Module	Parameter	Lokale Variablen	Funktion
FindMinMax	Pointlist	Imin, imax, jmin, jmax, iSplits, jSplits, iDistance, jDistance	• Die minimas und maximas der i und j-Werte der Koordinaten werden gesucht, um den „Rahmen“ des Gitters um das Schachbrett festzulegen • In den ConstantArrays JSplits und ISplits werden die Zellen des Gitters gespeichert. Diese werden über die Distanz der jeweilien Minimalwerte und Maximalwerte geteilt durch die gewünschte Anzahl an Zellen geteilt.
SortPointList	iSplits, jSplits, Pointlist	pi,pj	• Die Eckpunkte werden zunächst der Größe nach nach ihren i-Werten Sortiert. Die sortierte Liste wird dann durchgezählt, so dass jeder Punkt seinen Indexwert in I-Richtung bekommt • Danach werden die Eckpunkte der Größe nach nach ihren J-Werten sortiert und bekommen hier ebenfalls einen Index zugeordnet • (Diese Sortierung ist nach jetzigem Stand des Algorithmus vllt nicht mehr zwingend notwendig)
GoThroughConvexHulls	iSplits, jSplits, pj	ConvexHull	• Nun wird herausgefiltert, welcher Punkt in welche Zelle des erstelllten Gitters gehört, somit wird eine grobe Vorsortierung der Punkte für den weiteren Verlauf vorgenommen. • In einer For-Schleife welche alle iSplits durchzählt wird die Funktion FindPointsInConvexHull bei jedem Durchgang aufgerufen welche eine Liste mit Associations in die Liste ConvexHull hinzufügt. • Der Funktion werden die momentanen iSplits der Durchzählung übergeben und alle Jsplits. Des Weiteren wird die nach J sortierte Punkteliste übergeben
FindPointsInConvexHull	iSplits[[1,ii]], iSplits[[1,ii+1]], jSplits, pj	ConvexHullCell={}, ConvexHullList,ConvexHullCellKeys = <||>	• Eine Liste namens ConvexHullCell und eine Association nachmes ConvexHullKeys wird angelegt • Zwei For-Schleifen werden gestartet. Die erste läuft durch alle Jsplits, die zweite geht alle Punkte von pj durch. • Innerhalb der For-Schleife wird dann überprüft, welche Punkte aus pj sich innerhalb der übergebenen Isplits und den dazugehörigen jSplitts befinden. • Die Koordinaten, die Indizes und die Zellenbezeichnung werden dann in Keys in die Association ConvexHullCellKeys gespeichert und er Liste ConvexHullCell angehängt. Diese Liste wird dann and die Liste ConvexHull angehängt Wiederholung des Vorganges mit neuen iSplits.

FindStartVectors	ConvexHull	StartPointCloud={},
		StartPointCloudKeys=<‖>,
		VecI,VecJ,countI,countJ,
		Start, nextI,nextJ,
			• DiePunkte der Zellen (i = 1,j = All) und (i = all, j = 1) werden in eine neue Liste namens StartPointCloud gespeichert.
			• Die Liste wird zweimal durchlaufen
			  • Alle Punkte welche sich in den Zellen j = 1 und i = all aufhalten. Aus ihnen wird der geringste i- Wert ermittelt (VecI)
			  • Alle Punkte welche sich in den Zellen j = all und i = 1 aufhalten. Aus ihnen wird der geringste j- Wert ermittelt (VecJ)
			• Die Punkte mit den geringsten Werten werden in VecI und VecJ gespeichert.
			• Jetzt wird die Liste nochmals zweimal durchgegangen.
			  • Alle Punkte welche sich in den Zellen j = 1 und i = all aufhalten werden durchgegangen. Aus ihnen wird derjenige Wert ermittelt, welcher einen Wert für J besitzt der kleiner ist als der momentan j- Wert von VecI und dessen i-Wert kleiner ist als der i-Wert von VecI plus einem Offset. Dieser Wert ist das neue VecI
			  • Alle Punkte welche sich in den Zellen j = all und i = 1 aufhalten. Aus ihnen wird derjenige Wert ermittelt, welcher einen Wert für i besitzt der kleiner ist als der momentan i- Wert von VecJ und dessen j-Wert kleiner ist als der j-Wert von VecI plus einem Offset. Dieser Wert ist das neue VecJ
			• VecI und VecJ ergeben den gleichen Punkt und somit ist der Startwert gesetzt.
			• Nun sollen die ersten Punkte in i- und j-Richtung vom Startpunkt aus gefunden werden.
			• Es wird ein nexti und ein nextj definiert, dessen koordinaten sehr groß anfangen
			• Es wird wieder die StartPointListe zweimal durchlaufen
			  • Es werden Punkte gesucht, welche sich in der selben Zelle i wie der Startpunkt befinden und auch die Zellen +1 und -1 drum herum. Sollte es ein Punkt geben, der kleiner ist als das momentane nexti und größer als der Startpunkt, jedoch nicht gleich dem Startpunkt. So nimmt nexti dessen Wert an.
			  • Danach muss geprüft werden, ob das potentielle nexti auch wirklich das richtige nexti ist. Hierzu wird eine neue For-Schleife gestartet, welche wieder die StartPointCloud durchgeht und überprüft ob es einen Punkt gibt dessen j-Koordinatenabstand zum Startpunkt kleiner ist also der j-Koordinatenabstand des momentanen nexti zum Startpunkt und ob dessen i-Koordinatenabstand zum Startpunkt kleiner ist als der momentane i-Koordinatenabstand von nexti zum Startpunkt.

			• Ist dies der Fall so wird dieser Punkt zum neuen nexti. • Mit dem potentiellen nextj wird ebenso verfahren.
CreatePossiblePoint – ListsIAndJ	nextI, nextJ, Start, ConvexHull	IList={}, JList={}, IDir, JDir, distance, cache, PotNextI, PotNextJ	• IDir und JDir sind die Richtungsvekoren vom Startpunkt aus in beide Kantenrichtungen des Schachbretts. • Danach werden die ersten beiden Spalten in I- und J-Richtung jeweils durchlaufen, und in iList und JList gespeichert. • Diese Listen enthalten weitere potentiellen Punkte entlang der gesuchten Kante. • Die Kanten können natürlich durch die perspektivische verzerrung mancher Bilder auch noch weiter in die Zellen hineinragen. Hierum kümmert sich dann im späteren Algorithmus die SaftyJList[] und SaftyIList[] Funktionen
FindNeighbours	IList, JList ,Start, nextI, nextJ, ConvexHull	SortedPointsKeys = <||>, Sortedpoints = {}, proportionJ, proportionI, Jtemp, itemp, PotNextJDir, distanceNextPotPointJ, PotNextIDir, dinstanceNextPotPointI, NeighbourNumberJ, NeigbournumberI, distanceJ, distanceI, NextJDir, NextIDir, StartPropJForFirstCompleteGridJ	• StartPoint und NextPointI und NextPointJ werden die Keys NeighbourI und NeighbourJ gegeben mit StartPoint(NeighbourJ → 1, NeighbourI → 1), NextPointI(NeighbourJ → 1, NeighbourI → 2) und NextPointJ(NeighbourJ → 2, NeighbourI → 1). • Diese drei bereits bekannten Punkte werden dann auch in eine angelegte CheckPointListe gespeichert, diese wird für das spätere Prüfen von weiteren Punkten benötigt. • Nun wird zunächst in einer For-Schleife die Punkte von StartPoint und NextPointJ aus gesucht. • Anmerkung: Für die Punkte in I-Richtung des Schachbretts wird das selbe Verfahren angewandt. • Benötigt wird die Distanz zwischen dem momentanen StartPointJ, welcher nach jedem Durchlauf der Schleife den wert des momentanen NextPointJ bekommt und einem momentanen NextPointJ, welcher nach jedem Durchlauf der Schleife den wert des gereade neu gefundenen nächsten Punktes bekommt.. • Die Schleife selbst durchläuft alle Punkte, welche in der für die Richtung entsprechenden Richtung Liste sind. In diesem Fall die JList • Es wird außerdem bei der Suche den nächsten Punktes in j-Richtung eine Distanz namens proportion berechnet, welcher die Distanz i zwischen StartPoint und Nexpoint beinhaltet. • Innerhalb der durchlaufenden Liste wird derjenige Punkt gesucht welcher zum NextPointJ den geringesten Abstand in J Richtung hat und dessen Abstand in I-Richtung <= der i-Koordinate des NextPointJ + proportion+noch einen Puffer ist und >= der i-Koordinate des NextPointJ – proportion+noch einen Puffer.

			• Ist der nächste Punkt gefunden, so wird dieser der SortedPointsList und der der CheckPointsList übergeben mit den passenendem NeighbourI und NeighbourJ associationKey. • Des Weiteren bekommt für den nächsten Schleifendurchlauf StartPointJ die Werte von NextPointJ und NextPointJ in wird der neu gefundenen Punkt aus der JListe gespeichert. • Im Anschluss werden noch in AppendTo[SortedPoints,SaftyListJ[Start,CheckPointJ,proportionY,CheckCellForJ, ConvexHull,distanceJ]]; AppendTo[SortedPoints,CompleteJGrid[nextI,ConvexHull,StartDistanceJ,StartProportionJ,Start ,Jp,aI]] Weitere Punkte zur SortedList in J-Richtung hinzugefügt, bei ersterem nur in bestimmten Fällen. Mehr zu den Funktionen folgt. • Nicht zu vergessen: selbiges wie Oben wird auch mit den Punkten in I-Richtung vollzogen, bis auf die CompleteGrid Funktion
SaftyList	Start, CheckLastPointJ, proportionJ, LastJPointsCell, ConvexHull, NextJDir	SaftyList = {}, SaftsKeys = <||>, SaftyKeysList = {}, propJ, lastDir, lastdistanceJ	• Der Funktion werden die Parameter CheckLastPointJ und LastJPointCell mitgegeben. Diese stammen aus der Funktion FindNeighbours und es handelt sich um den letzten Punkt der innerhalb der JListe ermittelt wurde und dessen i-Zelle in welcher sich dieser befindet. • Da die I- bzw die JListe in jede Richtung nur die Punkte der ersten beiden Zellen beinhaltet, kann es bei einem rotierten Schachbrett sein, dass sich noch weitere Punkte in Zellen weiter oben/unten befinden • Die Funktion SaftyList, erstellt eine Liste aus möglichen weiteren Punkten, indem sie die in diesem Falle I-Zelle des letzten Punktes nimmt und diese so wie die unter und oberhalb dieser Zelle und alle deren J-Zellen aufwärts auf einen möglichen nächsten Punkt untersucht. → Dies geschieht nach dem selben Verfahren wie in FindNeighbours. • Sollte es noch einen geben wird dieser ebenfalls der CheckPointList und der SortedPointsList zugewiesen, ansonsten passiert nichts.
CompleteJGrid	StartPointI, ConvexHull, StartDistanceJ, proportionJ, Start,	PossiblePointsList = {}, SortedPointsKeys = <||>, SaftyPossiblePointsListJ = {}, propJ, StartPointForJGrid, distanceJ,	• Nachdem die äußersten Punkte der linken und unteren Kante des Schachbretts gefunden wurden, muss nun das restliche Grid des Schachbretts detektiert und mit den richtigen NeighbourI und NeighbourJ werten versehen werden. • Jeder Punkt der in I-Richtung als „Rahmenpunkt“ detektiert wurde, wird einmal als Startpunkt gesetzt, von ihm aus wird dann in einem sehr ähnlichem Verfahren wie schon zuvor der nächste Punkt in J-Richtung gesucht und wenn nötig tritt auch hier

	NeighbourNumberJ, aI	NextNeighbourNumberJ, distanceNextPotGridPointJ, tempj, NextPointJDir, NextJDir, CheckPointJ, CheckCellForJ	nochmal die SaftyList Funktion in kraft um auch wirklich alle Punkte jeder Reihe ausfindig zu machen