

Philipps



Universität
Marburg

Making a Pacman Artificial Intelligence

Dienstag

30.08.2022

11:30 - 12:30 Uhr



hessian.AI

Konkurrierende Pacmans

- Schreibe eine Funktion, die Pacman sagt, wie er sich bewegen soll
- Wähle eine Farbe
- Die verschiedenen Pacmans werden am Ende gegeneinander antreten!
- Ändere dazu lediglich die Datei mypacman.py



Getting started

Schaue dir den Start-Code an: **Day_2/Code/lecture_5_code**

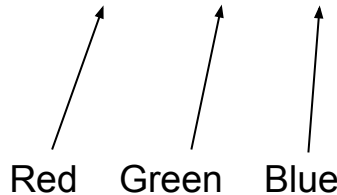


Your Pacman: Farbe wählen

- Kann eine Farbe deiner Wahl sein
- Die Farbe kann GELB, WEISS, ROT, PINK, TEAL, ORANGE, GRÜN sein.
- Benutzerdefinierte Farbe RGB-Werten (0-255) definieren.

```
color = ( 255, 10, 0)
```

Red Green Blue



```
class MyPacmanAI(Pacman):  
    def __init__(self, node, playerNum):  
        color = GREEN  
        Pacman.__init__(self, node,  
color, playerNum)
```

Pacman bewegen

Implementiere lediglich die Funktion move()

```
class MyPacmanAI(Pacman):  
    def __init__(self, node, playerNum):  
        color = GREEN  
        Pacman.__init__(self, node, color, playerNum)  
  
    def move(self, opponent, pellets, fruit, ghosts):  
        direction = random.choice([UP, DOWN, LEFT, RIGHT])  
        return direction
```

Pacman bewegen

Implementiere lediglich die Funktion move()

```
class MyPacmanAI(Pacman):  
    def __init__(self, node, playerNum):  
        color = GREEN  
        Pacman.__init__(self, node, color, playerNum)  
  
    def move(self, opponent, pellets, fruit, ghosts):  
        direction = random.choice([UP, DOWN, LEFT, RIGHT])  
        return direction
```

Zu Beginn haben wir eine **Liste** möglicher **Richtungen** [AUF, AB, LINKS, RECHTS] und der Code wählt **zufällig** eine aus

Pacman bewegen

Implementiere lediglich die Funktion move()

```
class MyPacmanAI(Pacman):  
    def __init__(self, node, playerNum):  
        color = GREEN  
        Pacman.__init__(self, node, color, playerNum)  
  
    def move(self, opponent, pellets, fruit, ghosts):  
        direction = random.choice([UP, DOWN, LEFT, RIGHT])  
        return direction
```

Das Spiel wird mehrmals pro Sekunde aktualisiert und ruft jedes Mal move() auf.

Mit anderen Worten, die Update- und Move-Funktionen werden in jeder **Iteration** der **Game Loop** ausgeführt.

Was wird Pacman tun?

Zu Beginn haben wir eine **Liste** möglicher **Richtungen** [AUF, AB, LINKS, RECHTS] und der Code wählt **zufällig** eine aus

Pacman bewegen

Implementiere lediglich die Funktion move()

```
class MyPacmanAI(Pacman):  
    def __init__(self, node, playerNum):  
        color = GREEN  
        Pacman.__init__(self, node, color, playerNum)  
  
    def move(self, opponent, pellets, fruit, ghosts):  
        direction = random.choice([UP, DOWN, LEFT, RIGHT])  
        return direction
```

Zu Beginn haben wir eine **Liste** möglicher **Richtungen** [AUF, AB, LINKS, RECHTS] und der Code wählt **zufällig** eine aus

Das Spiel wird mehrmals pro Sekunde aktualisiert und ruft jedes Mal move() auf.

Mit anderen Worten, die Update- und Move-Funktionen werden in jeder **Iteration** der **Game Loop** ausgeführt.

Was wird Pacman tun?

💡 Hinweis: Es gibt eine weitere Richtung STOP, in die Pacman geht, wenn er auf eine Wand trifft.

Run the Code

Code laufen lassen, indem du folgenden Befehl ausführst:

```
python run.py
```

- Daraufhin wird das Spielfenster geöffnet.
- Das Spiel wird gestartet, wenn du die Leertaste drückst.
- Du kannst das Spiel neu starten, indem du das Fenster schließt und das Skript erneut ausführst.

Pacman bewegen

- Was wird Pacman tun?
- Er springt hin und her und bewegt sich nicht sehr weit
- Pacman ändert seine Meinung zu schnell!
- Wir können das mit Timern beheben



Using Timers

- Wir können das mit Timern beheben

```
def move(self, opponent, pellets, fruit, ghosts):  
    direction = self.direction  
    if self.dt[0] > 2:  
        self.resetTimer(0)  
        direction = random.choice([UP, DOWN, LEFT, RIGHT])  
    return direction
```

- Verwende `self.dt[0]` für den Timer 0, `self.dt[1]` für Timer 1 und so weiter.
- Timer zählen Sekunden, `self.dt[0] > 2` bedeutet also: "wenn zwei Sekunden verstrichen sind"
- `self.resetTimer(0)` setzt den Timer 0 zurück und sagt ihm, dass er von vorne beginnen soll.

Using Timers

- Wir können das mit Timern beheben

```
def move(self, opponent, pellets, fruit, ghosts):  
    direction = self.direction  
    if self.dt[0] > 2:  
        self.resetTimer(0)  
        direction = random.choice([UP, DOWN, LEFT, RIGHT])  
    return direction
```

- Variablen mit "self." gehören zu deinem Pacman
- Variablen ohne "self." sind temporär und existieren nicht außerhalb von move()
- direction ist die Richtung, in die **Pacman gehen soll**
- self.direction ist die Richtung, in die **Pacman gerade geht**
- Was wird Pacman jetzt tun?

Time 1:

self.direction=RIGHT
direction=DOWN



Time 2:

self.direction=DOWN
direction=...



Pacman's Strategy

Was wird Pacman jetzt tun?

Probleme:

1. Pacman wählt die Richtung, kann aber auch gegen eine Wand laufen
2. Ignoriert den Geist
3. Bewegt sich nicht auf Pallets oder Früchte zu

Das erste Problem kann durch

`self.validDirections()` behoben werden. Dies gibt nur Richtungen zurück, in die sich Pacman bewegen kann, zum Beispiel [UP, LEFT] hier



Problem 1

Pacman wählt die Richtung, kann aber auch gegen eine Wand laufen.

Pacman muss wissen, in welche Richtung er sich bewegen kann!

Das erste Problem kann durch

`self.validDirections()` behoben werden. Dies gibt nur Richtungen zurück, in die sich Pacman bewegen kann, zum Beispiel [UP, LEFT] hier



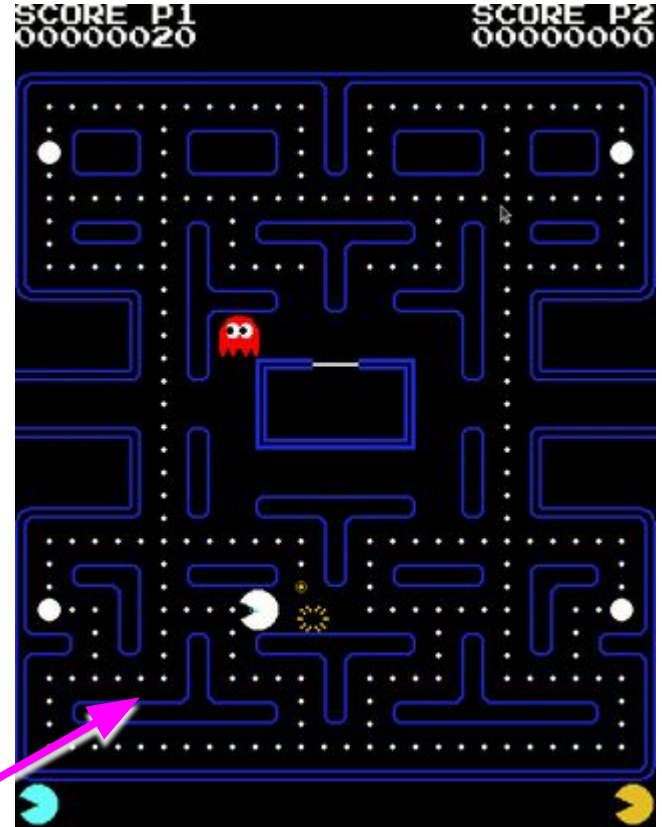
Problem 1

Pacman wählt die Richtung, kann aber auch gegen eine Wand laufen.

Wenn Pacman gegen eine Wand läuft, soll er in Bewegung bleiben. Wir können die STOP-Richtung wie folgt überprüfen:

```
elif self.dt[0] > 3 or self.direction == STOP:
```

Jetzt ändert Pacman alle 3 Sekunden die Richtung ODER wenn er gegen eine Wand läuft.



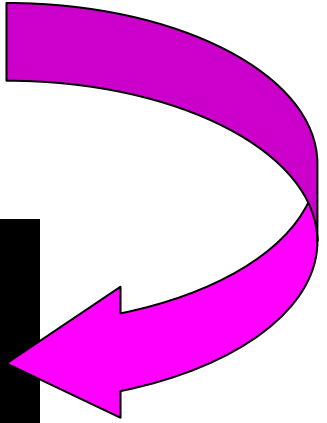
Sich auf Objekte zubewegen

- Um sich auf etwas zubewegen zu können, müssen wir seine **Position** (Standort) kennen.

```
def move(self, opponent, pellets, fruit, ghosts):  
    directions = self.validDirections()  
    self.goal = ghosts.blinky.position  
    printDirections(directions)  
    directions = self.goalDirection(directions)  
    direction = random.choice(directions)  
    return direction
```

Sich auf Objekte zubewegen

- Um sich auf etwas zuzubewegen, müssen wir dessen Position kennen
- Die Position des Geistes wird in **ghosts.blinky.position** gespeichert.



```
def move(self, opponent, pellets, fruit, ghosts):  
    directions = self.validDirections()  
    self.goal = ghosts.blinky.position  
    printDirections(directions)  
    directions = self.goalDirection(directions)  
    direction = random.choice(directions)  
    return direction
```

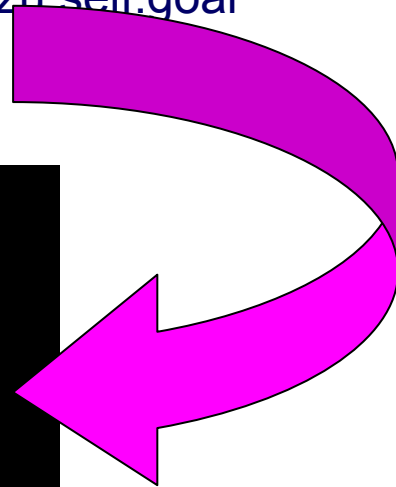
Sich auf Objekte zubewegen

```
def goalDirection(self, directions, minimize=True):  
    distances = []  
    for direction in directions:  
        goalDist = self.tileDistance(self.position + self.directions[direction]*TILEWIDTH, self.goal)  
        distances.append(goalDist)  
  
    matchValue = min(distances) if minimize else max(distances)  
    indices = [i for i in range(len(directions)) if distances[i] == matchValue]  
    goalDirections = []  
    for i in range(len(directions)):  
        if distances[i] == matchValue:  
            goalDirections.append(directions[i])  
    return goalDirections
```

Sich auf Objekte zubewegen

- Um sich auf etwas zuzubewegen, müssen wir dessen Position kennen
- Die Position des Geistes wird in **ghosts.blinky.position** gespeichert.
- Pacman hat eine Funktion `self.goalDirection()`, die die Richtung zu `self.goal` zurückgibt

```
def move(self, opponent, pellets, fruit, ghosts):  
    directions = self.validDirections()  
    self.goal = ghosts.blinky.position  
    printDirections(directions)  
    directions = self.goalDirection(directions)  
    direction = random.choice(directions)  
    return direction
```



Sich auf Objekte zubewegen

- Um sich auf etwas zuzubewegen, müssen wir dessen Position kennen
- Die Position des Geistes wird in `ghosts.blinky.position` gespeichert.
- Pacman hat eine Funktion `self.goalDirection()`, die die Richtung zu `self.goal` zurückgibt
- Wir können diese benutzen, um Pacman zu einem Objekt zu lenken:

```
def move(self, opponent, pellets, fruit, ghosts):  
    directions = self.validDirections()  
    self.goal = ghosts.blinky.position  
    printDirections(directions)  
    directions = self.goalDirection(directions)  
    direction = random.choice(directions)  
    return direction
```

Sich auf Objekte zubewegen

- Um sich auf etwas zuzubewegen, müssen wir dessen Position kennen
- Die Position des Geistes wird in `ghosts.blinky.position` gespeichert.
- Pacman hat eine Funktion `self.goalDirection()`, die die Richtung zu `self.goal` zurückgibt
- Wir können diese benutzen, um Pacman zu einem Objekt zu lenken:

```
def move(self, opponent, pellets, fruit, ghosts):  
    directions = self.validDirections()  
    self.goal = ghosts.blinky.position  
    printDirections(directions)  
    directions = self.goalDirection(directions)  
    direction = random.choice(directions)  
    return direction
```

💡 Tipps:

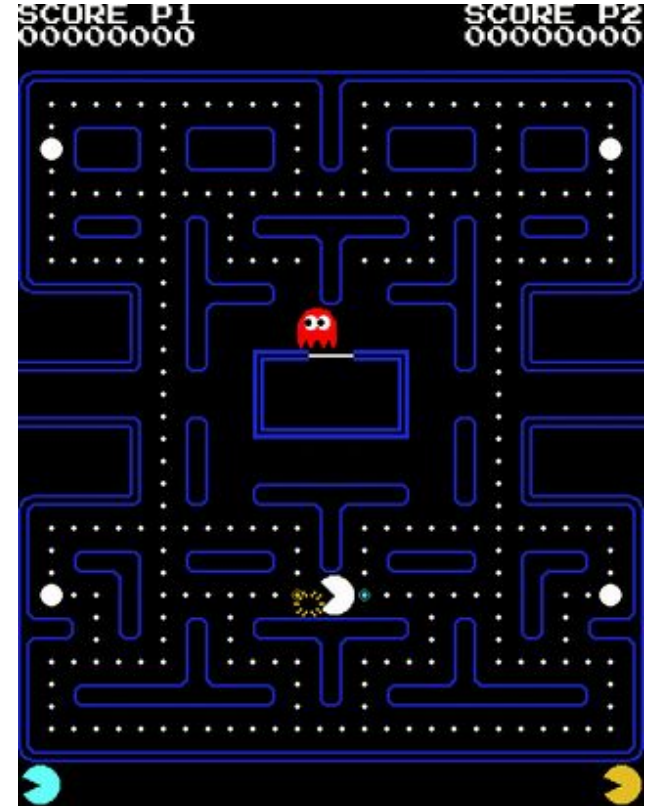
`printDirections()` gibt die Richtungsnamen im Terminal aus

Hinweis: `self.goalDirection()` nimmt eine Liste von Richtungen als Eingabe und gibt diejenige zurück, die dem Ziel am nächsten kommt. Das sind oft zwei Richtungen, so dass wir immer noch `random.choice` brauchen, um eine auszuwählen.

Sich auf Objekte zubewegen

- Was macht Pacman jetzt?

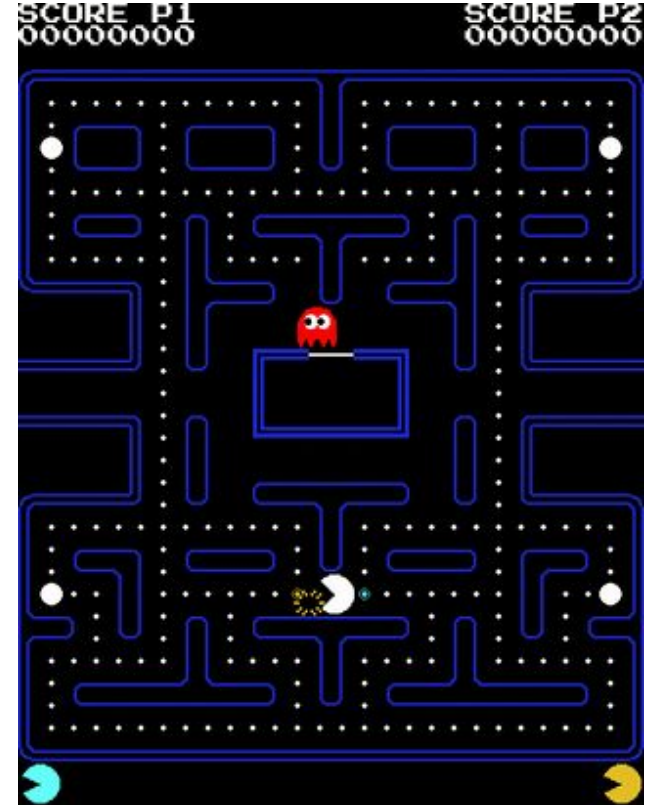
```
def move(self, opponent, pellets, fruit, ghosts):  
    directions = self.validDirections()  
    self.goal = ghosts.blinky.position  
    printDirections(directions)  
    directions = self.goalDirection(directions)  
    direction = random.choice(directions)  
    return direction
```



Sich auf Objekte zubewegen

- Was macht Pacman jetzt?
- **Er bewegt sich in Richtung der Geister!**

```
def move(self, opponent, pellets, fruit, ghosts):  
    directions = self.validDirections()  
    self.goal = ghosts.blinky.position  
    printDirections(directions)  
    directions = self.goalDirection(directions)  
    direction = random.choice(directions)  
    return direction
```



Sich auf Objekte zubewegen

- Was macht Pacman jetzt?
- Er bewegt sich in Richtung der Geister!



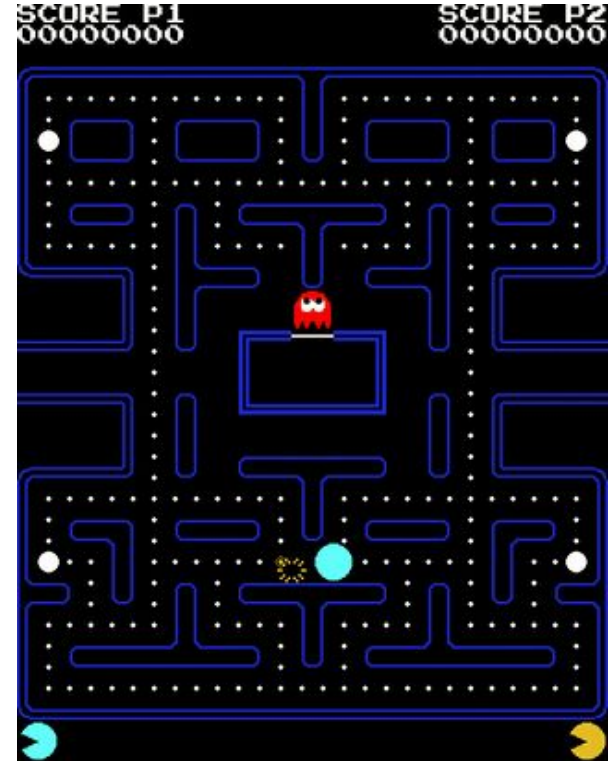
```
def move(self, opponent, pellets, fruit, ghosts):  
    directions = self.validDirections()  
    self.goal = ghosts.blinky.position  
    printDirections(directions)  
    directions = self.goalDirection(directions,  
    minimize=False)  
    direction = random.choice(directions)  
    return direction
```

Um sich weg statt hin zu bewegen, müssen wir minimize=False setzen.

Damit wird Pacman angewiesen, die Richtung zu finden, die den Abstand maximiert, anstatt ihn zu minimieren.

Sich von Objekten entfernen

- Pacman versteckt sich in der Ecke
- Er muss sich nicht um den Geist kümmern, es sei denn, er ist in der Nähe.
- Ansonsten kann er sich immer noch zufällig bewegen.



Conditional Movement

Um Geister nur zu meiden, wenn sie in der Nähe sind, und sich ansonsten zufällig zu bewegen, haben wir jetzt:


```
def move(self, opponent, pellets, fruit, ghosts):  
    directions = [self.direction]  
    d2ghost = self.tileDistance(ghosts.blinky)  
  
    if d2ghost < 10:  
        self.goal = ghosts.blinky.position  
        directions = self.validDirections()  
        printDirections(directions)  
        directions = self.goalDirection(directions, minimize=False)  
    elif self.dt[0] > 3 or self.direction == STOP:  
        self.resetTimer(0)  
        directions = self.validDirections()  
  
    direction = random.choice(directions)  
    return direction
```

Conditional Movement

self.tileDistance()
gibt die Entfernung
zwischen Pacman
und dem Ziel in
Tiles zurück

Tipp: Du kannst
dies auch während
des Testens
ausgeben

```
def move(self, opponent, pellets, fruit, ghosts):  
    directions = [self.direction]  
    d2ghost = self.tileDistance(ghosts.blinky)  
  
    if d2ghost < 10:  
        self.goal = ghosts.blinky.position  
        directions = self.validDirections()  
        printDirections(directions)  
        directions = self.goalDirection(directions, minimize=False)  
    elif self.dt[0] > 3 or self.direction == STOP:  
        self.resetTimer(0)  
        directions = self.validDirections()  
  
    direction = random.choice(directions)  
    return direction
```



Conditional Movement

Wenn der Geist weniger als 10 Felder entfernt ist, bewege dich weg.

Ansonsten wähle alle 3 Sekunden eine neue Richtung oder wenn Pacman gegen eine Wand läuft.

```
def move(self, opponent, pellets, fruit, ghosts):  
    directions = [self.direction]  
    d2ghost = self.tileDistance(ghosts.blinky)  
  
    if d2ghost < 10:  
        self.goal = ghosts.blinky.position  
        directions = self.validDirections()  
        printDirections(directions)  
        directions = self.goalDirection(directions, minimize=False)  
    elif self.dt[0] > 3 or self.direction == STOP:  
        self.resetTimer(0)  
        directions = self.validDirections()  
  
    direction = random.choice(directions)  
    return direction
```

Conditional Movement

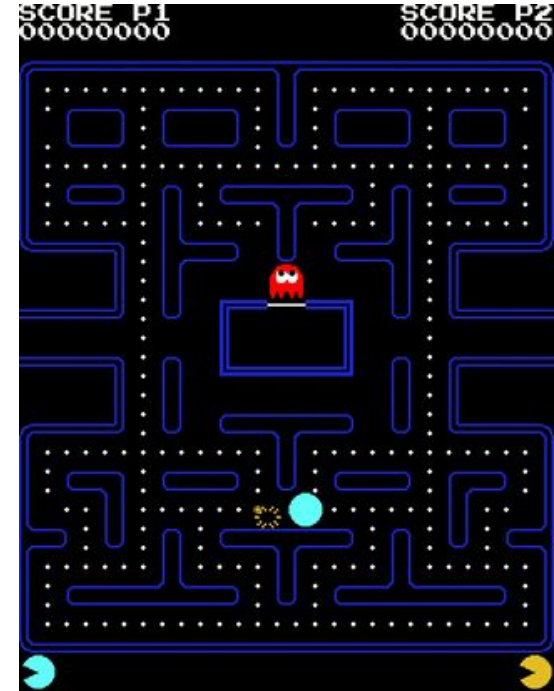
```
def move(self, opponent, pellets, fruit, ghosts):  
    directions = [self.direction]  
    d2ghost = self.tileDistance(ghosts.blinky)  
  
    if d2ghost < 10:  
        self.goal = ghosts.blinky.position  
        directions = self.validDirections()  
        printDirections(directions)  
        directions = self.goalDirection(directions, minimize=False)  
    elif self.dt[0] > 3 or self.direction == STOP:  
        self.resetTimer(0)  
        directions = self.validDirections()  
  
    direction = random.choice(directions)  
    return direction
```



Verstehst du, warum der Code das Verhalten des Pacman verursacht?

Bewegung in Richtung der Pallets

- Toll, aber wir wollen uns nicht wahllos bewegen. Wir wollen die Pallets essen.
- Zur Erinnerung: `self.goalDirection()` hilft uns, die Richtungen in Bezug auf `self.goal` zu bestimmen, das den Standort des Zielobjekts darstellt.
- Wir können `self.goalDirection()` verwenden, um uns auf Pallet-Objekte zu bewegen. Aber auf welches Pallet bewegen wir uns zu?

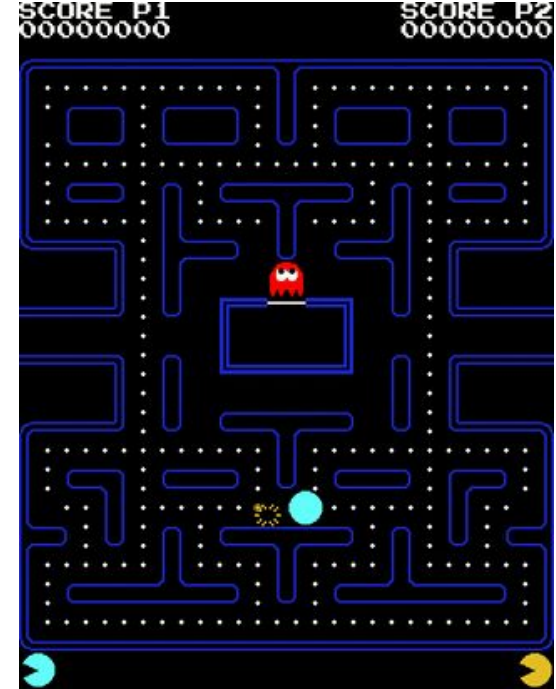


Bewegung in Richtung der Pallets

Wir können `self.goalDirection()` verwenden, um uns auf Pallet-Objekte zu bewegen. Aber auf welche Pallet bewegen wir uns zu?

Wir müssen:

1. Entfernungen zu allen Pallets ermitteln
2. Das Pallet mit der kürzesten Entfernung finden
3. Den Standort dieses Pallets als Ziel festlegen

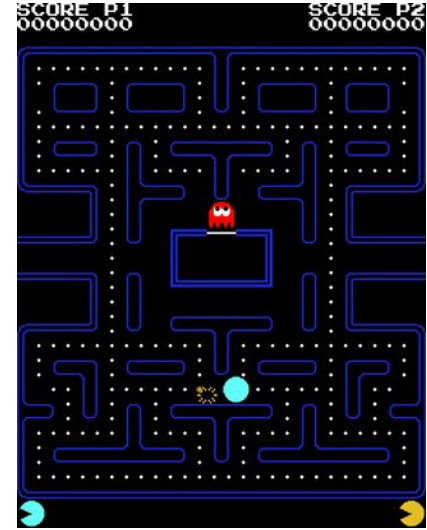


Bewegung in Richtung der Pallets

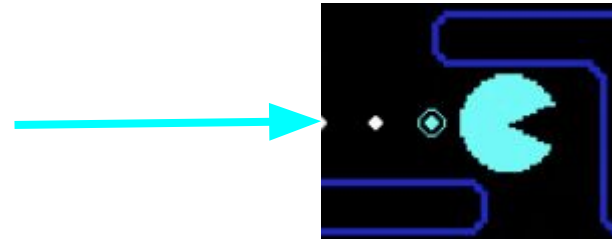
Wir können `self.goalDirection()` verwenden, um uns auf Pallet-Objekte zu bewegen. Aber auf welche Pallet bewegen wir uns zu?

Wir müssen:

1. Entfernungen zu allen Pallets ermitteln
2. Das Pallet mit der kürzesten Entfernung finden
3. Den Standort dieses Pallets als Ziel festlegen



💡 Die Pallet, die Pacman am nächsten ist, wird in deiner Farbe hervorgehoben



Bewegung in Richtung der Pallets

Wir müssen:

1. Entfernungen zu allen Pallets ermitteln
2. Das Pallet mit der kürzesten Entfernung finden
3. Den Standort dieses Pallets als Ziel festlegen

```
pelletDists = []  
for pellet in pellets.pelletList:  
  
    pelletDists.append(self.tileDistance(pellet)  
    )  
mindist = min(pelletDists)  
index = pelletDists.index(mindist)  
self.goal = pellets.pelletList[index].position  
directions = self.validDirections()  
directions = self.goalDirection(directions)
```

Bewegung in Richtung der Pallets

Wir müssen:

1. **Entfernungen zu allen Pallets ermitteln**
2. Das Pallet mit der kürzesten Entfernung finden
3. Den Standort dieses Pallets als Ziel festlegen

Durchlauf der Pellets und Hinzufügen der Entfernungen zu einer Liste pelletDists

```
pelletDists = []  
for pellet in pellets.pelletList:  
    pelletDists.append(self.tileDistance(pellet))  
mindist = min(pelletDists)  
index = pelletDists.index(mindist)  
self.goal = pellets.pelletList[index].position  
directions = self.validDirections()  
directions = self.goalDirection(directions)
```

Bewegung in Richtung der Pallets

Wir müssen:

1. Entfernungen zu allen Pallets ermitteln
2. **Das Pallet mit der kürzesten Entfernung finden**
3. Den Standort dieses Pallets als Ziel festlegen

Durchlauf der
Pellets und
Hinzufügen der
Entfernungen zu
einer Liste
pelletDists

```
pelletDists = []  
for pellet in pellets.pelletList:  
    pelletDists.append(self.tileDistance(pellet))  
mindist = min(pelletDists)  
index = pelletDists.index(mindist)  
self.goal = pellets.pelletList[index].position  
directions = self.validDirections()  
directions = self.goalDirection(directions)
```

mindist auf die
kleinste Zahl in der
Liste setzen

Bewegung in Richtung der Pallets

Wir müssen:

1. Entfernungen zu allen Pallets ermitteln
2. **Das Pallet mit der kürzesten Entfernung finden**
3. Den Standort dieses Pallets als Ziel festlegen

Durchlauf der
Pellets und
Hinzufügen der
Entfernungen zu
einer Liste
pelletDists

```
pelletDists = []  
for pellet in pellets.pelletList:  
    pelletDists.append(self.tileDistance(pellet))  
mindist = min(pelletDists)  
index = pelletDists.index(mindist)  
self.goal = pellets.pelletList[index].position  
directions = self.validDirections()  
directions = self.goalDirection(directions)
```

mindist auf die
kleinste Zahl in der
Liste setzen

Index auf die Stelle in
der Liste setzen, die den
kleinsten Wert enthält

Bewegung in Richtung der Pallets

Wir müssen:

1. Entfernungen zu allen Pallets ermitteln
2. Das Pallet mit der kürzesten Entfernung finden
3. **Den Standort dieses Pallets als Ziel festlegen**

Durchlauf der Pellets und Hinzufügen der Entfernungen zu einer Liste pelletDists

```
pelletDists = []  
for pellet in pellets.pelletList:  
    pelletDists.append(self.tileDistance(pellet))  
mindist = min(pelletDists)  
index = pelletDists.index(mindist)  
self.goal = pellets.pelletList[index].position  
directions = self.validDirections()  
directions = self.goalDirection(directions)
```

mindist auf die kleinste Zahl in der Liste setzen

Index auf die Stelle in der Liste setzen, die den kleinsten Wert enthält

Setze das Ziel auf die Position dieses Pallets

Bewegung in Richtung der Pallets

```
if d2ghost < 10:
    self.goal = ghosts.blinky.position
    directions = self.validDirections()
    directions = self.goalDirection(directions, minimize=False)
elif len(pellets.pelletList) > 0:
    pelletDists = []
    for pellet in pellets.pelletList:
        pelletDists.append(self.tileDistance(pellet))
    mindist = min(pelletDists)
    index = pelletDists.index(mindist)
    self.goal = pellets.pelletList[index].position
    directions = self.validDirections()
    directions = self.goalDirection(directions)
```

Wir können diesen Code in die move-Funktion einfügen, anstelle des folgenden Codes

```
elif self.dt[0] > 3 or
self.direction == STOP:
```

Wir ändern auch die elif-Anweisung, um zu prüfen, ob es irgendwelche Pallets auf dem Bildschirm gibt. Dies kommt nach der Überprüfung des Abstands zum Geist, hat also die zweite Priorität.

Bewegung in Richtung der Pallets

```
if d2ghost < 10:
    self.goal = ghosts.blinky.position
    directions = self.validDirections()
    directions = self.goalDirection(directions, minimize=False)
elif len(pellets.pelletList) > 0:
    pelletDists = []
    for pellet in pellets.pelletList:
        pelletDists.append(self.tileDistance(pellet))
    mindist = min(pelletDists)
    index = pelletDists.index(mindist)
    self.goal = pellets.pelletList[index].position
    directions = self.validDirections()
    directions = self.goalDirection(directions)
```

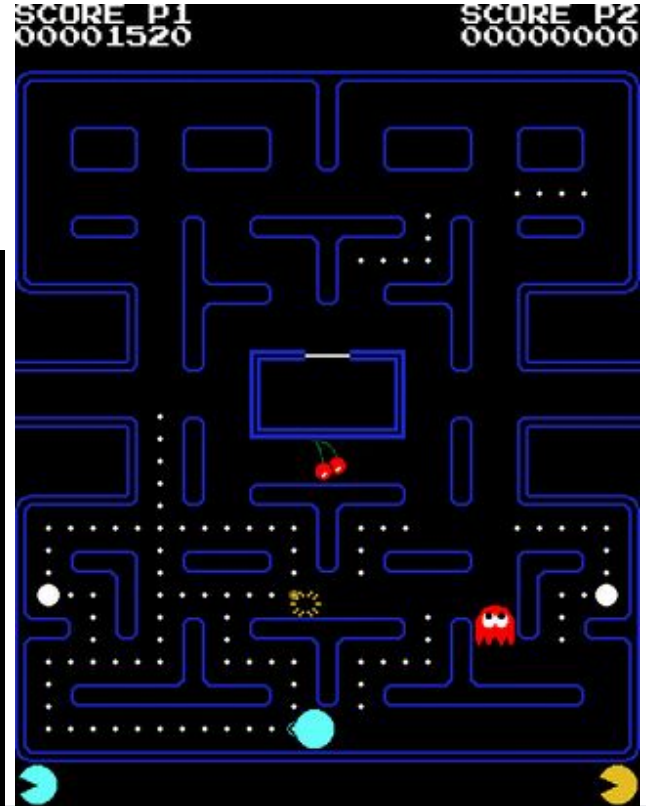


Verstehst du, warum der Code das Verhalten des Pacman verursacht?

Pacmans Strategie

Pacman weicht dem Geist auch dann aus, wenn er blau ist. Dies wird FREIGHT-Modus genannt.

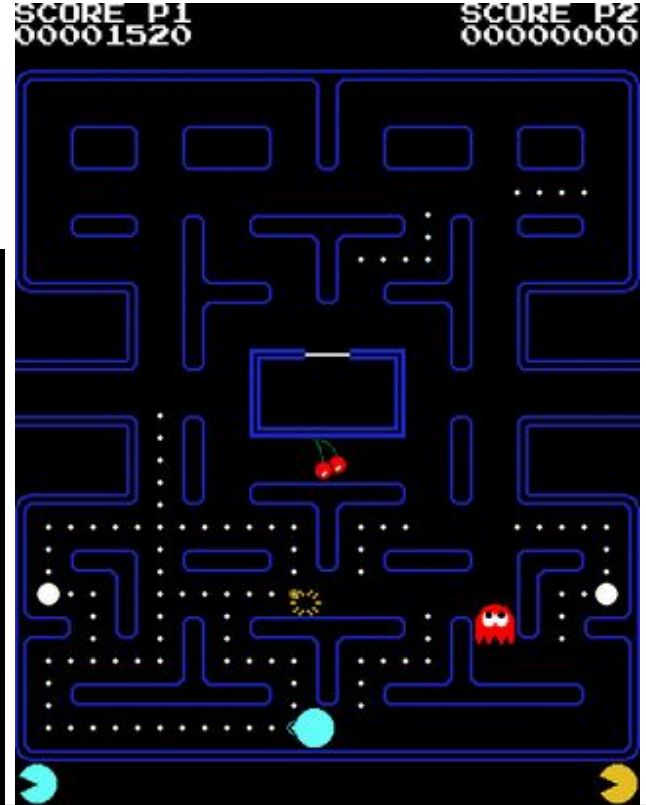
```
if d2ghost < 10:
    self.goal = ghosts.blinky.position
    directions = self.validDirections()
    directions = self.goalDirection(directions, minimize=False)
elif len(pellets.pelletList) > 0:
    pelletDists = []
    for pellet in pellets.pelletList:
        pelletDists.append(self.tileDistance(pellet))
    mindist = min(pelletDists)
    index = pelletDists.index(mindist)
    self.goal = pellets.pelletList[index].position
    directions = self.validDirections()
    directions = self.goalDirection(directions)
```



Pacmans Strategie

Wir können Pacman sagen, dass er dem Geist nur ausweichen soll, wenn er sich nicht in diesem Modus befindet!

```
if d2ghost < 10 and ghosts.blinky.mode.current != FREIGHT:
    self.goal = ghosts.blinky.position
    directions = self.validDirections()
    directions = self.goalDirection(directions, minimize=False)
elif len(pellets.pelletList) > 0:
    pelletDists = []
    for pellet in pellets.pelletList:
        pelletDists.append(self.tileDistance(pellet))
    mindist = min(pelletDists)
    index = pelletDists.index(mindist)
    self.goal = pellets.pelletList[index].position
    directions = self.validDirections()
    directions = self.goalDirection(directions)
```



Advanced: Scoring Directions

```
def move(self, opponent, pellets, fruit, ghosts):  
    d2ghost = self.tileDistance(ghosts.blinky)  
    scores = {UP: 0, DOWN: 0, RIGHT: 0, LEFT: 0}  
  
    ...  
  
    direction = max(scores, key=scores.get)  
    return direction
```



Advanced: Scoring Directions

Hier erstellen wir ein
Dictionary mit Bewertungen
für jede Richtung

```
def move(self, opponent, pellets, fruit, ghosts):  
    d2ghost = self.tileDistance(ghosts.blinky)  
    scores = {UP: 0, DOWN: 0, RIGHT: 0, LEFT: 0}  
  
    ...  
  
    direction = max(scores, key=scores.get)  
    return direction
```

Advanced: Scoring Directions

Hier erstellen wir ein Dictionary mit Bewertungen für jede Richtung

Am Ende bewegen wir uns in die Richtung mit der höchsten Punktzahl. `key=scores.get` erhält den Schlüssel anstelle des Wertes.

```
def move(self, opponent, pellets, fruit, ghosts):  
    d2ghost = self.tileDistance(ghosts.blinky)  
    scores = {UP: 0, DOWN: 0, RIGHT: 0, LEFT: 0}  
  
    ...  
  
    direction = max(scores, key=scores.get)  
    return direction
```

Advanced: Scoring Directions

Wenn der Geist in der Nähe ist, gibt es 10 Punkte, wenn du dich entfernst.

```
...  
  
    if d2ghost < 10:  
        self.goal = ghosts.blinky.position  
        directions = self.validDirections()  
        if ghosts.blinky.mode.current == FREIGHT:  
            directions = self.goalDirection(directions)  
            for d in directions:  
                scores[d] += 20  
        else:  
            directions = self.goalDirection(directions,  
minimize=False)  
            for d in directions:  
                scores[d] += 10  
    ...
```

Advanced: Scoring Directions

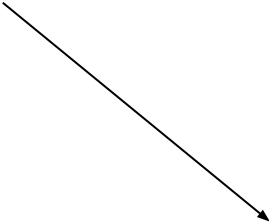
Aber wenn der Geist im FREIGHT-Modus ist, gibt es 20 Punkte, wenn er sich TOWARD bewegt.

Wenn der Geist in der Nähe ist, gibt es 10 Punkte, wenn du dich entfernst.

```
...  
  
    if d2ghost < 10:  
        self.goal = ghosts.blinky.position  
        directions = self.validDirections()  
        if ghosts.blinky.mode.current == FREIGHT:  
            directions = self.goalDirection(directions)  
            for d in directions:  
                scores[d] += 20  
        else:  
            directions = self.goalDirection(directions,  
minimize=False)  
            for d in directions:  
                scores[d] += 10  
    ...
```

Advanced: Scoring Directions

Eine Bewegung in
Richtung Pellets ist
immer 5 Punkte wert.



```
...  
    if len(pellets.pelletList) > 0:  
        pelletDists = []  
        for pellet in pellets.pelletList:  
            pelletDists.append(self.tileDistance(pellet))  
        mindist = min(pelletDists)  
        index = pelletDists.index(mindist)  
        self.goal = pellets.pelletList[index].position  
        directions = self.validDirections()  
        directions = self.goalDirection(directions)  
        for d in directions:  
            scores[d] += 5  
...
```


Advanced: Scoring Directions

Beachte, dass dies "if"- und nicht "else"- oder "elif"-Anweisungen sind. Der Grund dafür ist, dass wir wollen, dass die Bewertung alle relevanten Informationen kombiniert

```
def move(self, opponent, pellets, fruit, ghosts):  
    d2ghost = self.tileDistance(ghosts.blinky)  
    scores = {UP: 0, DOWN: 0, RIGHT: 0, LEFT: 0}  
  
    if d2ghost < 10:  
        ...  
  
    if len(pellets.pelletList) > 0:  
        ...  
  
    direction = max(scores, key=scores.get)  
    return direction
```

Weitere Optionen

- Kann auf `fruit.position` zugreifen, wenn `fruit` existiert (if `fruit:`)
- Kann auf `opponent.position` und `opponent.alive` zugreifen
- Kann auf `pellets.powerpellets` zugreifen
- Mehrere Timer verwenden: `self.dt[1]`, bezieht sich auf Timer 1, usw.
- Erstelle deine eigenen "Modi" für Pacman

```
def  init  (self, node, playerNum):  
    color = TEAL  
    Pacman.  init  (self, node, color, playerNum)  
    self.mode = 'powerpellets'
```

`self.mode` kann dann in `move()` aufgerufen und in der eigenen Logik verwendet werden