Projekt Anwendungsprogrammierung: Gomoku

Beschreibung des Programms aus Sicht der Anwender

Das Programm bietet die Möglichkeit, mehrere Versionen des Spiels Gomoku bzw. 5 Gewinnt zu spielen. Dabei ist dem Spieler überlassen, ob er gegen den Computer, einen realen Gegner oder die künstliche Intelligenz gegen sich selbst spielen lassen möchte. Zusätzlich kann noch entschieden werden, wie viele in einer Reihe zum Sieg benötigt werden, ob in der Mitte des Bretts begonnen werden soll und ob nur Anlegen an einen anderen Stein erlaubt ist. Die Brettgröße ist auch flexibel wählbar.

Es sind folgende Spielregeln zu beachten

Das klassische Gomoku wird auf einem 19×19 Brett gespielt. Dabei legt jeder Spieler abwechselnd einen Stein seiner Farbe auf das Brett. Wie beim Schach startet weiß, da schwarz vom besseren Spieler gespielt wird. Die Steine dürfen nicht mehr bewegt werden, nachdem sie gesetzt wurden. Jeder versucht eine gerade, ununterbrochene Reihe aus 5 Spielsteinen zu legen. Diese Reihe kann waagerecht, senkrecht oder diagonal verlaufen. Außerdem muss der erste Zug in der Mitte des Brettes erfolgen, alle weiteren müssen angelegt werden an schon liegende Steine, also Seitlich oder Diagonal daneben.
Gleichzeitig muss jeder Spieler verhindern, dass der andere Spieler dieses Ziel vor ihm erreicht. Das Spiel ist beendet, wenn ein Spieler 5 Steine in einer ununterbrochenen Reihe gelegt hat, oder einer aufgegeben hat. Die Spielregeln für alle anderen Versionen sind analog, es ändert sich nur die Brettgröße und die Anzahl Steine in einer Reihe, die zum Sieg benötigt werden.

Bedienungsanleitung

1 Spiel Starten

Beim Starten des Programms erscheint folgende Oberfläche(siehe Abb. 1).
Startet man nun das Spiel, indem man auf den *Spiel starten* Button drückt, so werden die Standardeinstellungen geladen. Das heißt die AI spielt gegen sich selbst auf einem 19×19 Brett.

Abbildung 1: Aussehen nach Programmstart

2 Einstellungen ändern

Möchte man mit anderen Einstellungen spielen, so hat man die Möglichkeit diese zu ändern, indem man auf den *Einstellungen Tab* wechselt (Abbildung 2).
Die Anzahl der Spieler lässt sich einfach anklicken. Die gewünschte Brettgröße kann man eintippen, oder man entscheidet sich für eine der klassischen Varianten, auswählbar aus der Combo Box. Zur Auswahl stehen dabei Gomoku 19, 17 und 15 und Tic Tac Toe. Die Anzahl Steine in einer Reihe lassen sich auch einfach eintippen, die restlichen Einstellungen kann man auch beliebig an- bzw. wegklicken. Das Hintergrundbild ist auch veränderbar, zur Auswahl stehen die beiden unten abgebildetem Bilder, welche man nur anklicken braucht.
Ist man mit seinen Einstellungen zufrieden, so klickt man auf den *Spiel Starten* Button oben rechts. Dann wird automatisch zum *Spiel* Tab gewechselt und man kann mit dem Spielen beginnen.
Möchte man die Einstellungen wieder auf die Standardeinstellungen zurücksetzen, so braucht man

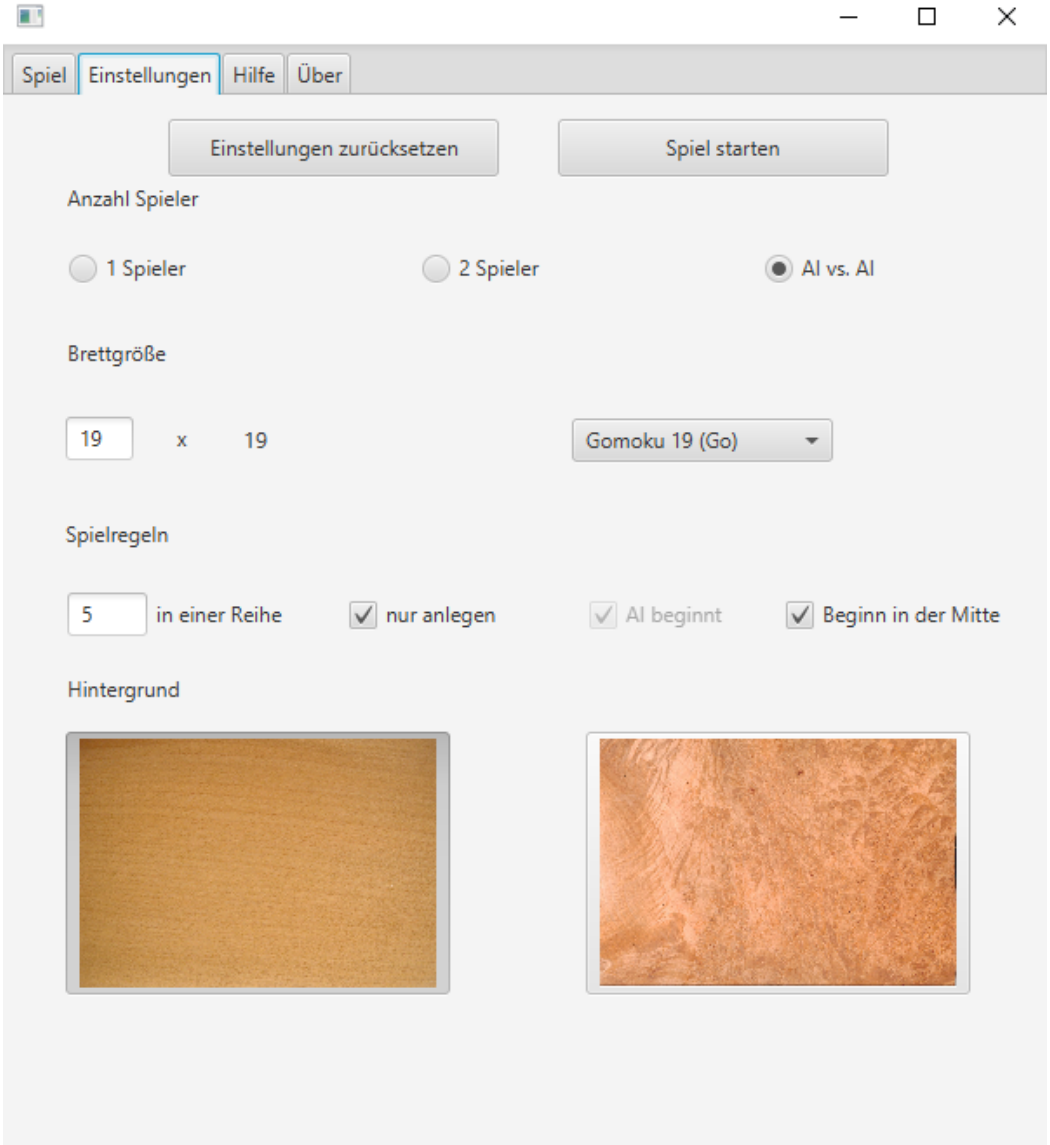


Abbildung 2: Einstellungen Tab

nur den Button *Einstellungen zurücksetzen* oben links zu klicken. Zu beachten ist auch, dass sich die Einstellungen nicht ändern lassen, solange ein aktuelles Spiel noch läuft (s. nächster Abschnitt).

3 Spiel abbrechen und neues Spiel starten

Möchte man ein aktuell laufendes Spiel abbrechen, so klicke man den Button *Neues Spiel* oben rechts (siehe Abb. 3).
Hierbei ist zu beachten, dass dann allerdings der gesamte Spielfortschritt verloren gehen würde, wobei der Spieler auch eine Warnung erhält (siehe Abb. 4).
Klickt man nun auf *Ja*, hat man wieder die Möglichkeit, die Einstellungen zu ändern oder man startet ein neues Spiel mit denselben Einstellungen.

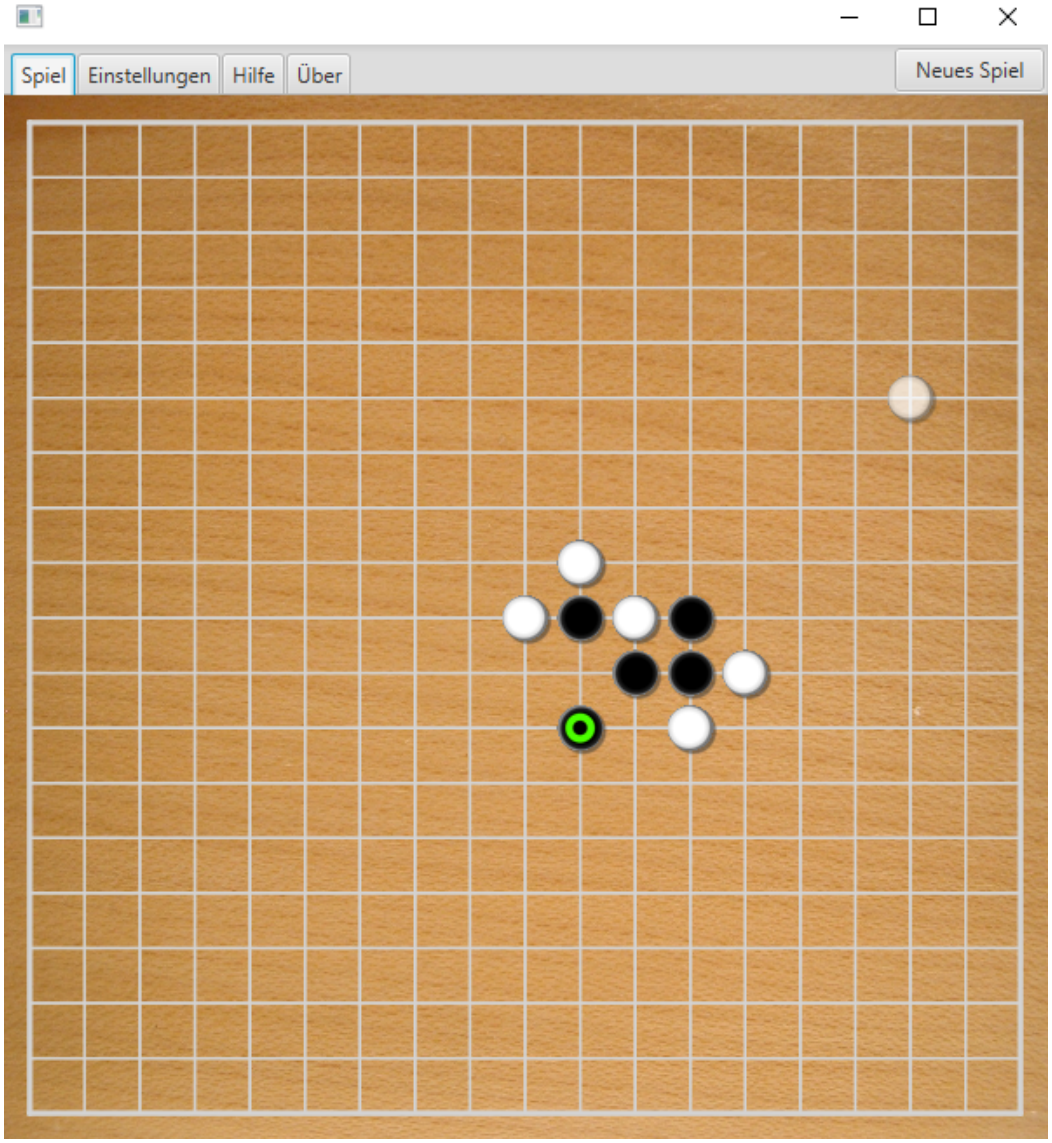


Abbildung 3: Laufendes Spiel

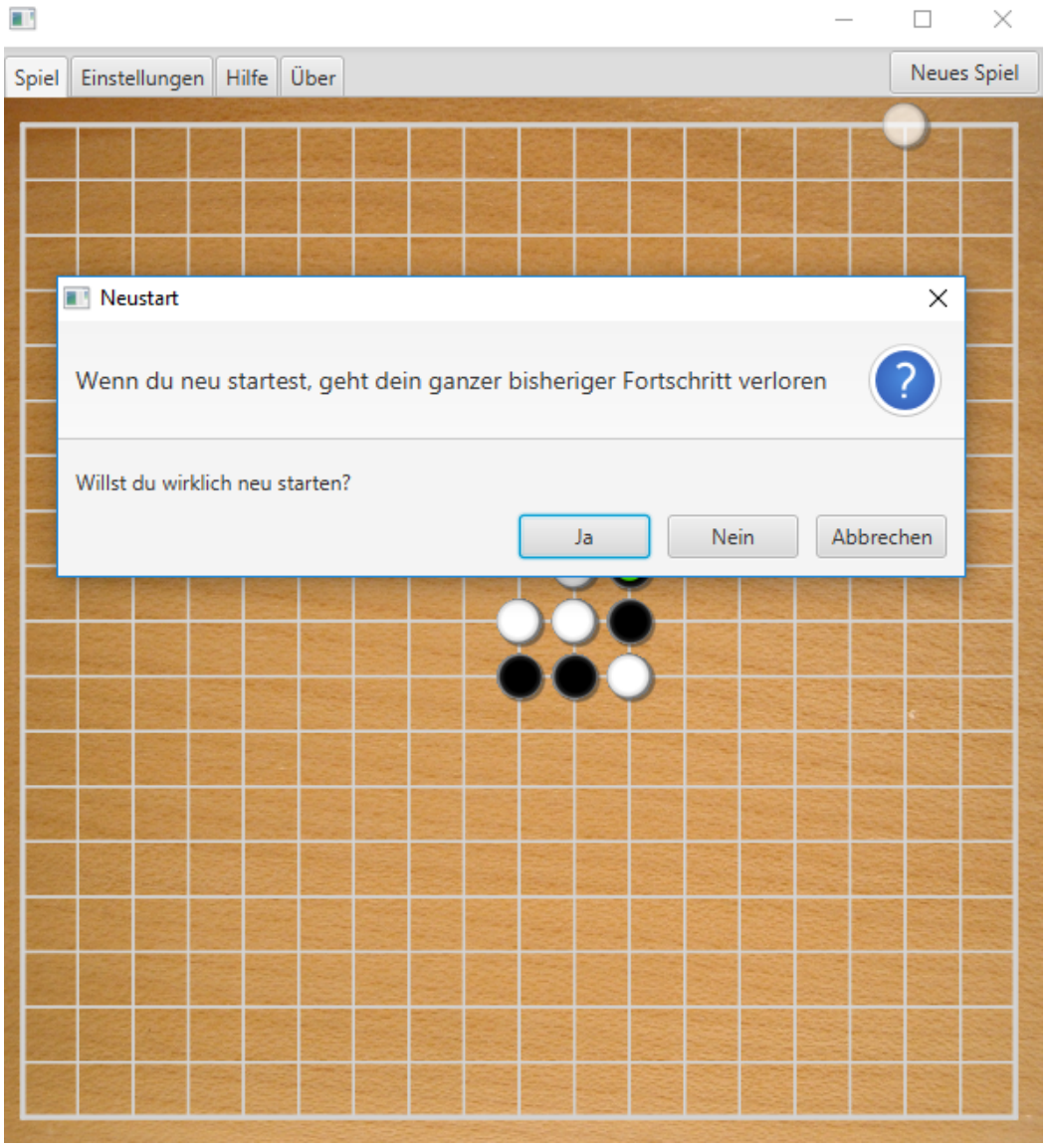


Abbildung 4: Warnung vor Neustart

4 AI gegen AI

Bei der Version AI gegen AI hat man noch zusätzliche Optionen.
Das Spiel lässt sich jederzeit pausieren, indem man auf den *Pause/Play* Button drückt (links neben *Neues Spiel*). Zudem lässt sich die Schnelligkeit der Züge beeinflussen anhand des Reglers links daneben (siehe Ab. 5).

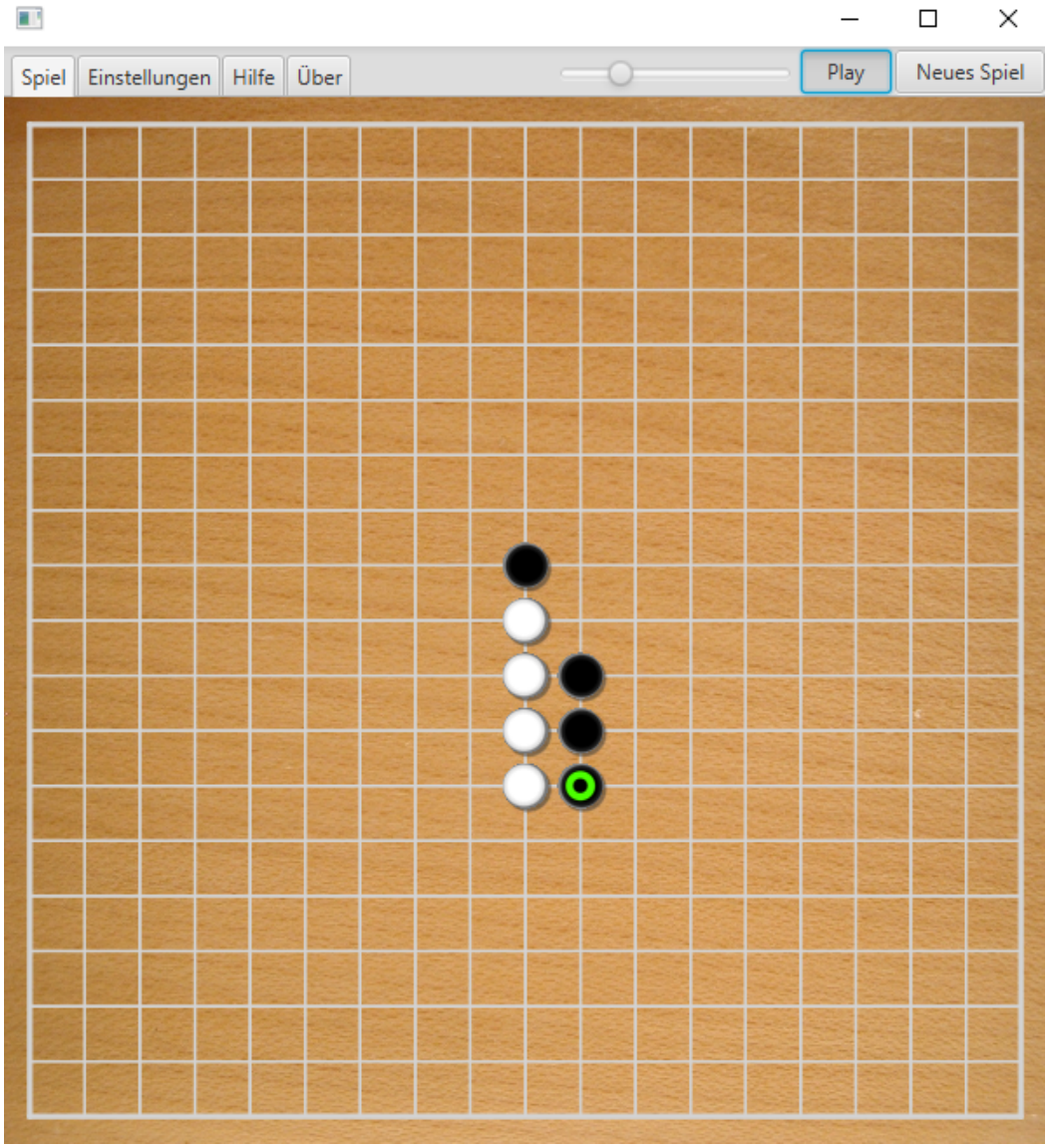


Abbildung 5: AI spielt gegen AI

5 Spielzug machen

Der Stein des jeweiligen Spielers, der gerade an der Reihe ist, erscheint an dem Mauszeiger und bewegt sich mit, wenn die Maus verschoben wird Möchte man einen Stein irgendwo platzieren, so braucht man nur auf die jeweilige Stelle im Brett zu klicken. Der Stein wird dann dort abgelegt

(solange diese Position legal ist, es also möglich ist an dieser Position zu spielen) und der Gegner ist an der Reihe.

6 Gewinner

Sollte ein Spieler gewinnen, erscheint folgende Meldung (siehe Abb. 6).

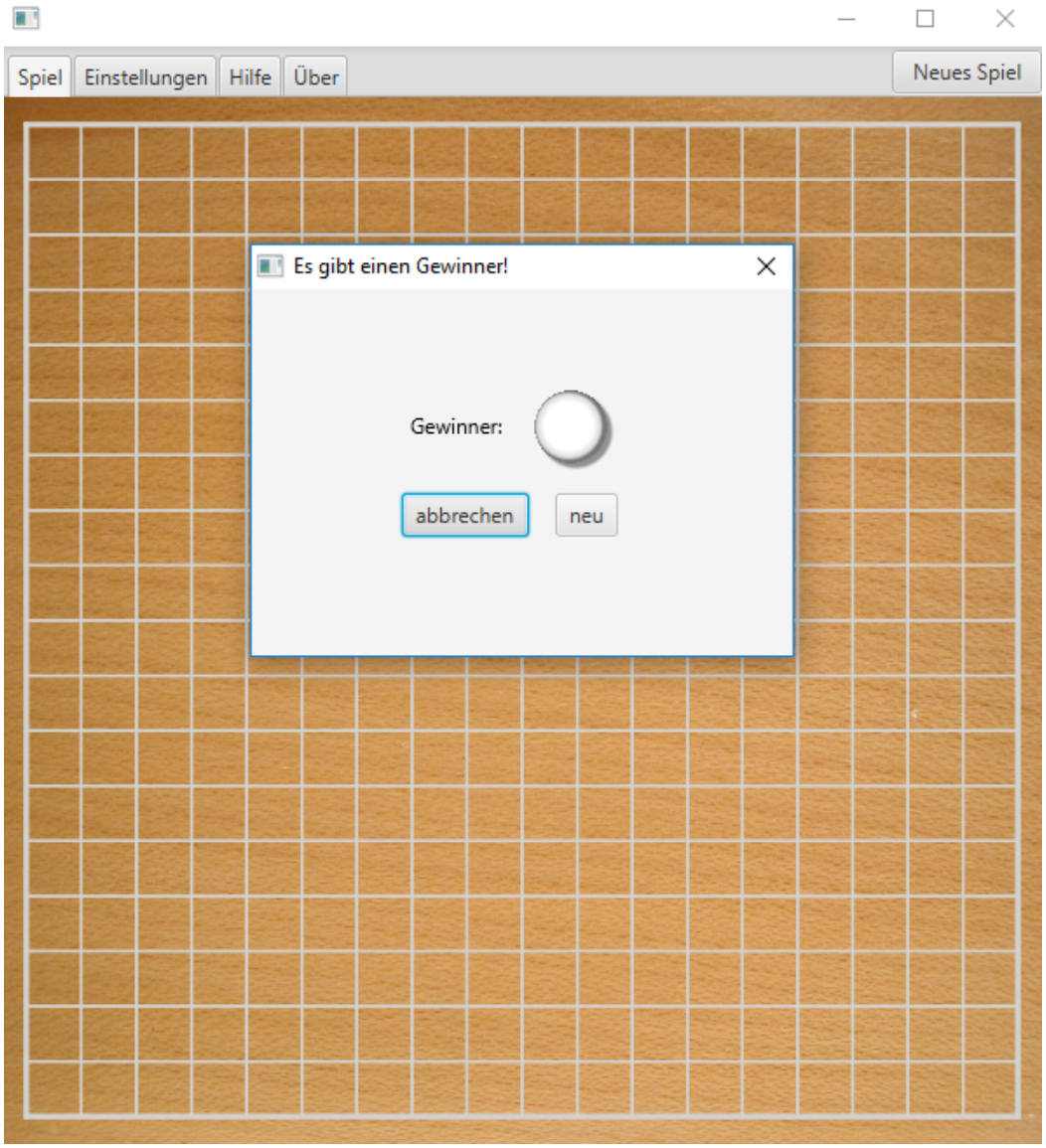


Abbildung 6: Meldung bei Gewinner

Hier hat man dann die Möglichkeit, das Spiel neu zu starten, indem man auf *neu* klickt. Oder man klickt auf *abbrechen*, um womöglich den Spielverlauf zu analysieren.

7 Hilfe und Über

Die Spielanleitung ist im Tab *Hilfe* nachzulesen und Informationen über Programm und Autor lassen sich im Tab *Über* nachschauen.

Beschreibung des Programms aus Sicht der Programmierer

Wenn das Spiel gestartet wurde sind Linien auf einem Gitter gezeichnet, worauf Bilder von Spielsteinen gezeigt werden sollen. Diese werden intern in einem `Brett` gespeichert und schon gespielte Züge von dort gelesen. Für einen Zug wird auf einen Mausklick gewartet, die Position in Gitter-Koordinaten umgerechnet und überprüft ob es legal ist dort einen Stein abzulegen. Wenn das der Fall ist, so wird der Zug gespeichert und ein Bild an der Stelle dargestellt.
Falls man gegen eine AI spielt, so wird diese sofort nach dem eigenen Zug gerufen einen Zug zu machen, was dann in eine Position resultiert wo diese Spielen möchte. Deren Zug wird dann auch ausgeführt und der Spieler ist wieder an der Reihe.
Lässt man die AI gegen sich selbst spielen, so wird ein Timer verwendet, welcher nach einer vorgegebenen Zeitspanne nach einem Zug der AI fragt und diesen ausführt. Dadurch spielt die AI nicht so schnell, wie es ihr möglich wäre, sondern meist etwas langsamer sodass man ihr folgen kann.
Die AI verlässt sich auf ihre Heuristik-Funktion (), welche eine Bewertung jeder legalen Position des momentanen Brettes ausgibt. Diese Information wird dann iteriert und eine Liste an möglichen besten Zügen wird zurückgegeben, wovon dann im `SpielController` einer zufällig gewählt und gespielt wird.

8 Model

Im Model befinden sich die folgenden Klassen:

`Brett`	– zur Speicherung eines Spielbrettes
`Options`	– speichert die Optionen als ein Paar zweier Objekte vom Typ `String` und `Object`
`SpielAI`	– generiert die AI
`SpielStein`	– zur Speicherung eines Spielsteins

Brett
− _dim : int − _spieler : int − _brett : SpielStein[ ][ ] − _gitterVert : List<Line> − _gitterHorz : List<Line> − _gitter : List<Line> − _SpielZuege : List<SpielZug> − _gitterWeite : double − _randX : double − _randY : double − _CheckAdjacent : boolean
+ Brett(dim : int, x : double, y : double) + redrawGitter(x : double, y : double) : void + roundCoord(x : double, y : double) : double[ ] + steinAt(int x, int y) : SpielStein + steinAt(double x, double y) : SpielStein − steinSet(int x, int y, SpielStein s) : boolean + makeMove(SpielZug zug) : boolean + printMoves() : void + getNextMoveColour() : int + List<SpielZug> getSpielZuege() : final + getDim() : int + getGitter() : List<Line> + getGitterWeite() : double + getRandX() : double + getRandY() : double + getSpieler() : final int + getBrett() : final SpielStein[ ][ ]

+ static Brett::SpielZug
+ x, y : int + stein:SpielStein + iView:ImageView
+ SpielZug(x:int, y:int, stein:SpielStein, iView:ImageView) + toString():String

Options
− _menge : HashSet<Tupel>
+ Options() + setOption(name : String , objekt : Object) : void + getOption(name : String) : Object + printOption(name : String) : void + toString() : String

−Options::Tupel
+ name : String + objekt : Object
+ Tupel(name : String, objekt : Object) + hashCode() : int + equals(Object obj) : boolean − getOuterType() : Options + toString() : String

SpielAI
− _brett : Brett − _possibleMoves : ArrayList<LinkedHashSet<Savegame>>
+ SpielAI(brett : Brett) + generateNextMoves() : void + updateMoves() : void + getBestMoves() : Integer[ ][ ] + addDoubleArray(a : Double[ ][ ], b : Double[ ][ ]) : static Double[ ][ ] + multDoubleArray(a : Double[ ][ ], f : double) : static void + twoDeepCloneDouble(a : Double[ ][ ]) : static Double[ ][ ] + printDoubleArray(a : Double[ ][ ]) : static void

SpielAI::Savegame
− moveNr : int − steine : int[ ][ ] − spielerAnz : int − dim : int − nextMove : int[ ]
+ Savegame(brett : Brett) + generateNextMoves() : LinkedHashSet<Savegame> + generateHeuristic() : Double[ ][ ] + hashCode() : int + toString() : String + equals(obj : Object) : boolean

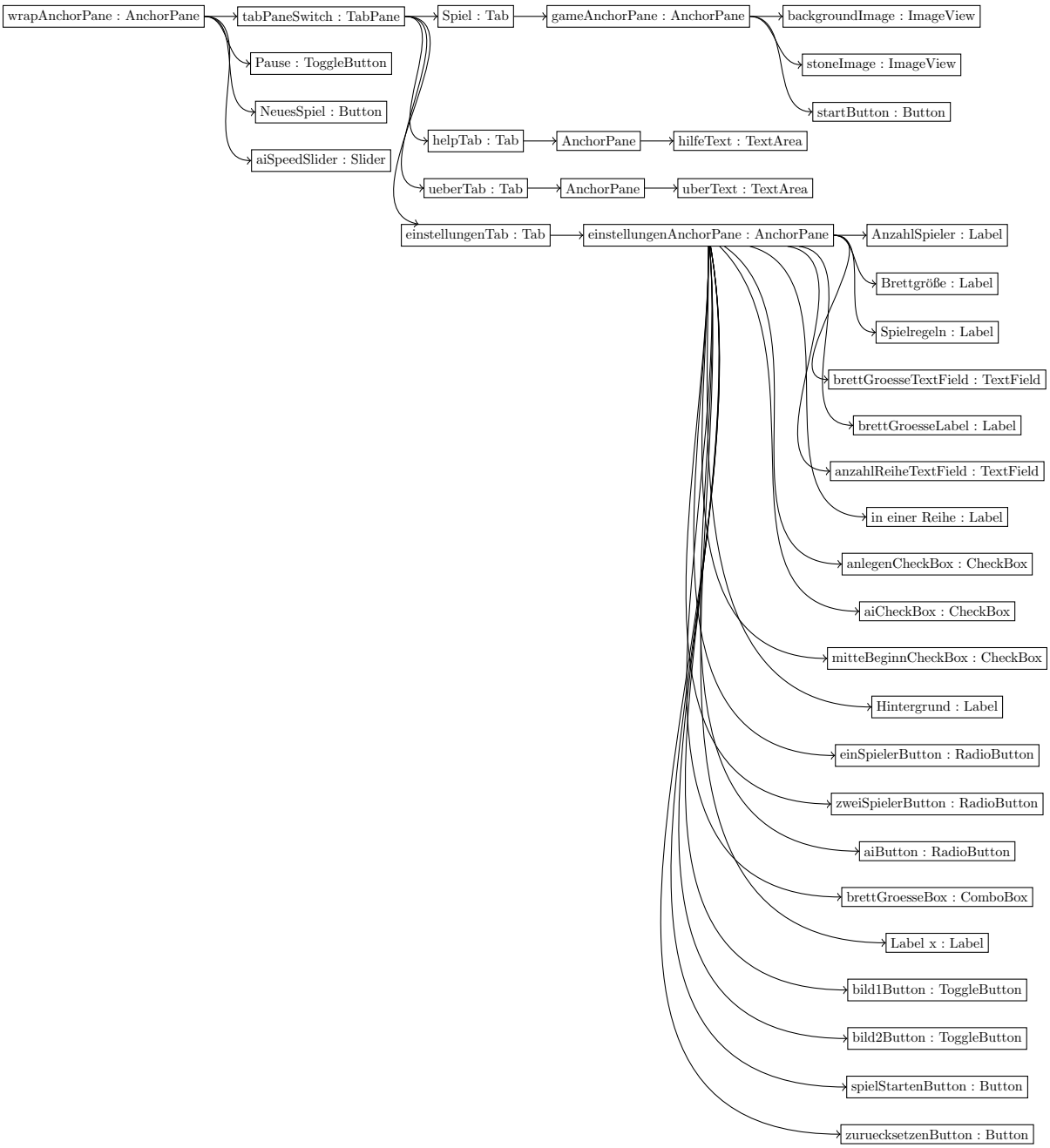
SpielStein
− farbe : int − img : Image − blackStone : static Image − whiteStone : static Image
+ getColor() : final int + getImage() : final Image

Die AI sucht sich den Besten Zug folgendermaßen mit der Funktion `getBestMoves()`aus:

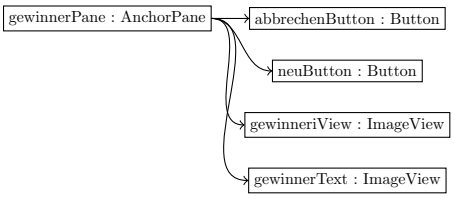
```
1:  h ← generateHeuristic()
2:  max ← −∞
3:  n ← 0
4:  for i ∈ {0, dim(h)} do
5:      for j ∈ {0, dim(h)} do
6:          if max < h_{i,j} then
7:              max ← h_{i,j}
8:              n ← 1
9:          else if max = h_{i,j} then
10:             n + +
11:         end if
12:     end for
13: end for
14: erg :∈ ℕ^{n×2}
15: for i ∈ {0, dim(h)} do
16:     for j ∈ {0, dim(h)} do
17:         if  max = h_{i,j}  then
18:             n ← n − 1
19:             erg_{n,0} ← i
20:             erg_{n,1} ← j
21:         end if
22:     end for
23: end for
24: return erg
```

9 View

Folgendes ist der Scene-Graph des `GameLayout`:



Folgendes ist der Scene-Graph des `GewinnerLayout`:



10 Controller

Der Spielcontroller ist zuständig für das gesamte Spiellayout und den zeitlichen Ablauf des Spieles. Zusätzlich gibt es noch den `GewinnerController` für das `GewinnerLayout`, welches immer dann erscheint, sobald ein Spieler gewonnen hat bzw. unentschieden gespielt wurde.

GewinnerController
− gewinnerPane : AnchorPane − abbrechenButton : Button − neuButton : Button − gewinneriView :ImageView − gewinnerText : Text − spielController : SpielController − gewinnerStage : Stage
+ handleAbbrechenButton(event : ActionEvent) : void + handleNeuButton(event : ActionEvent) : void + setGewinnerImage(image Image) : void + setDialogStage(gewinnerStage : Stage) : void + setGewinnerText(s : String) : void + setDialogSpielController(spielController : SpielController):void

Main
+ optionen : static Options + primaryStage : static Stage
+ start(primaryStage : Stage) : void + main(args : String[]) : static void

SpielController
− mitteBeginnCheckBox : CheckBox − backgroundImage : ImageView − tabPaneSwitch : TabPane − ueberTab : Tab − anlegenCheckBox : CheckBox − brettGroesseLabel : Label − einstellungenAnchorPane : AnchorPane − gameAnchorPane : AnchorPane − aiCheckBox : CheckBox − helpTab : Tab − gameTab : Tab − brettGroesseTextField : TextField − zweiSpielerButton : RadioButton − stoneImage : ImageView − brettGroesseBox : ComboBox<String> − bild2Button : ToggleButton − einstellungenTab : Tab − anzahlReiheTextField : TextField − bild1Button : ToggleButton − einSpielerButton : RadioButton − aiButton : RadioButton − hilfeText : TextArea − uberText : TextArea − neuButton : Button − spielStartenButton : Button − startButton : Button − newGameButton : Button − pauseGameButton : ToggleButton − zuruecksetzenButton : Button − wrapAnchorPane : AnchorPane − aiSpeedSlider : Slider − radioButtonGroup : final ToggleGroup − bildGroup : final ToggleGroup − choiceBoxOptions : ObservableList<String> − spielbrett : Brett − gameDone : boolean − s : SpielStein − currWidth, currHeight : double − lastPlayed : ImageView − winningStone : List<ImageView> − gegner : SpielAI − lastTime : static long − aiPaused : boolean − zweiAiTimer : AnimationTimer
− initialize() : void − standardEinstellungen() : void + bildeBrett() : void − handleSpielerAnzahlButton(ActionEvent event) : void − handleBrettGroesseBox(ActionEvent event) : void − handleBrettGroesseFeld(ActionEvent event) : void − handleSpielregeln(ActionEvent event) : void − handleBackground(ActionEvent event) : void + neustart() : void − handleSpielStartenButton() : void − handleZuruecksetzenButton(ActionEvent event) : void − handleStartButton() : void − disable() : void − enable() : void − handleNewGameButton() : void − handlePauseGameButton(ActionEvent event) : void − handleMouseMoved(MouseEvent event) : void − handleSizeChanged() : void − handleSizeChanged(boolean forceIt) : void − updatePlayMarkers() : void − handleDragDetected(MouseEvent event) : void − handleMouseClicked(int x, int y) : void − handleMouseClicked(MouseEvent event) : void − letAImakeMove() : void − handleGewinner() : boolean − handleGewinner(boolean unentschieden) : boolean − checkIfGewinner() : boolean − handleKeyPressed(KeyEvent event) : void − handleKeyReleased(KeyEvent event) : void