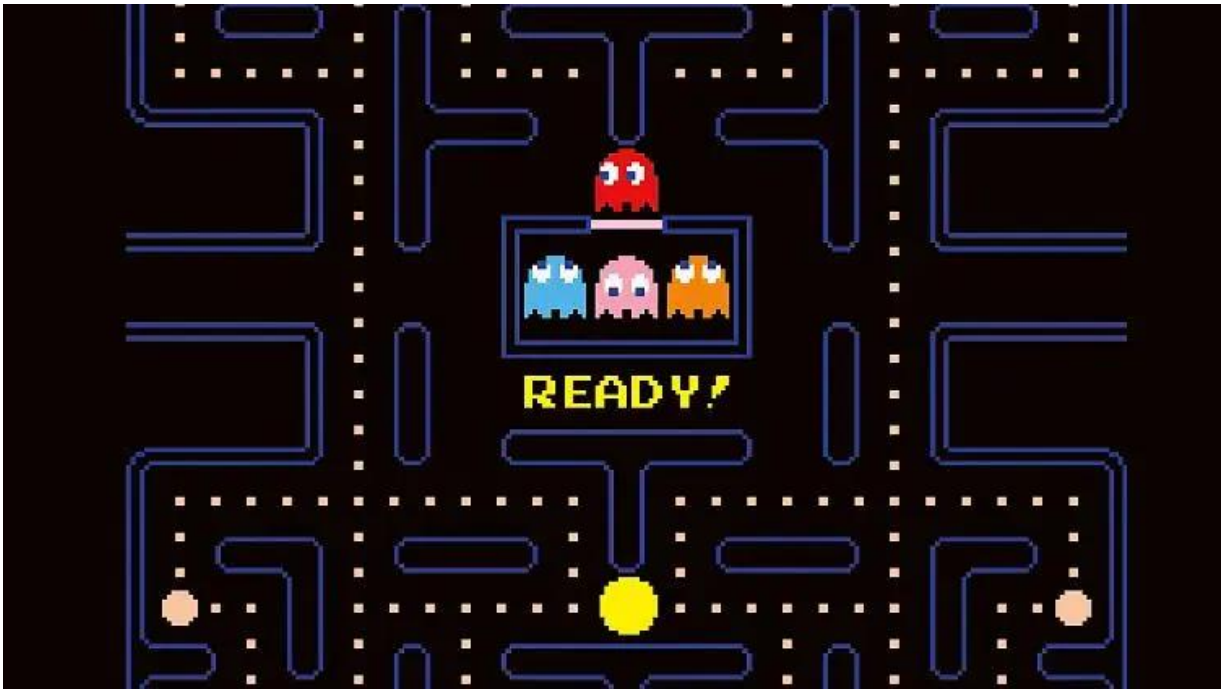
Einleitung:
Der grundlegende Algorithmus von Pacman basiert stark auf den Prinzipien der Markov Decision Processes (MDPs). Im Gegensatz zu MDPs ist unser Pacman jedoch ein determinitisches Problem, wenn man von den Bewegungen der Geister absieht. Dies vereinfacht den Algorithmus erheblich, da die Bewegung des Agenten deterministisch ist. Wir betrachten also die deterministische Bewegung des Agenten. D.h. nicht im klassischen Sinne eines MDPs, wie z.B. das Abweichen von der getroffenen Wahl des Agenten. Die Güte einer Situation und die Wahl der nächsten Aktion hängen von Zahlenwerten (Utilities) ab, die zu Beginn des Vorgangs vorgegeben werden. Danach werden die Werte iterativ in eine Map verteilt.



Mapping (MDP_Mapper & ValueManager):
Pro Spielfeld wird von einem Mapper ein Zahlenwert definiert und in eine Map gespeichert. Diese Werte können entweder konstant sein oder werden dynamisch berechnet werden.
Ein Value Manager wird verwendet, um die einzelnen Werte der Map zu berechnen. Zu diesem Zweck verwendet der Value Manager „logische Felder“, die ein Feld in der Map repräsentieren. Diese Felder (en. fields) sind mit eine Kostensuche ausgestattet, die spezielle Berechnungen für den Value-Manager durchführt, wie z.B.: Wie weit ist der Agent bzgl. der Powerpill vom nächsten Geist entfernt? Befinde ich mich in einem Respawn-Feld, das gerade aktiv ist? Solche Fragen werden von unseren logischen Feldern beantwortet.
Auf der nächsten Seite finden Sie eine Tabelle, die die Zuordnung der Startwerte beschreibt.

PacmanTileType:	Berechnung:
WALL, PACMAN	Große negative Konstanten als Pseudowerte, die überall ignoriert werden
POWERPILL	Zunächst hat die Powerpill den Wert 0. Wenn der Agent 2 Schritte von der Powerpill entfernt ist, so prüft der Agent, ob Geister im Umkreis von 5 Schritten vorhanden sind. Ist dies der Fall? Dann wird ein Wert gewählt, der auf dem stärksten Typ der umgebenden Geister basiert: {Hunter: 4.5, Eager: 4.0, Random: 3.5 } Außerdem wird berücksichtigt, ob der Geist in der Nähe, unter Effekt einer vorherigen Powepill noch steht, dann wäre die Einnahme einer Powerpill Verschwendung im Sinne der Auflösung von Sackgassen oder kritischen Situationen.
EMPTY	Zunächst hat der Dot den Wert 0. Befindet sich der Dot an eine Respawn-Stelle, die in Kürze aktiviert wird, dann wird der Wert -5 zugewiesen. (Dieser Wert ersetzt später den iterativ berechneten Wert des Feldes, um es dem Agenten unmöglich zu machen, in die Falle zu geraten. Leere Felder an den Enden von Sackgassen erhalten die Länge der Sackgasse als negativer Wert, um zu verhindern, dass der Agent in der Sackgasse stecken bleibt.
DOT	Wie bei den leeren Feldern wird zuerst auf Respawn-Stellen geprüft und wenn diese aktiv sind, wird der Wert -5 vergeben. Fall: Ein Dot befindet sich in einer Sackgasse: Der Agent prüft, ob diese Sackgasse alle verbleibenden Dots hat, dann wird der Agent mutig und geht schnell in die Sackgasse hinein, da die Felder den Wert 10 haben. Heuristik: Wenn die Anzahl der Schritte des Agenten mit Powerpill-Effekt größer ist als die Länge der Sackgasse, so schafft es der Agent, diese Sackgasse leer zu räumen, z.B. indem der Hunter-Geist vom Pacman wegläuft. Andernfalls wird geprüft, ob die Sackgasse sicher ist, das wird sichergestellt durch UCS-Suche am Eingang der Sackgasse. In diesem Fall bekommen die Dots den Wert 1,23. Schließlich prüft der Agent, ob er sich in der Sackgasse befindet. Wenn dies nicht der Fall ist, dann wird einfach 0 vergeben. Der Standardwert für ein normaler Dot berechnet wird wie folgt berechnet: 1.0+1.233*(1/AnzahlÜbrigGebliebeneDots)-AnzahlDerGeisterInNähe(1)
GHOST, GHOST_AND_DOT, GHOST_AND_POWERPILL	Abhängig vom Geistertyp wird ein negativer Wert zugewiesen: {Hunter: -4.5, Eager: -4.0, Random: -3.5 } Dies sind die Werte, die in MDPs-Verfahren verwendet werden.Später werden diese Werte durch -5 ersetzt, um zu verhindern, dass der Agent in den Geist einläuft. Natürlich wird vorher geprüft, ob diese Geister unter dem Einfluss der Powerpill stehen, dann werden keine solchen Werte gesetzt, um dem Agenten die Möglichkeit zu geben, auch diese Geister zu eliminieren. Stattdessen werden positive Werte vergeben, und zwar auch nach Abhängigkeit vom Geistertyp.

Iteratives Erweitern der Werte (MDP_Runner):

Nachdem die Map mit den Anfangswerten „Runde 0“ gefüllt wurde, werden die weiteren Werte für die n Runden berechnet, mit n = Breite * Höhe (die Views) .
Außer bei Wänden und Pacman wird jedes Feld in jeder Runde aktualisiert. Dies geschieht mit Bezug eine Konstante „Gamma“ := 0.1 und den durchschnittlichen Wert aller benachbarten Felder.
Der neue Wert berechnet sich durch:

Range*((1-G)*mdpMap.getValue(i,j) + (G)*neighbourAvgValue) + STEP_COST

Range	Konstanter Faktor := 1, Pacman weicht von seinem Ziel nicht ab.
G	„Gamma“: Abwertungskonstante
STEP_COST	Konstanter Wert, der für die Schriitkosten steht.
neighbourAvgValue	Durchschnitt benachbarter Felder.

Nach diesem iterativen Prozess, d.h. nach n Runden, werden die oben genannten spezifischen Werte in der Map abgelegt. Diese Werte werden während des Mapping-Prozesses in einer externen Liste gespeichert, damit sie später in die Map übernehmen werden können. Sie ersetzen also bestimmte Werte des iterativen Prozesses durch eine der Situation angemessene Zahl.

Policy:

Nachdem die Map mit den Feinheiten bestückt wurde und nun die Endwerte vorliegen, kann der Agent nun die Werte seiner Nachbarn betrachten und sich an den Ort begeben, die seinen Nutzen maximiert.
Nach der letzten Iteration werden die Feinheiten in die Map eingefügt. Z.B. wird jedem Feld in der Nähe 1 eines Geistes der Wert -10 zugewiesen. Dies verhindert, dass der Pacman z.B. aus „Gier“ in den Geist hineinläuft.
Zum Schluss wird die beste Aktion anhand des höchsten Nachbarn ausgewählt und an den Server gesendet.

Deadend Handler:

Um die Kontrolle über die Sackgassen in einer Map zu behalten, wird ein Deadend-Handler verwendet. Dieser verwendet eine Matrix, die für jedes Feld die Anzahl der angrenzenden Wände berechnet.
Ist dieser Wert 3, so handelt es sich um das Ende einer Sackgasse.
Durch einen rekursiven Aufruf kann nun der Anfang der Sackgasse berechnet und die darin enthaltenen Felder zwischengespeichert werden. Diese sind für den Value Manager von großer Bedeutung, um die Werte entsprechend eintragen zu können.
Hinweis: Verschachtelte Sackgassen werden in diesem Modell nicht berücksichtigt.

Weiteres:

- Damit der Agent weiß, wo er in den letzten z.B. 5 Schritten war, wird ein Ringpuffer verwendet, um die letzten Positionen des Pacmans zu speichern. Dies würde dem Agenten helfen, mit dem Rest der Map zu experimentieren.
- Wie bereits erwähnt, wird für die Suche UCS verwendet. Es wurde so weit wie möglich vereinfacht. Es geht nur um die Position der Felder, sowie die Anzahl der Schritte bis dorthin. (s. package search)
- Wenn Sie die Map verfolgen möchten, dann können Sie im Package util nach der Klasse Logging suchen und dort ON auf true setzten. Es wird dann für jede Bewegung die spezielle Dead-End-Map ausgegeben, sowie die DBMS-Map.

Leistungen:

Welt	Ø Verbleibende Anzahl an Dots:	Siegesrate
#raster_RR	1	93.0%
#raster_HR	0	91.0%
#offen_HEE	10	45.0%
#labyrinth_RR	15	50.0%
#labyrinth_HR	9	48.0%

Bestes Ergebnis für 100 Runden beim 3-maligen Durchlaufen

Welt	Ø Verbleibende Anzahl an Dots:	Siegesrate
#raster_RR	1	94.0%
#raster_HR	3	86.3%
#offen_HEE	10	37.3%
#labyrinth_RR	19	40.3%
#labyrinth_HR	11	44.0%

300 Runden

Welt	Ø Verbleibende Anzahl an Dots:	Siegesrate
#raster_RR	1	91.5%
#raster_HR	2	85.1%
#offen_HEE	11	35.6%
#labyrinth_RR	16	46.4%
#labyrinth_HR	11	39.8%

1000 Runden

Entwurf: