3.3 Implementation des Q-Players:

Die verwendete Durchführungsweise des Q-Learnings an der die Implementierung sich orientiert ist die folgende:

Dabei ist alpha der Prozentsatz für den zudem der alte Wert aus der Datenbank bestehen bleiben soll. Es wurde ein Wert von 0.95 gewählt, sodass eine 5 % Veränderung möglich ist, sollte der Q-Player, denselben Zustand noch einmal betreten und den Wert verändern wollen. Das verhindert, dass der Wert jedes Mal überschrieben wird, sobald er neu berechnet wird. Stattdessen konvergiert mit steigender Lernzeit gegen einen festen Wert. Gamma ist der Lernparameter mit dem gewichtet wird, wie stark zukünftige Züge in die Bewertung einfließen. Es wurde ein Wert von 0.8 gewählt. Die Belohnung wird verteilt, sobald das Spiel gewonnen ist.

Um dieses Verfahren an das Vier-gewinnt Spiel anzupassen, wurden mehrere Veränderungen gemacht. Es wurde nicht nur eine Belohnung für ein gewonnenes Spiel, sondern auch eine Bestrafung für ein verlorenes Spiel gesetzt. Diese werden immer dann verteilt, wenn ein Spiel abgeschlossen ist. Der Q-Player wird über die Methode reactToWinOrLose(..) darüber informiert und schreibt den neuen Wert in die Datenbank. Dies passiert getrennt von der Berechnung der restlichen Formel, da immer dann wenn ein Spiel abgeschlossen ist die Berechnung von nicht mehr nötig ist, das es keine zukünftigen Züge gibt.

Die Formel wurde für das 2-Spieler Spiel angepasst. Im ursprünglichen Q-Learning wird ein 1-Spieler Spiel betrachtet, sodass der Spieler nach einem Zug direkt wieder am Zug ist und daher direkt die nächsten Züge aus der Datenbank gelesen werden können. Bei Vier-Gewinnt muss erst der gegnerische Spieler seinen Stein platzieren bevor der Q-Player selbst wieder an der Reihe ist. Die Berechnung erfolgt in der Methode *turn()* und wurde wie folgt angepasst:

wird in der Methode *avgValueForNextStateAllActions(..)* berechnet über eine Simulation der nächsten möglichen Züge. Zunächst werden alle möglichen Züge für den Q-Player generiert. Das Generieren von Zügen wird ermöglicht durch die Methode *generateActions(..)*. Basierend auf diesen Zügen werden die die nächsten Züge für den Gegner berechnet. Es wird nur eine Heuristik verwendet um einen dieser Züge auszuwählen. Dafür wird der Spielzustand invertiert und in der Datenbank des Q-Players geschaut, ob dieser Züge für diesen Spielzustand vorhanden sind. Falls ja, dann wird derjenige ausgewählt mit dem höchsten Wert. Damit wird nur die bestmögliche Aktion für den Gegner betrachtet, was sinnvoll erscheint, da dieser immer diese Aktion machen würde und nicht eine beliebige andere. Falls nein, wird ein zufälliger Zustand ausgewählt. Anschließend werden für diese ausgewählte Aktion des Gegners alle möglichen Züge des Q-Players generiert. Es wird der Durchschnitt der Werte all dieser Aktionen gebildet und in gespeichert. Der Durchschnitt wird aus dem Grund gebildet, dass ansonsten die Bestrafungen nicht mit in den Q-Wert einbezogen werden.

