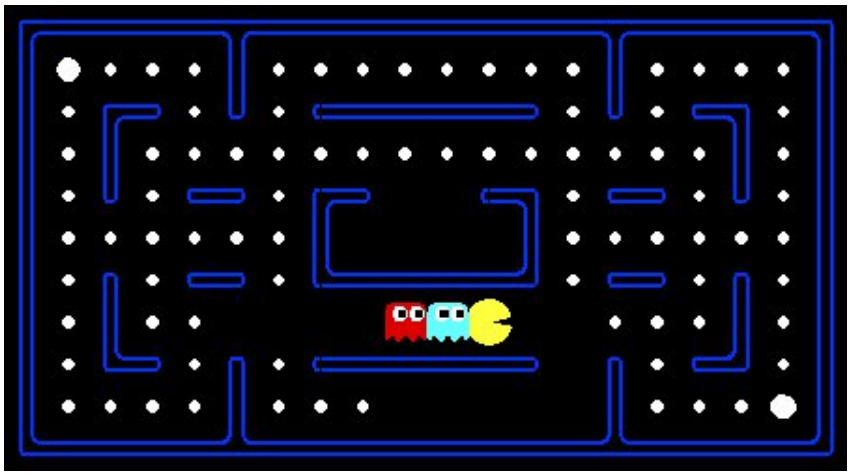
Praktikum 3: Multi-Agent Search



Einleitung:

In diesem Praktikum werden Sie Agenten für die klassische Version von Pacman entwerfen, einschließlich der Geister. Auf dem Weg werden Sie sowohl Minimax- als auch Expectimax-Suche implementieren und sich an der Gestaltung von Bewertungsfunktionen versuchen. Wir werden für dieses Praktikum das zweite Projekt des CS 188 Spring 2024 Kurses verwenden (https://inst.eecs.berkeley.edu/~cs188/sp24/projects/proj2/).

Das ZIP-Archiv für dieses Projekt "multiagent.zip" enthält die folgenden Dateien:

Dateien, die Sie bearbeiten werden:	
multiAgents.py	Hier werden alle Ihre Suchagenten enthalten sein.
Dateien, die Sie sich vielleicht ansehen möchten:	
pacman.py	Die Hauptdatei, die Pacman-Spiele ausführt. Diese Datei beschreibt auch einen Pacman GameState-Typ, den Sie in diesem Praktikum ausgiebig verwenden werden.
game.py	Die Logik hinter der Funktionsweise der Pacman-Welt. Diese Datei beschreibt mehrere unterstützende Typen wie AgentState, Agent, Direction und Grid.
util.py	Nützliche Datenstrukturen zur Implementierung von Suchalgorithmen. Sie müssen diese für dieses Praktikum nicht verwenden, aber möglicherweise sind sie für Ihre Implementierung nützlich.
Unterstützende Dateien, die Sie ignorieren können:	
graphicsDisplay.py	Grafiken für Pacman
graphicsUtils.py	Unterstützung für Pacman-Grafiken
textDisplay.py	ASCII-Grafiken für Pacman
ghostAgents.py	Agenten zur Steuerung von Geistern
keyboardAgents.py	Tastatur-Schnittstellen zur Steuerung von Pacman
layout.py	Code zum Lesen von Layoutdateien und Speichern ihres Inhalts
autograder.py	Projekt-Autograder
testParser.py	Analysiert Autograder-Test- und Lösungsdateien
testClasses.py	Allgemeine Autograding-Testklassen
test_cases/	Verzeichnis mit den Testfällen für jede Frage
multiagentTestClasses.py	Projekt-3-spezifische Autograding-Testklassen

Wenn sie das Projekt heruntergeladen haben können Sie mit folgendem Command eine Runde Pacman spielen:

```
python pacman.py
```

Damit z.B. der ReflexAgent Pacman steuert, können Sie folgenden Befehl ausführen:

```
python pacman.py -p ReflexAgent
```

Wenn Sie den ReflexAgent auf einem anderen Layout testen wollen können Sie diesen Befehl ausführen:

```
python pacman.py -p ReflexAgent -l testClassic
```

Sie werden sehen das selbst auf dem simplen Layout `testClassic` der aktuell ReflexAgent schlechte Ergebnisse liefert. Ihr Ziel für die nächsten Aufgaben wird es sein mit unterschiedlichen Methoden einen guten Agenten zu entwerfen.

Aufgabe 1: ReflexAgent

Verbessern Sie den ReflexAgent in multiAgents.py, damit er anständig spielt. Der bereitgestellte Code des ReflexAgents bietet einige nützliche Beispiele für Methoden, die den GameState nach Informationen abfragen. Ein fähiger ReflexAgent muss sowohl die Positionen von Nahrung als auch die von Geistern berücksichtigen, um gut zu performen. Ihr Agent sollte das TestClassic-Layout leicht und zuverlässig meistern können:

```
python pacman.py -p ReflexAgent -l testClassic
```

Mit einer sehr guten Bewertungsfunktion sollten Sie es auch schaffen zuverlässig das Standard-Layout mit 1 und 2 Geistern zu schaffen (mit `--no-graphics` können Sie die Grafik ausstellen):

```
python pacman.py --frameTime 0 -p ReflexAgent -k 1
```

```
python pacman.py --frameTime 0 -p ReflexAgent -k 2
```

Standardmäßig ist die Bewegung der Geister zufällig. Sie können auch zum Spaß mit etwas intelligenteren richtungsabhängigen Geistern spielen, indem Sie `-g DirectionalGhost` verwenden.

Das Projekt bietet einen Autograder, welcher für jede Aufgabe prüft, ob Sie die Aufgabe richtig gelöst haben. Der Autograder testet für die erste Aufgabe den ReflexAgenten 10-mal auf dem Layout openClassic aus. Sie erhalten 0 Punkte, wenn Ihr Agent abstürzt oder nie gewinnt. Sie erhalten 1 Punkt, wenn Ihr Agent mindestens 5-mal gewinnt, oder 2 Punkte, wenn Ihr Agent alle 10 Spiele gewinnt. Sie erhalten einen zusätzlichen Punkt, wenn die durchschnittliche Punktzahl Ihres Agenten größer als 500 ist, oder 2 Punkte, wenn sie größer als 1000 ist. Sie können Ihren Agenten unter diesen Bedingungen mit folgendem Befehl testen:

```
python autograder.py -q q1
```

Geben Sie in Ihrer Präsentation zur Praktikumsabgabe an wie viele Punkte Sie erhalten haben (inkl. Screenshot der Konsole).

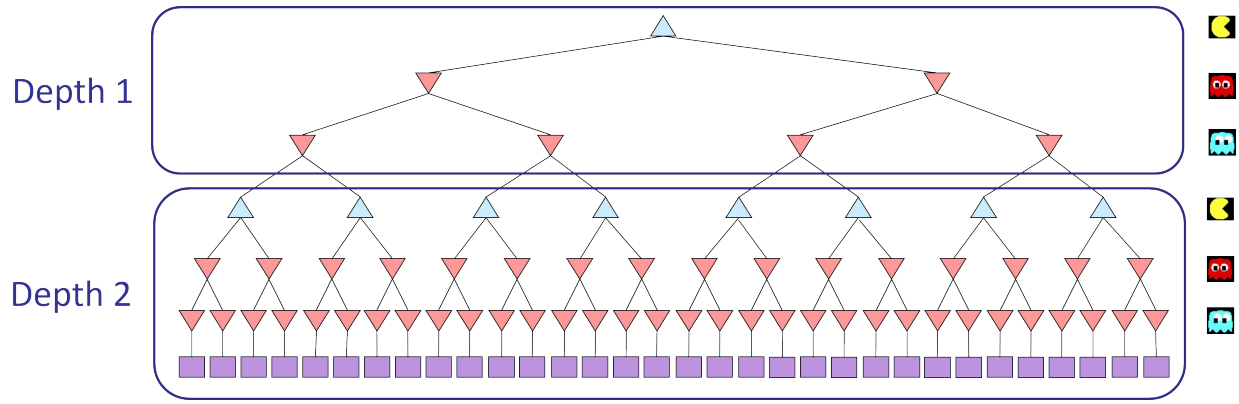
Hinweise:

– Denken Sie daran, dass newFood die Funktion asList() hat.

– Versuchen Sie den Kehrwert wichtiger Werte (wie die Entfernung zu Nahrung) zu verwenden, anstatt nur die Werte selbst.
– Es kann hilfreich sein, die internen Inhalte verschiedener Objekte zum Debuggen anzusehen. Dies können Sie tun, indem Sie die String-Darstellungen der Objekte ausdrucken. Sie können beispielsweise newGhostStates mit print(newGhostStates) ausdrucken.

Aufgabe 2: Minimax

Im nächsten Schritt werden Sie den MinimaxAgent implementieren. Ihr MinimaxAgent sollte mit einer beliebigen Anzahl von Geistern funktionieren. Insbesondere wird Ihr Minimax-Baum für jede Max-Ebene mehrere Min-Ebenen haben (eine für jeden Geist). Ihr Code sollte auch den Spielbaum bis zu einer beliebigen Tiefe erweitern. Bewerten Sie die Blätter Ihres Minimax-Baums mit der bereitgestellten self.evaluationFunction.



Der MinimaxAgent erweitert MultiAgentSearchAgent, was den Zugriff auf self.depth und self.evaluationFunction ermöglicht. Stellen Sie sicher, dass Ihr Minimax-Code auf diese beiden Variablen Bezug nimmt.

Der Autograder für diese Aufgabe untersucht Ihren Code daraufhin, ob der MinimaxAgent die korrekte Anzahl von Spielzuständen erkundet. Als Ergebnis wird der Autograder sehr pingelig sein, wie oft Sie GameState.generateSuccessor aufrufen. Wenn Sie es öfter oder weniger oft als nötig aufrufen, wird der Autograder reklamieren.

```
python autograder.py -q q2
```

Analysieren Sie auch das Verhalten vom MinimaxAgenten in unterschiedlichen Layouts. Was fällt Ihnen auf? Notieren Sie Ihre Ergebnisse/Erkenntnisse in Ihrer Präsentation.

```
python pacman.py -p MinimaxAgent -l openClassic -a depth=3

python pacman.py -p MinimaxAgent -l mediumClassic -a depth=3

python pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4

python pacman.py -p MinimaxAgent -l trappedClassic -a depth=3
```

Hinweise:

– Implementieren Sie den Algorithmus am besten rekursiv.
– Die korrekte Implementierung von Minimax wird dazu führen, dass Pacman das Spiel in einigen Tests verliert. Dies ist kein Problem: Da es das korrekte Verhalten ist, wird es die Tests bestehen.

– Die Approximation der Bewertungsfunktion für den Pacman-Test in diesem Teil ist bereits geschrieben (self.evaluationFunction). Sie sollten diese Funktion nicht ändern, aber erkennen, dass wir jetzt Zustände statt Aktionen bewerten, wie wir es beim Reflex-Agenten getan haben.
– Pacman ist immer Agent 0, und die Agenten ziehen in der Reihenfolge des steigenden Agentenindex.
– Wenn Pacman glaubt, dass sein Tod unvermeidlich ist, wird er versuchen, das Spiel so schnell wie möglich zu beenden, wegen der konstanten Strafe für das weitere Leben.

Aufgabe 3: Alpha-Beta Pruning

Erstellen Sie einen neuen Agenten in der Klasse AlphaBetaAgent, der Alpha-Beta Pruning verwendet, um den Minimax-Baum effizienter zu untersuchen. Prüfen Sie wie viel schneller die Suche mit Alpha-Beta Pruning abläuft. Im Idealfall sollte Tiefe 3 auf smallClassic in nur wenigen Sekunden pro Zug oder schneller ablaufen.

```
python pacman.py -p AlphaBetaAgent -a depth=3 -l smallClassic
```

Die Minimax-Werte des AlphaBetaAgent sollten mit den Minimax-Werten des MinimaxAgent identisch sein. Allerdings können sich die ausgewählten Aktionen aufgrund unterschiedlichen Verhaltens bei Unentschieden unterscheiden.

Um Ihren Code zu testen und zu debuggen, führen Sie folgende Befehl aus:

```
python autograder.py -q q3
```

Aufgabe 4: Expectimax

Der MinimaxAgent und AlphaBetaAgent gehen immer davon aus das der Gegner die optimale Entscheidung trifft. Jedoch ist das nicht immer der Fall. Jeder, der schon einmal Tic-Tac-Toe gewonnen hat, kann das bestätigen. In dieser Aufgabe werden Sie den ExpectimaxAgent implementieren, welcher auch davon ausgeht, dass der Gegner nicht die optimale Entscheidung trifft. ExpectimaxAgent nimmt nicht mehr den Mindestwert über alle Geisteraktionen, sondern berücksichtigt die erwartete Aktion gemäß dem Modell des Agenten, wie die Geister handeln. Um Ihren Code zu vereinfachen, nehmen Sie an, dass Sie nur gegen einen Gegner antreten, der seine getLegalActions gleichmäßig nach dem Zufallsprinzip auswählt.

Wie bei den Aufgaben davor können Sie Ihre Implementierung mit folgendem Befehl testen:

```
python autograder.py -q q4
```

Vergleichen Sie die Performance vom ExpectimaxAgent und dem AlphaBetaAgent miteinander. Besonders beim trappedClassic Layout sollten Sie einen Unterschied erkennen können. Fügen Sie Ihre Erkenntnisse in Ihre Präsentation ein.

```
python pacman.py -p AlphaBetaAgent -l trappedClassic -a depth=3 -q -n 10
```

```
python pacman.py -p ExpectimaxAgent -l trappedClassic -a depth=3 -q -n 10
```

[Optional] Aufgabe 5: Evaluation Function

In Aufgabe 2 bis 4 haben wir nur den Gamescore betrachtet, um den Zustand zu bewerten. In dieser optionalen Aufgabe wollen wir eine bessere Evaluations Funktion entwickeln. Wenn Sie den

ExpectimaxAgent mit einer besseren Evaluation Function testen wollen, implementieren Sie die Funktion betterEvaluationFunction und führen Sie folgenden Befehl aus:

```
python autograder.py -q q5
```

Der Autograder wird Ihren Agenten 10-mal auf dem smallClassic-Layout ausführen. Der Autograder wird Ihrer Bewertungsfunktion auf folgende Weise Punkte zuweisen:

- Wenn Sie mindestens einmal gewinnen, ohne dass der Autograder die Zeit überzieht, erhalten Sie 1 Punkt. Jeder Agent, der diese Kriterien nicht erfüllt, erhält 0 Punkte.
- +1, wenn Sie mindestens 5-mal gewinnen, +2, wenn Sie alle 10-mal gewinnen
- +1 für eine durchschnittliche Punktzahl von mindestens 500, +2 für eine durchschnittliche Punktzahl von mindestens 1000 (einschließlich verlorener Spiele)
- +1, wenn Ihre Partien im Durchschnitt weniger als 30 Sekunden auf dem Autograder benötigen, wenn sie mit --no-graphics ausgeführt werden.
- Die zusätzlichen Punkte für die Durchschnittspunktzahl und die Berechnungszeit werden nur vergeben, wenn Sie mindestens 5-mal gewinnen.

Abgabe

1. Laden Sie Ihre Ergebnisse zusammen mit einer kurzen Präsentation in ILU hoch, damit wir überprüfen können, ob Sie das Praktikum korrekt bearbeitet haben.
2. Bereiten Sie sich auf den MCT zu diesem Praktikum vor. Zur Erinnerung: Sie benötigen insgesamt mindestens 15 Punkte aus allen drei MCTs und müssen in mindestens zwei MCTs jeweils mindestens 5 Punkte erreichen, um das Praktikum zu bestehen.