

ANF 8: Das Programm kann Computergegner zur Verfügung stellen.

Das Programm stellt vier verschieden starke Computergegner zur Verfügung. Der leichteste Gegner zieht nach simplen Regeln wie sie in den Anforderungen vorgeschlagen wurden, der stärkste Gegner berechnet vier Halbzüge voraus und ermittelt anhand einer Bewertungsfunktion die Lukrativität dieser Spielsituation. Näheres dazu unter dem Punkt Computergegner und in der Benutzerdokumentation unter dem Punkt „Computergegner“.

ANF 9: Das Programm kann eine Statistik zur Verfügung stellen.

Nahezu alle sowie noch ein paar weitere Punkte aus der Aufgabenstellung wurden in die Statistik übernommen. Diese wird nach jedem Spiel aktualisiert (sofern die Einstellung dies vorsieht) und kann für jeden Spieler individuell unter dem Menüpunkt „Statistik“ eingesehen werden.

ANF 10: Das Programm kann das Spielbrett drehen.

Aktiviert man die entsprechende Option in den Einstellungen, wird das Spielfeld wie in den Anforderungen beschrieben bei jedem Halbzug gedreht. Dies gilt nicht für ein Spiel gegen einen Computergegner. Dabei wird lediglich darauf geachtet, dass der menschliche Spieler von unten nach oben zieht (auch wenn er die schwarzen Figuren gewählt hat).

Zusätzliche Features

Zug rückgängig machen

Das Programm stellt die Möglichkeit zur Verfügung, den letzten Zug rückgängig zu machen. Besonders beim Spiel gegen einen Computergegner kann dies oftmals sehr hilfreich sein. Diese Option ist nur im Spiel möglich. Wenn das Spiel durch Matt, Patt, Ablauf der Zeit, Aufgabe oder Einigung auf Unentschieden endet, besteht nicht mehr die Möglichkeit, den letzten Zug rückgängig zu machen.

Ausgabe der durchgeführten Züge in Form einer vereinfachten Schachnotation

Während des Spiels wird der jeweils letzte Zug und bei Spielende werden alle durchgeführten Züge in einer stark vereinfachten Schachnotation dargestellt. Hinter dem Zug befindet sich jeweils die Angabe der Zugzeit. Näheres dazu in der Benutzerdokumentation unter dem Punkt „Schachnotation“.

Spielwiederholung ansehen

Am Ende eines Spiels besteht die Möglichkeit, sich das gesamte Spiel noch einmal anzusehen. Klickt man auf den Button „Wiederholung ansehen“ werden die Figuren zurück in die Grundaufstellung gestellt. Anschließend wird alle zwei Sekunden ein Zug automatisch durchgeführt und zur Verdeutlichung farbig markiert. Möchte man diese Wiederholung unterbrechen, kann man auf den Button „Stopp“ klicken. Mit „Start“ wird die Wiederholung fortgesetzt.

Kurzer Überblick über die wichtigsten Klassen und Methoden sowie deren Aufgaben

Die Main-Methode befindet sich in der Klasse **SpielGUI** und erzeugt lediglich ein neues Objekt der Klasse **SpielGUI**. Die init-Methode zeigt die Eröffnungsseite an, die seitenAuswahl-Methode wechselt zwischen den verschiedenen Seiten hin und her.

Auf der Eröffnungsseite befinden sich die fünf Buttons, mit denen man zu den verschiedenen Funktionsseiten gelangt. Der **SeitenwechselListener** ist der ActionListener dieser Buttons und ruft über **SpielGUI** die entsprechende Seite auf, welche durch den Knopfdruck gewählt wurde.

Für jede einzelne Seite gibt es eine eigene Klasse im Package ***gui***. Die wichtigste und größte Klasse ist **SpielfeldGUI**.

Sie sorgt dafür, dass das Spielfeld entsprechend aufgebaut wird, dass Spiele angelegt werden und verarbeitet nahezu alle Aktionen, die während eines Spieles durchgeführt werden. Mit den Methoden spielfeldAufbau und spielfeldUIUpdate wird das Spielfeld aufgebaut bzw. aktualisiert. Dabei geht es überwiegend darum, die Figurenbilder dort anzuzeigen, wo auch die Figuren stehen. Das Spielfeld selbst besteht aus einem 8 * 8 GridLayout, welches mit Objekten der Klasse **Feld** besetzt ist. **Feld** erbt von JLabel und verfügt über x- und y-Koordinaten (von 0 bis 7), um den Algorithmik-Klassen die Position der Figuren auf dem

Spielfeld geben zu können. Des Weiteren werden grafische Hilfestellungen wie mögliche Felder und bedrohte Felder angezeigt und an vielen Stellen muss unterschieden werden, ob der zweite Spieler ein Computergegner ist oder nicht. Außerdem verwaltet diese Klasse den Timer, der direkt auf die Systemzeit zugreift und als eigener Thread implementiert ist. Die hauptsächliche Schnittstelle zwischen GUI und Algorithmik befindet sich in der spielerzugGUI- und der wennComputerDannZiehen-Methode. Dort wird die ziehen-Methode des menschlichen bzw. Computerspielers aufgerufen.
Selbstverständlich gibt es auch noch an einigen anderen Stellen Schnittstellen, welche aber nur nebensächlich sind (Auswertungsmethoden, aktueller Spieler, Anzeige von besonderen Feldern, Umwandeln von Bauern am Ende der Reihe etc.).

Das Ziehen einer Figur durch einen menschlichen Spieler wird in der Methode ziehe der Klasse **Spielfeld** durchgeführt. Dabei muss die Figur angegeben werden, die gezogen wird, sowie das Feld auf das sie gezogen wird und für die Statistik auch die Dauer dieses Zuges. Anschließend wird (nach Prüfung auf die Sonderzüge Rochade, En-Passant und Umwandlung von Bauern) die Figur von ihrem Startfeld auf das angegebene Zielfeld gezogen. Befindet sich dort eine gegnerische Figur wird diese geschlagen. Ob dieser Zug überhaupt möglich ist, wurde bereits vor Aufruf der Methode in der GUI getestet.

Dieser Test findet in den Figuren-Klassen im Package ***Figuren*** statt. Die abstrakte Klasse ***Figur*** besitzt eine Methode getKorrekteFelder, welche eine Liste mit den Feldern auf die die entsprechende Figur ziehen darf, zurückgibt. In der GUI wird vor einem Aufruf der ziehen-Methode getestet, ob das gewünschte Feld Teil dieser Liste ist. Nur wenn dem so ist, darf diese Figur auf dieses Feld gezogen werden und die Methode wird aufgerufen.

Die Berechnung der korrekten Felder passiert in zwei Schritten:

1. Ermittlung aller möglichen Felder (getMoeglicheFelder in den SubKlassen von Figur)
2. Test ob auf diese Felder gezogen werden darf, ohne dass der König danach im Schach steht.

Neben der Methode ziehe gibt es auch die Methode zugRueckgaengig in der Klasse **Spielfeld**. Dieser führt grundsätzlich die Methode ziehe nur rückwärts aus unter Beachtung einiger weniger Sonderfälle. Die Methode kann immer nur den letzten durchgeführten Zug rückgängig machen und verfügt daher über keine Parameter.

Abgesehen von den Klassen **Computerspieler** und **Gesamtdatensatz** sind die restlichen Klassen im Package ***Daten*** nur Datenklassen, die kaum erwähnenswerte Methoden haben.

Das Package ***Zuege*** stellt verschiedene Arten von Zügen bereit. Alle Klassen erben von der Hauptklasse **Zug**. Die Unterscheidung zwischen verschiedenen Zügen ist wichtig, um das Ziehen und vor allem das Rückgängig-Machen fehlerfrei gewährleisten zu können. Auch für das Laden und die Spielwiederholung sind die verschiedenen Züge existenziell.

Die Klasse **Computerspieler** erbt von **Spieler** und stellt einen Computergegner da. Sie verfügt über eine eigene ziehen-Methode welche einen bestimmen Zug ermittelt und ihn anschließend über die Spielfeld.ziehe-Methode zieht.

Die Klasse **Gesamtdatensatz** stellt Methoden für das Speichern und Laden aller benötigten Daten zur Verfügung. Beim Start des Programms wird ein neuer Gesamtdatensatz erstellt bzw. aus den Quelldateien geladen und mit dem Schließen des Programms werden alle Daten wieder gespeichert. Für den Fall dass das Programm unplanmäßig beendet wird, gibt es an einigen Stellen automatische Speicherungen.

Detaillierterer Einblick in ausgewählte Funktionen und Methoden

Speichern und Laden

Wie in der Aufgabenstellung gefordert, werden alle benötigten Daten in Textdateien gespeichert. Hierfür wurde im Programmordner ein neuer Order „settings“ erstellt. In ihm befindet sich die Textdatei mit den Einstellungen „settings.txt“ sowie den beiden Unterordnern „Spiele“ und „Spieler“. In „Spiele“ gibt es für jedes Spiel eine Textdatei (Name der Datei entspricht dem Spielnamen), in „Spieler“ gibt es für jeden Spieler eine Textdatei (Name der Datei entspricht dem Spielernamen).

Das Speichern von Spielen

In der ersten Zeile steht ein Datumsstempel mit dem Zeitpunkt der letzten Speicherung. Dies ist wichtig, wenn das Spiel geladen werden soll.
Anschließend folgen (jeweils eine eigene Zeile) die Namen der Spieler mit ihrer jeweiligen Farbe (wie im gesamten Programm true für weiß und false für schwarz), die Farbe des aktuellen Spielers, alle Einstellungen mit denen das Spiel gestartet wurde (jedes Attribut eine Zeile) und die bisher durchgeführten Züge (in Schachnotation, jeder Zug eine eigene Zeile).

Vorher wurde das Spielfeld so gespeichert und geladen, wie es aktuell aussah. Der Nachteil hieran war, dass die Dateien erheblich länger und unverständlicher waren, nach dem Laden keine Züge rückgängig zu machen waren und dass die Spielwiederholung nicht bei geladenen Spielen funktioniert hat.

Das Laden von Spielen

Beim Laden werden alle Informationen ausgelesen und temporär gespeichert. Ist die Datei vollständig gelesen, können aus den einzelnen Informationen wieder neue Objekte erstellt werden aus denen insgesamt ein neues Spiel erzeugt werden kann. Dies wird an die aufrufende GUI-Methode übergeben.

Mittlerweile wird nur noch die Schachnotation geladen und nicht mehr das komplette Spielfeld. Aus jeder Zeile der Schachnotation wird ein Zug rekonstruiert der später gezogen wird. Wurden alle bisher durchgeführten Züge aus der Grundaufstellung erneut gezogen, erhält man das gewünschte Spielfeld. Dadurch wurde auch das Zug rückgängig machen nach dem Laden und das Anzeigen des Spielvideos möglich.

Das Speichern von Spielern

Bei Spielern wird ausschließlich ihre Statistik (jedes Attribut eine Zeile) gespeichert. Der Name ist aus dem Dateinamen abzuleiten und die aktuelle Farbe wird nicht benötigt, da die Spiele nicht in chronologischer Reihenfolge geladen werden müssen.

Das Laden von Spielern

Die Daten werden ausgelesen und ein neuer Spieler wird mit einer neuen Statistik angelegt und der Spieler-Liste im Gesamtdatensatz zugefügt.

Der Computergegner

Wie in der Benutzerdokumentation beschrieben steht, gibt es vier verschiedene Computergegner von denen die drei stärksten grundsätzlich nach dem gleichen Prinzip funktionieren.

Funktionsweise und Algorithmus

Es gibt in der Schachprogrammierung einen anerkannten Algorithmus für das Programmieren eines Computergegners:
Hierfür werden mittels eines Zuggenerators alle möglichen Züge ermittelt und anschließend gezogen. Für jede neue Situation wird das Gleiche durchgeführt, bis man eine gegebene Abbruchtiefe erreicht hat (darin besteht der Unterschied zwischen den drei Gegnern). Erreicht man die Abbruchtiefe wird die entstandene Spielsituation mithilfe einer Bewertungsfunktion bewertet. Mithilfe des MiniMax-Algorithmus‘ (man geht immer davon aus, dass der entsprechende Spieler den besten Zug durchführt) ist dann die Ermittlung des besten Zuges für den entsprechenden Spieler möglich.
Um Rechenzeit zu sparen muss jedoch der MiniMax-Algorithmus optimiert werden. Dies tut die Alpha-Beta-Suche (vgl. gegebene Internetquelle im Programmcode). Hierbei wird die Untersuchung und Bewertung eines „Astes“ in dem Augenblick unterbrochen, wo klar wird, dass dieser nicht mehr in Frage kommt. Dafür werden sogenannte „Fenster“ an die einzelnen „Knoten“, die die Bewertungen verwalten und koordinieren müssen, übergeben. Fällt ein Wert in entsprechender Richtung aus diesem Fenster raus, müssen alle weiteren Äste nicht mehr betrachtet werden. Bei geschickter Implementierung entsteht eine Rechenzeitersparnis von mindestens 70% (mittels Rekursionszähler selbstständig ermittelt).

Referenzen

Zur Überprüfung der Spielstärke wurden mit allen vier Gegnern Spiele gegen das Windows – Spiel „Chess Titans“ durchgeführt.
Dabei zeigte sich, dass Karl Heinz schlechter spielt als Stufe 1 von 10. Rosalinde konnte alle Spiele gegen Stufe 1 gewinnen und eins von dreien gegen Stufe 2. Ursula gewann beide Spiele gegen Stufe 3 und gewann und verlor jeweils eins gegen Stufe 4. Walter gewann ein Spiel gegen Stufe 5, verlor eins knapp und eins deutlich. Diese Testspiele stützen unserer Meinung nach die in der Benutzerdokumentation angegebenen Vorschläge zur Wahl der Spielstufe.

Beispieldatensatz zum Testen

Das Programm verfügt bereits über einen Beispieldatensatz an Spielern und Spielen. Sollten keine Daten vorhanden sein, ist das Programm in der Lage, neue zu erstellen. Dabei werden die in der Benutzerdokumentation aufgeführten Standardeinstellungen gespeichert, sowie die vier Computergegner Karl Heinz, Rosalinde, Ursula und Walter angelegt.
Der Beispieldatensatz verfügt über die zwei menschlichen Spieler Christian und Marvin und zwei gespeicherte Spiele. Dabei wurde das Spiel „Battle“ zwischen den Spielern Christian (weiß) und Marvin (schwarz) von Marvin gespeichert, das Spiel „Testspiel“ zwischen Christian (schwarz) und dem Computergegner Ursula (weiß) von Christian.

Schwachstellen in der Programmierung und ihre Gründe

Das Programm erscheint an einigen Stellen verbesserungwürdig. Das liegt daran, dass wir aus verschiedenen Gründen nicht immer die gewünschte Implementierung durchführen konnten. Im Folgenden sollen einige uns bekannte Schwachstellen genannt und kurz erklärt werden.

Doppelte Speicherung Figuren – Felder

Den Feldern wird übergeben, welche Figur auf ihnen steht. Ebenso wird den Figuren übergeben auf welchen Feldern sie stehen. Dies erscheint auf den ersten Blick sinnlos, da bei dem Bewegen einer Figur beide Verweise geändert werden müssen. Grundsätzlich stimmt das auch, jedoch werden die Positionen der Figuren und die Belegung der Felder an unzähligen verschiedenen Stellen benötigt, sodass es ein vermeidbarer Aufwand wäre, das entsprechend weggefallene Attribut irgendwie über das andere zu erhalten. Dies ginge vor allem zu Lasten der Rechenzeit und würde den Computergegner erheblich langsamer machen, da bei der Ermittlung der Figur auf einem gegebenen Feld jedes Mal eine Iteration über alle Figuren nötig wäre, um zu testen, ob deren Feld gleich dem Feld ist, was betrachtet wird.

Eindimensionale Felder-Liste

Anstatt – wie es sich anbieten würde – ein zweidimensionales Array für die Felder zu nehmen, um anschließend über die x- und y-Koordinaten deutlich einfacher zugreifen zu können, verwenden wird eine eindimensionale Felder-Liste. Genau wie der vorherige Punkt diente das am Anfang der Rechenzeitersparnis. Zu Beginn des Projektes konnten wir noch nicht absehen, wie komplex das Programm werden wird und waren bemüht sehr rechenzeitfreundlich zu programmieren um einen starken, aber gleichzeitig schnellen Computergegner implementieren zu können. Da der Zugriff auf eine Liste schneller geht als auf ein zweidimensionales Array, entschieden wir uns für die Liste. Im Gegensatz zum vorherigen Punkt befinden wir diese Entscheidung zum jetzigen Zeitpunkt für etwas unglücklich. Es wäre vermutlich einfacher gewesen ein Array zu implementieren, jedoch ist dies jetzt unter keinen Umständen mehr zu ändern und wir halten es auch für sauberer, überall Listen zu verwenden, anstatt mal Listen und mal Arrays.

Alle Figuren haben das „bereitsGezogen“-Attribut

Obwohl nur die Figuren König, Bauer und Turm (Bauer nicht einmal verpflichtend) dieses Attribut bräuchten, besitzen alle Figuren dieses Attribut, das angibt, ob diese Figur bereits gezogen wurde oder noch an der Startposition steht. Bei König und Turm ist das verpflichtend

nötig um testen zu können, ob eine Rochade durchgeführt werden darf. Bei dem Bauer vereinfacht es die Abfrage, ob er zwei Felder nach vorne ziehen darf (erster Zug). Hätte man es da weglassen wollen, hätte man konkret über die Farbe die entsprechende Reihe abfragen müssen (2 bei weiß und 7 bei schwarz). Der Grund, warum wir uns letzten Endes trotzdem dafür entschieden haben, allen Figuren dieses Attribut zu geben, war, dass so der Quellcode erheblich übersichtlicher und einfacher wird. Es gibt nur sechs verschiedene Figuren und drei davon hätten jedes Mal gesondert behandelt werden müssen. Da wir grundsätzlich mit der abstrakten Klasse Figur arbeiten, hätte man quasi überall erst die Figur in **Koenig**, **Turm** und **Bauer** casten müssen, um das entsprechende Attribut abzufragen oder zu setzen. Dafür hätte es jedes Mal mehrere if-Bedingungen benötigt und das erschien uns im Ganzen betrachtet nicht mehr für sinnvoll. So wird dieses Attribut ganz normal und überall gleich behandelt, bloß wird es eben nicht bei allen Figuren genutzt.

GUI-Aktualisierung erst nach dem Computerzug

Die GUI wird bei einem Spiel gegen einen Computergegner erst aktualisiert, wenn dieser gezogen hat. Dies führt dazu, dass das Programm während der Bedenkzeit des Computergegners „hängt“. Deutlich wird das lediglich bei Walter, dem stärksten Computergegner, da seine Rechenzeit üblicherweise deutlich über einer Sekunde liegt. Entsprechende Aufforderungen an die GUI, sich zu aktualisieren, brachten nicht den gewünschten Erfolg und auch mehrmaliges Nachfragen in Sprechstunde und Review führten zu keinem Ergebnis. Wir nehmen an, dass Java die GUI erst aktualisiert, wenn alle Berechnungen durchgeführt sind. Da die ziehen-Methode des menschlichen Gegners in der gleichen Methode aufgerufen wird wie die ziehen-Methode des Computergegners, beendet er erst alle Berechnungen dieser Methode bevor er die Oberfläche aktualisiert um so Rechenzeit zu sparen. Dies führt zu oben genanntem Problem. Keiner der üblichen Befehle (revalidate, repaint etc.) zeigten Wirkung. Die einzige Möglichkeit dies zu umgehen bestände darin, einen eigenen Button zu kreieren, auf den zu Drücken ist, wenn der Computer ziehen sol. Dies fanden wir jedoch noch unpraktischer und entschieden uns für die gewählte Implementierung.

Hinweis: Bei der Spielwiederholung trat dieses Problem ebenfalls auf. Da gelang es uns jedoch über die Systemzeit und einen eigenen Thread das Problem zu lösen.

Bei der Größenveränderung des Spielfeldes verzerrt sich alles

Die Forderung mit Layout-Managern zu arbeiten haben wir erfüllt. Uns wurde sogar von Tutoren geraten, das gesamte Fenster nicht größenverstellbar zu machen, um nicht das Problem zu bekommen, dass das Spielfeld dadurch verzerrt wird.
Mit den verwendeten Layout-Managern war es uns nicht möglich, das Spielfeld immer quadratisch zu bekommen.
Beim Starten des Programms wird jedoch darauf geachtet, ob die Standardgröße (1200 * 800) größer ist als die Bildschirmgrenzen. Falls das so sein sollte, wird automatisch der größtmögliche Wert gesetzt. Das führt jedoch dazu, dass auf Computern mit geringer Bildschirmauflösung keine Größenveränderung möglich ist, da die Mindestgröße der maximalen Größe entspricht. Auf eine Anpassung des Seitenverhältnisses wurde verzichtet.

Aussetzen der GUI-Aktualisierung beim fehlerhaften Laden

Kann ein Spiel nicht geladen werden, wird eine Fehlermeldung angezeigt. Die GUI wird nicht zuverlässig aktualisiert, sodass es zu Anzeigefehlern kommen kann. Die Gründe dafür sind vermutlich die gleichen wie sie oben beschrieben sind.

Nicht vom Programm erstellte Dateien werden nicht gelöscht

Sollte vom Benutzer / Korrektor eine Spiel(er)-Datei selbstständig absichtlich falsch angelegt worden sein, um zu testen, wie das Programm mit fehlerhafter Eingabe umgeht, werden diese Dateien nach erfolglosen Laden-Versuch absichtlich nicht gelöscht. Das Programm muss nur von sich selbst erzeugte Dateien löschen können. Zum Schutz vor falschen Quelldateien wurden zudem alle Textdateien mit einem Schreibschutz versehen.

Fehler-Ausgaben auf der Konsole

Viele Befehle verlangen try-catch-Blöcke, da sie Fehlermeldungen werfen können. Einige davon werden vom Programm behandelt, andere nicht. Bei denen die nicht behandelt werden, wird nur der Text der Fehlermeldung auf der Konsole ausgegeben. Der Grund hierfür ist, dass es ohne absichtliche Manipulation des Programms nicht möglich ist, solche Fehler auszulösen und wir der Meinung waren, dass wir diesen Fall nicht abfangen müssen.
Alle Fehler die aus dem Programm heraus entstehen können, wie zum Beispiel fehlerhafte Eingaben und fehlerhafte Spiele werden mit entsprechenden Hinweisfenstern auf der GUI abgefangen.