Python Programmierkurs
Sommersemester 2025
Lehrstuhl für Produktions- und Logistikplanung
Betreuung: Prof. Dr. Rouven Schur
Wissenschaftliche Unterstützung: Kai Winheller

Entwicklung von Preisalgorithmen für einen simulierten Markt

Teilnehmerinnen:	**Matrikulationsnummer:**
Ayda Beiram Zadeh	3207263
Melisa Mavric	3214382

Inhaltsverzeichnis

1. Einleitung

Im Rahmen des Masterseminars „Dynamic Pricing“ wurden zwei Preisalgorithmen entwickelt, die in unterschiedlichen Marktkonstellationen agieren: ein **Oligopol-Bot**, der sich gegen konkurrierende Anbieter behaupten muss, und ein **Monopol-Bot**, der als alleiniger Anbieter auftritt. Beide Algorithmen operieren unter vergleichbaren Unsicherheiten: Die **Spieldauer ist nicht exakt bekannt** – sie liegt lediglich im Bereich einer **dreistelligen Rundenzahl** – und der **verfügbare Lagerbestand** ist begrenzt. Diese Rahmenbedingungen erfordern eine flexible und anpassungsfähige Preisstrategie.

Während der Oligopol-Bot neben der Marktnachfrage auch das Verhalten der **Konkurrenz** beobachten und antizipieren muss, richtet sich der Monopol-Bot ausschließlich nach dem **Zahlungsverhalten der Kundschaft**. In beiden Fällen liegt der Fokus auf der **Gewinnmaximierung durch dynamische Preissteuerung**, wobei unterschiedliche Mechanismen zur Anwendung kommen – etwa **gegnerorientierte Reaktion beim Oligopol-Bot** oder **Zufallspreise zur Margenabschöpfung im Monopolmodell**.

Diese Analyse beleuchtet die **strategischen Überlegungen**, die **algorithmische Umsetzung in Python** sowie die **Anpassung an Marktunsicherheit** in beiden Marktformen.

2. Strategisches Vorgehen

2.1 Oligopol-Bot

Die strategische Ausrichtung des entwickelten Oligopol-Bots orientiert sich an den typischen Rahmenbedingungen eines oligopolistischen Marktes mit limitiertem Lagerbestand und begrenztem Zeithorizont. Da die genaue Anzahl der Runden im Vorfeld nicht bekannt ist, wird von einem endlichen, aber variabel langen Entscheidungszeitraum ausgegangen. Daraus ergeben sich mehrere zentrale Handlungsfelder, die das strategische Verhalten des Bots strukturieren:

- **Frühe Explorationsphase zur Markterkundung**: In den initialen Runden wird bewusst auf festgelegte Testpreise zurückgegriffen, um die Preisbereitschaft der Nachfrageseite sowie das Verhalten der Konkurrenz zu sondieren.
- **Adaptive Preissteuerung auf Basis vergangener Verkaufszahlen**: Der Bot passt seine Preisstrategie dynamisch an, indem er den Absatz der letzten Runden auswertet und daraus Handlungsempfehlungen ableitet – etwa zur Preissenkung bei ausbleibenden Verkäufen oder zur Preiserhöhung bei stabiler Nachfrage.

- **Analyse gegnerischen Preisverhaltens**: Durch die Auswertung vergangener Preissetzungen der Konkurrenz identifiziert der Bot sogenannte Undercutter – also Mitbewerber, die systematisch niedrigere Preise anbieten.
- **Druckbasierte Entscheidungslogik („Verkaufsdruck“) :** Als zentrales Steuerungskriterium wird das Verhältnis von Lagerbestand zu verbleibender Rundenzahl berechnet. Dieses Verhältnis bestimmt die Dringlichkeit, mit der Verkäufe erzielt werden müssen.
- **Gezielte Preissignale durch Bluffstrategien**: In Phasen geringer Verkaufsdringlichkeit wird mit Hilfe einer Zufallskomponente temporär ein erhöhter Preis gesetzt, um höhere Zahlungsbereitschaften abzuschöpfen oder das Verhalten potenzieller Copycats zu beeinflussen.
- **Absicherung durch Preisband**: Jede Preissetzung wird durch eine Min-Max-Klemme auf ökonomisch sinnvolle Grenzen begrenzt.

2.2 Monopol-Bot

Die Preisstrategie des Monopol-Bots folgt grundsätzlich einem ähnlichen Aufbau, unterscheidet sich jedoch durch die Abwesenheit von Konkurrenz. Die strategischen Maßnahmen zielen ausschließlich auf die Optimierung der Nachfrageabschöpfung ab:

- **Explorative Markteinstiegsphase**: Wie beim Oligopol-Bot nutzt der Monopol-Bot die ersten drei Runden zur Marktsondierung mithilfe fester Testpreise. Ziel ist es, ein erstes Gefühl für die Zahlungsbereitschaft der Kundschaft zu entwickeln.
- **Verkaufsbasierte Preissteuerung**: Der Bot analysiert die Verkaufszahlen der letzten Perioden und passt seine Preisstrategie entsprechend an. Bleiben Verkäufe aus, erfolgt eine Preissenkung; bei stabiler Nachfrage wird der Preis schrittweise erhöht.
- **Bluff-Komponente bei geringem Verkaufsdruck**: In Zeiten niedriger Verkaufsdringlichkeit wird mit einer definierten Wahrscheinlichkeit ein überhöhter Preis gesetzt, um potenziell höhere Erträge zu erzielen.
- **Preisband zur Stabilisierung**: Auch der Monopol-Bot nutzt eine Min-Max-Klemme zur Begrenzung von Preissprüngen nach oben und unten.

Im Unterschied zum Oligopol verzichtet der Monopol-Bot vollständig auf die Beobachtung von Mitbewerbern und deren Preissetzung – sein Fokus liegt ausschließlich auf dem eigenen Verkaufsverhalten und der daraus ableitbaren Nachfrageprognose.

3. Dynamik des Verkaufsdrucks

Ein zentrales Steuerungselement im Entscheidungsmodell des Bots ist der sogenannte **Verkaufsdruck**, welcher als Quotient aus dem aktuellen Lagerbestand und der erwarteten verbleibenden Spielzeit verstanden wird. Dieser Indikator dient dazu, die Dringlichkeit verkaufsfördernder Maßnahmen abzubilden und die Preisstrategie entsprechend auszurichten.

Da die tatsächliche Anzahl verbleibender Runden im Wettbewerb nicht exakt bekannt ist – es wird lediglich ein dreistelliger Bereich (100–199 Runden) angenommen – wird zur Erhöhung der Robustheit ein konservativer Sicherheitsaufschlag in Höhe von zehn Runden berücksichtigt. Die operative Formel zur Bestimmung des Verkaufsdrucks lautet entsprechend:

Verkaufsdruck = Lagerbestand / (Verbleibende Runden + 10)

Ein **hoher Verkaufsdruck** (z. B. bei großem Lagerbestand und geringem verbleibendem Zeitraum) indiziert eine unmittelbare Verkaufsnotwendigkeit. In diesem Fall reagiert der Bot mit einer Preissenkung, um den Absatz zu beschleunigen. Ein **niedriger Verkaufsdruck** hingegen erlaubt ein strategisch defensiveres Vorgehen, bei dem Preissteigerungen oder gezielte Bluffpreise genutzt werden können, um höhere Zahlungsbereitschaften der Nachfrager abzuschöpfen.

Diese Steuergröße fungiert als **zentrales Bindeglied zwischen Marktdynamik, Lagerplanung und Preisstrategie**. Sie ermöglicht beiden Algorithmen, situativ angemessen auf unterschiedliche Spielsituationen zu reagieren und ihre Preislogik flexibel auszurichten.

4. Algorithmische Umsetzung

4.1 Oligopol Bot

Die Implementierung der Preisentscheidungslogik erfolgt auf Grundlage eines regelbasierten Algorithmus, der mehrere Informationsquellen kombiniert und verschiedene Reaktionsmechanismen integriert. Die Struktur des Algorithmus folgt dabei einem deterministischen Hauptpfad, ergänzt um eine probabilistische Komponente zur Erhöhung der strategischen Varianz. Die zentralen algorithmischen Elemente lassen sich wie folgt systematisieren:

- **Regelbasierte Entscheidungslogik**: Die Kernstruktur basiert auf verschachtelten *if-else*-Konditionen, welche die Preisentscheidung anhand definierter Schwellenwerte für Absatz, Gegnerverhalten und Verkaufsdruck steuern.
- **Historienbasierte Aggregation des Verkaufsverhaltens**: Die vergangenen drei Verkaufsperioden werden herangezogen, um den kumulierten Absatz des Bots zu ermitteln. Diese Aggregation dient als Proxy für die aktuelle Marktnachfrage und wird

zur Ableitung von Preisbewegungen genutzt (z. B. Preisanhebung bei stabiler Nachfrage).
- **Analyse gegnerischen Preisverhaltens**: Über die letzten fünf Perioden hinweg wird geprüft, welche Wettbewerber systematisch niedrigere Preise setzen (sog. Undercutter). Diese Information beeinflusst die Reaktion des Bots, indem er in solchen Fällen gezielt unterbietet, um Marktanteile zu sichern.
- **Stochastische Bluff-Komponente**: Bei geringem Verkaufsdruck wird mit einer festgelegten Wahrscheinlichkeit von 10 % ein künstlich erhöhter Preis gesetzt (sog. *Bluff*), um mögliche höhere Zahlungsbereitschaften der Kundschaft

auszunutzen oder das Verhalten der Konkurrenz zu beeinflussen.
• **Begrenzung durch Preisband**: Zur Sicherstellung ökonomisch sinnvoller

Preissetzungen wird jede Entscheidung am Ende durch eine Preisgrenze kontrolliert. Hierzu wird die folgende Min-Max-Klemme implementiert:

$$\text{price} = \text{round}(\max(\min(\text{new\_price}, \text{MAX\_PRICE}), \text{MIN\_PRICE}), 2)$$

Diese Sicherungslogik verhindert sowohl ungewollte Preisabstürze unterhalb des Mindestniveaus als auch strategisch sinnlose Überhöhungen.

In Summe ermöglicht die algorithmische Architektur eine adaptive, datengestützte und dennoch nachvollziehbare Preissteuerung, die sowohl auf kurzfristige Marktveränderungen als auch auf strategische Langfristeffekte reagieren kann.

4.2 Monopol Bot

Die algorithmische Architektur des Monopol-Bots lehnt sich strukturell stark an jene des Oligopol-Bots an: Auch hier erfolgt die Preisentscheidung über eine **regelbasierte Logik**, die Absatzverhalten, Verkaufsdruck und Zufallselemente berücksichtigt.

Anders als im Oligopol-Modell verzichtet der Monopol-Bot jedoch vollständig auf die Analyse gegnerischen Preisverhaltens. Stattdessen richtet er seinen Fokus ausschließlich auf die **eigene Verkaufsperformance** der letzten Perioden. Die Bewertung vergangener Absatzmengen dient dabei als Grundlage für dynamische Preisbewegungen:

- **Absatzbasierte Preisanpassung:** Je nach Verkaufsverlauf der letzten Runden wird der Preis gesenkt, erhöht oder stabil gehalten.

Auch die **Bluff-Komponente** ist analog implementiert – sie wird bei geringem Verkaufsdruck mit einer festgelegten Wahrscheinlichkeit von 10 % aktiviert, um temporär höhere Preise durchzusetzen.

Der Verkaufsdruck selbst wird identisch berechnet wie im Oligopol-Bot, wirkt jedoch rein intern, da externe Marktreaktionen keine Rolle spielen.

Insgesamt stellt der Monopol-Bot eine **vereinfachte, jedoch wirkungsvolle Ausprägung** derselben Entscheidungsarchitektur dar – angepasst an ein Marktumfeld ohne Wettbewerbseinflüsse.

5. Explorationsphase

Beide entwickelten Bots setzen zu Beginn des Spiels auf eine strukturierte **Explorationsphase**, in der bewusst vordefinierte Preise verwendet werden, um Informationen über das Marktumfeld zu sammeln. Diese Phase erstreckt sich über die ersten drei Perioden und dient als Grundlage für die nachfolgenden adaptiven Preisentscheidungen.

5.1 Oligopol-Bot

Im Fall des Oligopol-Bots verfolgt die Explorationsphase mehrere Ziele:

- **Marktsondierung**: Durch konstante Anfangspreise können erste Erkenntnisse über die allgemeine Preisbereitschaft der Kundschaft gewonnen werden.
- **Konkurrenzbeobachtung**: Da die eigenen Preise in dieser Phase stabil bleiben, lassen sich Rückschlüsse auf das Verhalten der Mitbewerber ziehen.
- **Baseline-Kalibrierung**: Die gesammelten Informationen werden zur Justierung der Preislogik in den Folgerunden verwendet.

Die eingesetzten Testpreise bewegen sich in einem moderaten Rahmen, um Verkaufschancen zu erhalten, aber auch unterschiedliche Preisniveaus abzutasten.

5.2 Monopol-Bot

Auch der Monopol-Bot beginnt mit einer dreirundigen Explorationsphase, setzt jedoch den Fokus ausschließlich auf die **Nachfrageseite**. Da kein Wettbewerb existiert, besteht das primäre Ziel darin, die **Zahlungsbereitschaft der Konsumenten** zu erfassen.

- **Absatzbeobachtung bei festem Preisniveau**: Durch das gezielte Variieren von Einstiegspreisen kann der Bot ableiten, in welchen Preisbereichen eine ausreichende Nachfrage besteht.
- **Grundlage für spätere Dynamik**: Die gesammelten Informationen bilden das Fundament für die spätere adaptive Preissetzung.

Auch hier gilt: Die gewählten Preise sind so gewählt, dass sie realistische Verkäufe ermöglichen, gleichzeitig aber unterschiedliche Reaktionen der Kundschaft erfassbar machen.

6. Bewertung und Robustheit

6.1 Oligopol Bot

Die Simulationsergebnisse verdeutlichen die hohe Anpassungsfähigkeit und Stabilität des entwickelten Oligopol-Bots über ein breites Spektrum möglicher Marktparameter hinweg. Insbesondere in Szenarien mit hoher Unsicherheit – etwa bezüglich der tatsächlichen Anzahl an Spielperioden oder der Höhe des initialen Lagerbestands – zeigt sich die Stärke der entwickelten Entscheidungslogik. Während viele einfache Preisalgorithmen entweder starr oder ausschließlich kurzfristig opportunistisch agieren, verfolgt der hier vorgestellte Bot einen integrativen Ansatz: Durch die dynamische Berücksichtigung des Verkaufsdrucks sowie die flexible Reaktion auf Absatzhistorie und Konkurrenzverhalten gelingt es, situativ angemessene Preisanpassungen vorzunehmen.

Besonders hervorzuheben ist die Robustheit gegenüber Veränderungen in zwei kritischen Parametern: Zum einen reagiert der Bot stabil auf variierende Startinventare, da er seine Verkaufsdringlichkeit nicht absolut, sondern relativ zum verbleibenden Zeithorizont bewertet. Zum anderen zeigt sich, dass der Bot in kürzeren wie längeren Simulationsläufen konkurrenzfähig bleibt, da die algorithmische Architektur bewusst keine lineare Zeitsensitivität aufweist, sondern kontinuierlich zwischen Lagerrisiko und Preisspielraum abwägt.

Darüber hinaus beweist der Bot Resilienz gegenüber aggressiven Mitbewerbern mit systematisch niedriger Preisstrategie (sog. Undercutter), indem er deren Verhalten erkennt und situativ unterbietet, ohne sich dauerhaft auf ruinöse Preispfade einzulassen.

Gleichzeitig schützt die implementierte Min-Max-Klemme vor ökonomisch sinnlosen Preissprüngen. Insgesamt stellt der Bot ein Beispiel für strategische Robustheit im Spannungsfeld zwischen Flexibilität, Marktwettbewerb und Risikomanagement dar.

6.2 Monopol-Bot

Der Monopol-Bot zeigte ebenfalls eine bemerkenswerte **Anpassungsfähigkeit und Stabilität** gegenüber unterschiedlichen Nachfrageprofilen. Insbesondere in Szenarien mit gleich- oder normalverteilten Zahlungsbereitschaften erwies sich die **regelbasierte Entscheidungslogik** als tragfähig: Absatz und Ertrag blieben auch unter variierenden Rahmenbedingungen auf konstantem Niveau.

Die algorithmische Struktur reagiert flexibel auf Absatzschwankungen und kombiniert **verkaufsbasierte Preissteuerung**, **Verkaufsdruckanalyse** und eine **Preisbandbegrenzung**, um unter Unsicherheit **wirtschaftlich sinnvolle Preisentscheidungen** zu treffen. Die Min-Max-Klemme stellt dabei sicher, dass weder unökonomisch niedrige noch überhöhte Preise gesetzt werden.

Im Gegensatz zum Oligopol-Bot, der zusätzlich externe Preissignale berücksichtigt, basiert die Entscheidungslogik im Monopol-Modell ausschließlich auf interner Absatzbeobachtung. Dieser Fokus auf endogene Datenquellen führt zu einer **vereinfachten, aber effektiven** Steuerung, die in den Simulationen durchweg konsistente Ergebnisse erzielte.

Insgesamt bestätigt die Simulationsauswertung, dass auch im Monopolumfeld eine **strategisch belastbare und adaptive Preispolitik** möglich ist, selbst ohne Wettbewerbseinflüsse. Der Bot beweist damit, dass robuste Preisgestaltung nicht zwingend komplexe externe Analysen voraussetzt, sondern auch mit reduzierter Informationslage erzielt werden kann

7. Fazit

Die im Rahmen des Projekts entwickelten Preisalgorithmen für simulierte Oligopol- und Monopolmärkte verdeutlichen, dass auch auf Basis deterministischer Regelwerke **adaptive und robuste Preispolitiken** gestaltet werden können. Beide Bots wurden so konzipiert, dass sie unter Marktunsicherheit und begrenztem Lagerbestand **situativ sinnvolle Entscheidungen** treffen und dabei ein Gleichgewicht zwischen Absatz und Ertrag wahren.

7.1 Oligopol-Bot

Der Oligopol-Bot kombiniert **interne Absatzbeobachtung** mit **externer Marktbeobachtung**, insbesondere der Preisstrategien der Mitbewerber. Dadurch kann er gezielt auf Unterbietungen reagieren, Marktanteile verteidigen und taktisch über Bluffpreise agieren. Die dynamische Einbindung des Verkaufsdrucks ermöglicht

eine kontinuierliche Anpassung an Spielverlauf und Restlaufzeit, während eine implementierte **Min-Max-Klemme** ökonomisch sinnvolle Preisgrenzen sichert.

Besonders hervorzuheben ist die **strategische Robustheit** des Bots: Er reagiert stabil auf variierende Lagerbestände, Spielzeiten und aggressives Wettbewerbsverhalten. Dabei gelingt es ihm, sowohl kurzfristige Preisanpassungen als auch langfristige Spielstrategien miteinander zu verknüpfen. Insgesamt beweist der Bot eine hohe **Flexibilität und Wettbewerbsfähigkeit** – sowohl in stabilen als auch in volatilen Marktumfeldern.

7.2 Monopol-Bot

Der Monopol-Bot basiert auf einer **vereinfachten, intern gesteuerten Entscheidungsarchitektur**, die sich ausschließlich an Absatzverlauf, Verkaufsdruck und Lagerstand orientiert. Trotz dieser reduzierten Informationslage erzielt der Bot in unterschiedlichen Nachfrageverteilungen – insbesondere bei gleich- und normalverteilten Zahlungsbereitschaften – **stabile Absatz- und Ertragswerte**.

Bereits in der Anfangsphase nutzt der Bot eine **explorative Preisstrategie**, um die Zahlungsbereitschaft der Kundschaft systematisch zu testen. Die anschließende Preissteuerung erfolgt über eine Kombination aus **historienbasierter Verkaufsanalyse**, **Verkaufsdruckbewertung** und einem durch **Min-Max-Klemme** abgesicherten Preisband. Ergänzt wird diese Logik durch eine **stochastische Bluff-Komponente**, die bei geringem Verkaufsdruck höhere Preissignale erlaubt, ohne die Stabilität zu gefährden.

Die Simulationsergebnisse zeigen, dass der Monopol-Bot auch ohne externe Datenströme eine **strategisch tragfähige und ertragsorientierte Preisstrategie** realisieren kann. Gerade durch seine Klarheit und Regelgebundenheit bietet er ein **robustes Modell zur Preissteuerung in Märkten ohne Wettbewerbseinflüsse**.

7.3 Ausblick

Trotz der überzeugenden Ergebnisse zeigt sich, dass regelbasierte Systeme – so effizient sie unter klaren Rahmenbedingungen auch sind – an ihre Grenzen stoßen, wenn Marktstrukturen komplexer, Kundenverhalten heterogener oder Reaktionen kurzfristiger werden.

Eine Weiterentwicklung der Bots durch **maschinelle Lernverfahren**, insbesondere **Reinforcement Learning**, könnte es ermöglichen, Preispfade nicht nur regelbasiert zu optimieren, sondern selbstständig zu erlernen. So ließen sich beispielsweise

Kundensegmente mit unterschiedlichen Preisreaktionen besser bedienen oder **langfristige Ertragsziele** gegen kurzfristige Absatzmaximierung abwägen.

Auch in realwirtschaftlichen Szenarien – etwa im **E-Commerce** oder in **simulationsgestützten Preisstrategien großer Online-Plattformen** – bieten solche adaptiven Systeme ein hohes Potenzial. Die in diesem Projekt entwickelte Entscheidungsarchitektur kann hierfür als **robuste Grundlage** dienen, die mit datengetriebenen Elementen gezielt erweitert werden kann.

8. Kommentierter Code

8.1 Oligopol-Bot

```
# ------------------------------------------------------------
# Adaptive Oligopol Bot (Kommentierte Version)
# Ziel: Robuste Preissetzung im Oligopol bei unbekannter Rundenanzahl & Lager
# Strategien:
# - Exploratives Verhalten zu Beginn
# - Analyse vergangener Verkäufe zur Preissteuerung
# - Gegnerbeobachtung zur Reaktion auf Unterbietung
# - Anpassung an Verkaufsdruck (Bestand/Runden)
# - Bluff-Komponente bei geringem Druck
# ------------------------------------------------------------

import random  # Zufallskomponente für Bluffing

def price(market_data):
    # === Konfigurierbare Parameter ===
    DEFAULT_PRICE = 36.0
    MIN_PRICE = 30.0
    MAX_PRICE = 100.0
    EXPLORATION_PRICES = [34.99, 37.99, 40.99]  # Testpreise zu Beginn
    UNDERCUT_STEP = 0.5       # Abstand zur aktiven Unterbietung
    BLUFF_DELTA = 3.0         # Zuschlag bei Bluff
    BLUFF_CHANCE = 0.1        # Wahrscheinlichkeit eines Bluffs (10%)
    HISTORY_DEPTH = 5         # Tiefe der Gegneranalyse (5 Perioden)
    SALE_WINDOW = 3           # Fenster für Verkaufsanalyse (3 Perioden)

    # === 1. Sonderfall: Erste Runde ===
    if not market_data:
        return DEFAULT_PRICE

    # === 2. Letzte Rundeninformationen extrahieren ===
    last_round = market_data[-1]
```

```
    period = last_round.get("period", 0)
    remaining = last_round.get("remaining_rounds", 100)
    inventory = last_round.get("own_inventory", 20)
    own_price = last_round.get("own_price", DEFAULT_PRICE)
    own_sales = last_round.get("own_sold", 0)

    # === 3. Gegnerpreise auslesen ===
    opponent_prices = []
    opponent_names = []
    for key in last_round:
        if key.endswith("_price") and not key.startswith("own"):
            opponent_prices.append(last_round[key])
            opponent_names.append(key.replace("_price", ""))

    min_opp_price = min(opponent_prices, default=DEFAULT_PRICE)
    avg_opp_price = sum(opponent_prices) / len(opponent_prices) if opponent_prices
else DEFAULT_PRICE

    # === 4. Explorationsphase in den ersten drei Runden ===
    if period < len(EXPLORATION_PRICES):
        return EXPLORATION_PRICES[period]

    # === 5. Eigene Verkäufe der letzten 3 Runden analysieren ===
    recent_sales = [r.get("own_sold", 0) for r in market_data[-SALE_WINDOW:]]
    sale_sum = sum(recent_sales)

    # === 6. Gegnerverhalten analysieren (Undercutter identifizieren) ===
    undercutters = set()
    for r in market_data[-HISTORY_DEPTH:]:
        my_p = r.get("own_price", DEFAULT_PRICE)
        for name in opponent_names:
            opp_p = r.get(f"{name}_price", DEFAULT_PRICE)
            if opp_p < my_p:
                undercutters.add(name)

    # === 7. Verkaufsdruck berechnen: Lager / (Restzeit + Sicherheitsaufschlag) ===
    adjusted_remaining = max(remaining + 10, 1)  # +10 zur Sicherheit
    pressure = inventory / adjusted_remaining

    # === 8. Preisfindung basierend auf Verkaufszahlen und Gegnern ===
    new_price = own_price

    if sale_sum == 0:
        # Wenn keine Verkäufe → aggressiv unterbieten
        new_price = max(min_opp_price - UNDERCUT_STEP, MIN_PRICE)
```

```
    elif sale_sum >= 2:
        # Gute Verkäufe → leicht erhöhen
        new_price = min(own_price + 1.0, MAX_PRICE)
    else:
        # Mittlere Verkäufe → Gegner beobachten
        if len(undercutters) >= 2:
            # Viele Gegner unterbieten → unterbieten
            new_price = min_opp_price - 0.01
        else:
            # Wenige Gegner → leicht über Marktpreis
            new_price = avg_opp_price + 0.5

    # === 9. Bluff-Komponente bei geringem Verkaufsdruck ===
    if pressure < 0.7 and random.random() < BLUFF_CHANCE:
        new_price += BLUFF_DELTA

    # === 10. Weitere Anpassung bei extremem Verkaufsdruck ===
    if pressure > 1.6:
        new_price -= 2.0  # Hoher Druck → stark senken
    elif pressure < 0.5:
        new_price += 0.5  # Geringer Druck → leicht erhöhen

    # === 11. Preis runden und im gültigen Bereich halten ===
    return round(max(min(new_price, MAX_PRICE), MIN_PRICE), 2)
```

8.2 Monopol Bot

```
# ------------------------------------------------------------
# Adaptive Monopol-Bot – Kommentierte Version
# Ziel: Robuste Preissetzung im Monopol bei unsicherer Nachfrage und Spielzeit
# Strategien:
# - Exploratives Verhalten alle 7 Runden zur Preisvariation
# - Direkte Reaktion auf Verkaufserfolg oder -misserfolg
# - Mittelfristige Anpassung durch Verkaufsquote (10 Perioden)
# - Dynamische Preisanpassung durch Verkaufsdruck (Lager / verbleibende Runden)
# - Bluff-Komponente bei geringem Verkaufsdruck
# - Preisband zur Sicherung wirtschaftlich sinnvoller Preisgrenzen
# ------------------------------------------------------------

def price(market_data):
    import random  # Für die Zufallskomponente bei Exploration und Bluffing

    # === Schritt 1: Erste Runde – Startwert setzen ===
```

```
if len(market_data) == 0:
    return 60.0  # Konservativer Startpreis bei Spielbeginn

# === Schritt 2: Teamnamen aus den Spalten erkennen ===
keys = market_data[0].keys()
my_team = [k[:-6] for k in keys if k.endswith('_price')]
my_team = my_team[0] if my_team else 'my_team'

# === Schritt 3: Vergangene Preise und Verkäufe extrahieren ===
prices = []
sold = []

for entry in market_data:
    price = entry.get(f'{my_team}_price', None)
    quantity = entry.get(f'{my_team}_quantity', None)
    if price is not None and quantity is not None:
        prices.append(price)
        sold.append(quantity)

# === Schritt 4: Aktuelle Spielinformationen ===
t = len(prices)                  # Anzahl vergangener Runden
current_price = prices[-1]       # Letzter gesetzter Preis
current_quantity = sold[-1]      # Verkaufsergebnis der letzten Runde

# === Schritt 5: Preisgrenzen definieren ===
max_price = 150.0  # Ökonomisch maximale Obergrenze
min_price = 1.0    # Mindestpreis zur Verlustvermeidung

# === Schritt 6: Verkaufsquote (letzte 10 Runden) als Erfolgsindikator ===
success_rate = sum(sold[-10:]) / min(10, len(sold))

# === Schritt 7: Reaktion auf letzte Runde ===
if current_quantity == 1:
    # Wenn verkauft wurde → Preis etwas erhöhen
    next_price = current_price + 2.0
else:
    # Kein Verkauf → Preis war zu hoch → deutlich senken
    next_price = current_price - 3.0

# === Schritt 8: Explorative Preisvariation alle 7 Runden ===
if t % 7 == 0:
    # Zufällige Preisabweichung zum Austesten neuer Nachfragebereiche
    next_price = current_price + random.uniform(-10, 10)

# === Schritt 9: Verkaufsquote nutzen zur Feinsteuerung ===
```

```
if success_rate < 0.4:
    # Dauerhaft schlechte Verkäufe → stärker senken
    next_price -= 5.0
elif success_rate > 0.8:
    # Sehr gute Verkaufsquote → stärker erhöhen
    next_price += 3.0

# === Schritt 10: Verkaufsdruck berechnen und anwenden ===
remaining_rounds = market_data[-1].get('remaining_rounds', 100)
inventory = market_data[-1].get(f'{my_team}_inventory', 100)
urgency_factor = inventory / max(1, remaining_rounds)
# Verkaufsdruck = Lagerbestand geteilt durch verbleibende Zeit

if urgency_factor > 1.2:
    # Viel Lager, wenig Zeit → Preis senken
    next_price *= 0.95
elif urgency_factor < 0.5:
    # Wenig Lager, viel Zeit → Preis leicht erhöhen
    next_price *= 1.05

# === Schritt 11: Bluff-Komponente bei geringem Druck ===
if urgency_factor < 0.7 and random.random() < 0.1:
    # Bei niedrigem Druck → 10 % Wahrscheinlichkeit für Bluff
    next_price += 5.0  # Bluffaufschlag

# === Schritt 12: Preisband zur Sicherung wirtschaftlicher Grenzen ===
next_price = min(max(next_price, min_price), max_price)  # Klemme anwenden
return round(next_price, 2)  # Ergebnis auf 2 Nachkommastellen runden
```