מבוא לבינה מלאכותית

תרגיל בית 2

שם סטודנט: אברהאם סעיד

תעודת זהות: 209112036

שם סטודנט: עומר מחאמיד

תעודת זהות: 308198134

חלק א - $ImprovedGreedy$:

1. $Game = \ < S,\ A,\ f,\ c,\ s_0,\ R >$:

- ❖ $S = State\ Space$, קבוצת המצבים: כל מטריצה בגודל $5x5$ שיש בה 2 רובוטים, 2 תחנות דלק, 2 חבילות ו- 2 יעדים, עם האפשרות שבאותה משבצת יכולים להיות יותר מפריט אחד ממה שהזכרנו.
- ❖ A קבוצת שחקנים: או קבוצת הפעולות שכל שחקן יכול לבצע:

$$A = \{A_1,\ A_2\}$$

$$\forall i \in \{1,2\}:\ A^i = \{move\ north,\ move\ south,\ move\ east,\ move\ west,\ pick\ up,\ drop\ off,\ charge\}$$

- ❖ f פונקצית המעברים: מחזירה את המצב המתקבל כתוצאה מביצוע פעולה $a \in A$ על מצב $s \in S$ אם היא אפשרית.
- ❖ c פונקצית המחיר: כמה משלמים כדי לבצע פעולה $a \in A$ על מצב $s \in S$.
- ❖ s_0 מצב התחלתי לכל $Agent$, $s_0 \in S$.
- ❖ R: כמה שחקן $Agent$ מרוויח כאשר מבצע פעולה $a \in A$ על מצב $s \in S$, נשים לב כי לכל שחקן פונקצית R משלו, ונסמן: $R = \{R_1, R_2\}$, ונגדיר:

$R(s,\ "drop\ off") =$ כמה נקודות מרוויחים כאשר מורידים חבילה ביעד

$R(s,\ "charge") =$ כמה נקודות סוללה מרוויחים

---

2. קודם כל נפריד בין שני מקרים:

- המצב הוא סופי:
  - אם ניצחנו זה מה שאנו רוצים לקבל לכן נחזיר ∞.
  - אם הפסדנו אז נחזיר $-\infty$.
  - אחרת, נחזיר 0.

- אחרת, כלומר המצב אינו סופי:

במצב הזה הרובוט יכול להיות בשני מצבים : מחזיק חבילה או שלא מחזיק. נזכיר כי בכל תנועה הוא מפסיד יחידת סוללה אחת. והוא מרוויח נקודת בהגעתו ליעד כמרחק בין החבילה ליעד כפול 2, נסמן את נקודות האלה ב- M. נסמן את הנקודות שלו ב- A.

  - אם הוא מחזיק את החבילה:

    * אם הדלק שלו מספיק לו להוביל את החבילה אל היעד אז נחזיר:

$$K \cdot A - (number\_of\_steps) + M$$

* אם הדלק לא מספיק אז:

  - אם הדלק שלו מספיק לו להגיע לתחנה הקרובה ביותר יחזיר:

    $$K \cdot A - (number\_of\_steps)$$

    כך ש- $number\_of\_steps$ הוא מה שמפסיד עד שיגיע לתחנה.

  - אחרת, נחזיר $-\infty$.

כך ש- K זה מספר גדול כדי לשמור על יוריסטיקה חיובית.

– אם הוא לא מחזיק חבילה:

* אם הדלק שלו מספיק לו להוביל את החבילה אל היעד אז נחזיר:

$$K \cdot A \ - \ (number\_of\_steps)$$

כך ש- K זה מספר גדול כדי לשמור על יוריסטיקה חיובית, $number\_of\_steps$ זה מספר הצעדים בינו לבין החבילה הקרובה ביותר אליו.

* אם הדלק לא מספיק אז:

  - אם הדלק שלו מספיק לו להגיע לתחנה הקרובה ביותר יחזיר:

    $$K \cdot A - (number\_of\_steps)$$

    כך ש- $number\_of\_steps$ הוא מה שמפסיד עד שיגיע לתחנה, K זה מספר גדול כדי לשמור על יוריסטיקה חיובית.

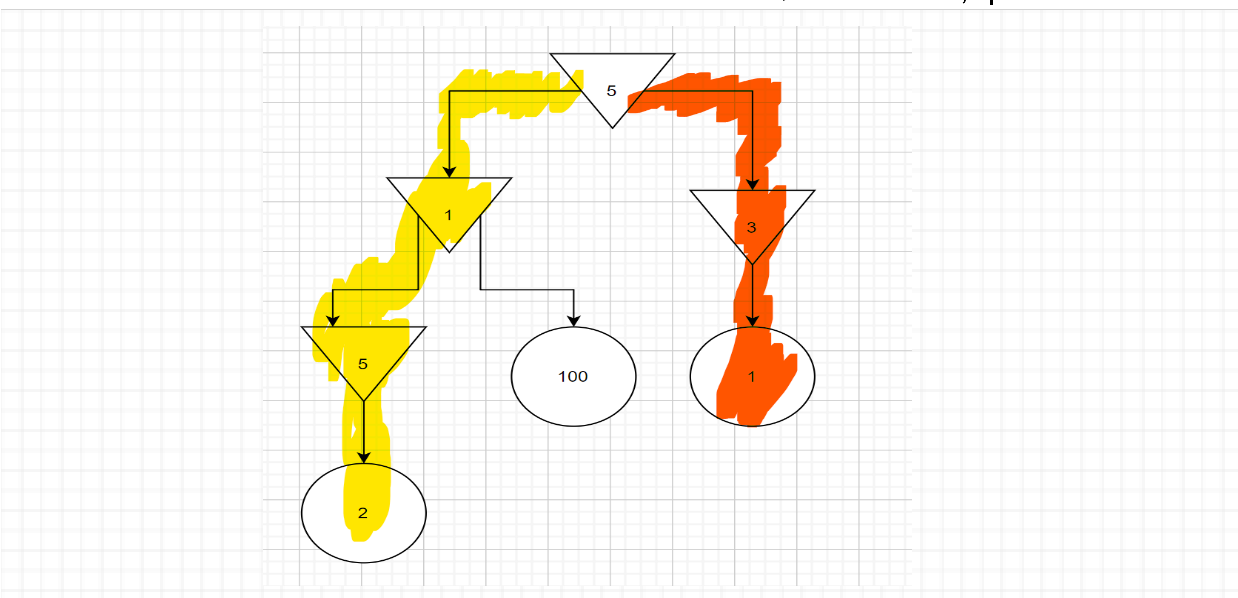
  - אחרת, נחזיר $-\infty$.

חלק ב - $RB - Minimax$:

1. נתחיל בכך שיוריסטיקה קלה לחישוב בודקים יותר אפשרויות כך שאנו שהבדיקה מהירה יותר ובכך בודקים יותר צמתים תחת אותה הגבלת זמן של יוריסטיקה קשה. כי יוריסטיקה קשה דורשת זמן יותר ומשאבים יותר לחישוב וזה יכול להוביל לכך שהזמן יגמר לפני שנצליח לבדוק את כל האפשרויות, ולהגיע לפתרונות יותר אופטימליים.
בנוסף, יוריסטיקה קלה עשויה להוביל לפתרון לא אופטימלי כי היא אינה הכי מיודעת בניגוד לקשה.

---

2. דנה אינה צודקת, נדגים את זה ע"י תיאור דוגמה נגדית:



אלגוריתם $MiniMax$ יתחיל בריצה שלו בצד שמאל (כי הוא יותר קל), ואז ימשיך בריצה ויחזיר את המסלול המסומן בצהוב בציור, אבל מה שהוא היה צריך להחזיר זה את המסלול הקל יותר ממנו שהוא האדום, בדוגמא הזו ראינו מסלול יותר קל יותר קצר שהאלגוריתם אינו בוחר בו, ולכן היא לא צודקת.

---

3. נתמודד עם זאת ע"י הרצת אלגוריתם מינימקס כל פעם עד דרגה מסויימת (עומק מסוים), וכל עוד הזמן לא נגמר נמשיך עם האלגוריתם לעומק יותר גבוה מקודם, כאשר נגמר נזמן נחזיר את התשובה שקיבלנו באיטרציה האחרונה שנסרקה כולה.
השם הכללי לאלגוריתמים אשר יכולים לשפר את ביצועיהם ככל שיש להם יותר זמן הוא $anytime$, ודוגמה לאחד שלמדנו בקורס הוא $anytime - \alpha\beta$.

.5

a. במצב זה, כל שחקן בעץ המשחק יחושב כשחקן - $maximum$ והוא רוצה למקסם את עצמו ללא תלות לשחקנים האחרים.

```
function MiniMax(State, Agent):
    if G(State) then return U(State, Agent)
    Turn <- Turn(State)
    Children <- Succ(State)
    CurrentMax <- -∞
    Loop for child in Children
        v <- MiniMax(child, Agent)
        CurrentMax <- Max(v, CurrentMax)
    Return (CurrentMax)
```

---

b. מכיוון שכל שאר השחקנים רוצים שאני אפסיד, אז הם תמיד יבחרו את מה שיגרום למצב הכי גרוע אצלי, ולכן כל שאר השחקנים הם שחקני מינימום, ובן זמן מכיוון שהשחקן שלי רוצה לנצח אז הוא יהיה שחקן מקסימום.

```
function MiniMax(State, Agent):
    if G(State) then return U(State, Agent)
    Turn <- Turn(State)
    Children <- Succ(State)

    ////    My player's turn, who playes as a max
    if Turn = Agent then:
        CurrentMax <- -∞
        for child i n Children:
            v <- MiniMax(child, Agent)
            CurrentMax <- Max(v, CurrentMax)
        return CurrentMax
    ////    Not my player's turn, so another player who playes as a min
    else:
        CurrentMin <- ∞
        Loop for child in Children
            v <- MiniMax(child, Agent)
            CurrentMin <- Min(v, CurrentMin)
        Return (CurrentMin)
```

---

c. כל שחקן מנסה למקסם את התועלת (הרווח) של השחקן הבא אחריו בתור.

```
////    We could add the K (number of players as an argument)
function MiniMax(State, Agent):
    if G(State) then return U(State, Agent)
    Turn <- Turn(State)
    Children <- Succ(State)
    CurrentMax <- -∞
    Loop for child in Children
        ////    The change from the first part, run the Algorithm for the next Agent
        ////    So that we calculate the max profit for him using MiniMax
        ////    If we add K as an argument, we should add %K, so that we can make it cycle
        v <- MiniMax(child, Agent + 1)
        CurrentMax <- Max(v, CurrentMax)
    Return (CurrentMax)
```

<u>חלק ג - $Alpha - Beta$:</u>

2. כן, זה שונה כי $Alpha - Beta$ גוזם ענפים, ולכן הוא מעבד את הבנים של הצומת בזמן קצר יותר מ-$RB - Minimax$. כך הוא יכול לבדוק יותר אפשרויות באותו פרק זמן, מה שהופך אותו ליעיל ומודע יותר. לכן, $Alpha - Beta$ נחשב לביצועי יותר מ-$RB - Minimax$.

חלק ד - $Expectimax$:

1. הסוכן בוחר באופן רנדומלי מבין 7 אופרטורים לכן ההסתברות היא $\frac{1}{7}$.

2. כמו אלגוריתם $Alpha - Beta$, גוזמים את הצמתים שבהם מתקיים:
$v > currMin$ או $v < currMin$ לכן נעשה זאת רק כבור $\alpha = 1$ קבוע. ברגע שמגיעים לצומת בן עבורה $h(s) = 1$ ואנחנו במשחק $maximum$ אין צורך להמשיך לשאר הבנים כי זה הערך הכי גדול שאפשר, לכן גוזמים את כל שאר הבנים.

חלק ה - משחק עם פקטור סיעוף גדול:

.1

a. האופרטורים לא משתנים מכיוון שאין אפשרות להחזיק שתי חבילות יחד. לכן, או שעושים "$drop\ off$" (אם מחזיקים חבילה) או "$pick\ up$" (אם לא מחזיקים חבילה), במקרה הגרוע מספר הפעולות יהיה שווה למספר הפעולות המקורי, ולמרות שייתכן שיש צמתים שבהם מספר הפעולות יקטן כאשר מוסיפים מחסומים בסביבה, אלא זה לא משפיע על מקדם הסיעוף כי מספר האופרטורים אינו משתנה, ולכן הוא נשאר 7.

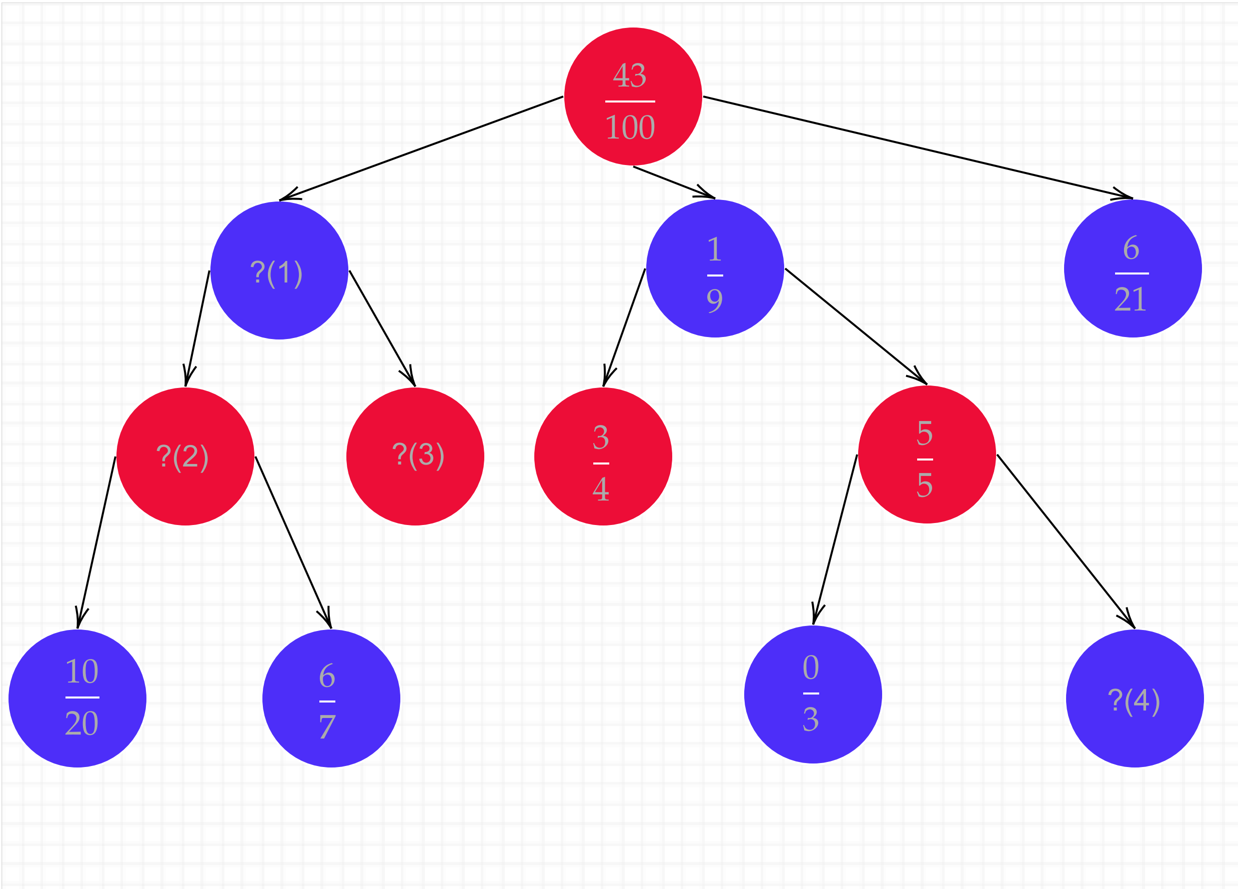
b. נוספה פעולה חדשה חוקית על 23 משבצות (כל המשפצות חוץ משתי המשפצות שנמצאו בהם הרובוטים) לכן מקדם הסיעוף יגדל ב- 23 ויהיה 30.

.2

a. כמו שאמרנו מקדם הסיעוף ב- $1.b$ הוא 28 לכן סיבוכיות הזמן של האלגוריתמים היא $O\left(28^d\right) = O\left(b^d\right)$ לכן האלגוריתמים אינן סבירים כי מקדם הסיעוף גדל הרבה. אך אפשר להשתמש באלגוריתם $Greedy$ שבוחר את הצעד הבא תוך זמן קצר, כי מספר הבנים בכל שכבה חסום ולכן האלגוריתם יהיה תלוי בעומק העץ.

b. נשתמש באלגוריתם $monte-carlo$ ו- $Minimax$, כך כאשר נגיע למצב בעץ שיש בו הרבה בנים נשתמש בגרסת $monte-carlo$ לפי התרגול, כך שניקח K בנים ונחשב בתוחלת מי ה- Max ומי ה- Min ונבחר את הערך לפי $Minimax$.

חלק ו - יבש - שאלה פתוחה - $MCTS$:



.1

חישובי עזר:

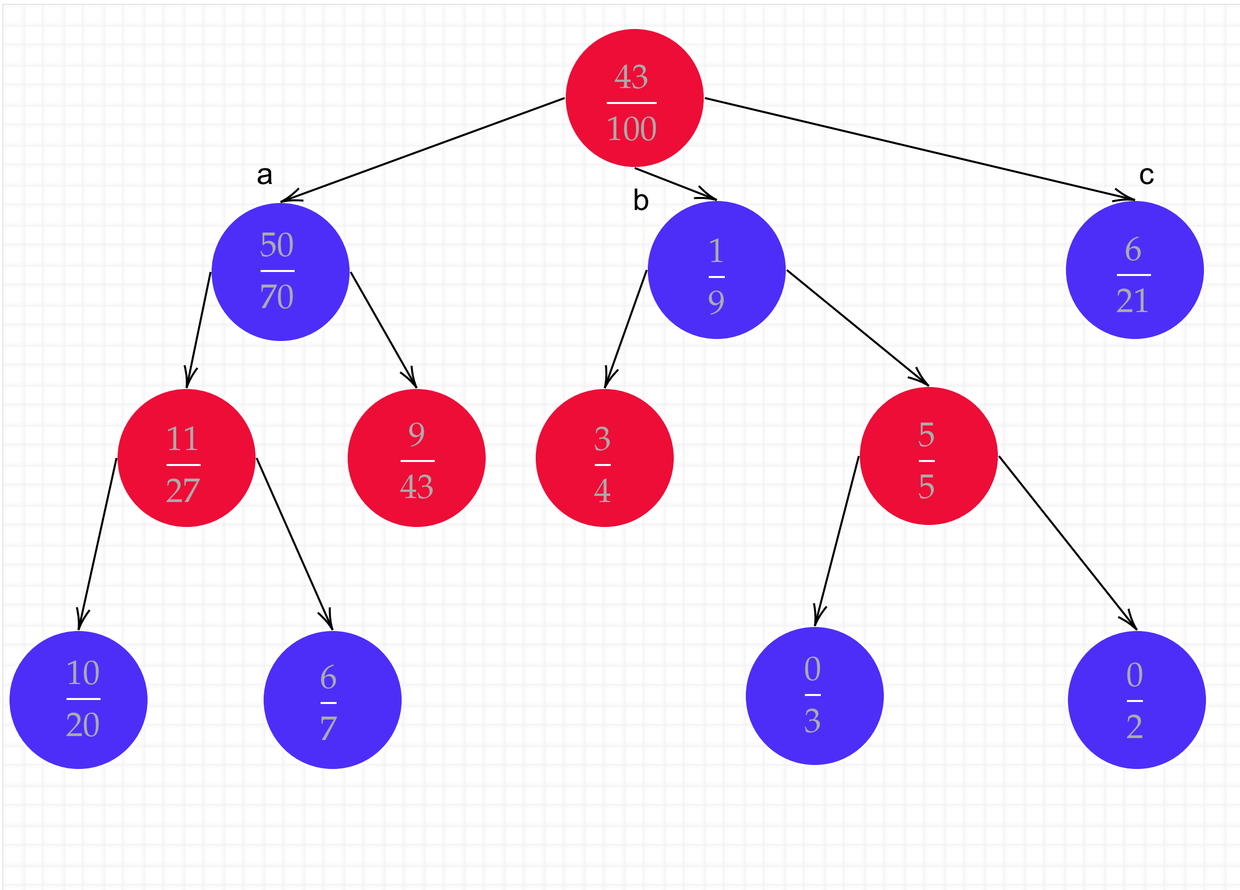
$$\frac{100-43-1-6}{100-9-21} = \frac{50}{70} \quad (1)$$

$$\frac{(20+7)\ -\ (10+6)}{20+7} = \frac{11}{27} \quad (2)$$

$$\frac{70-50-11}{70-27} = \frac{9}{43} \quad (3)$$

$$\frac{5-5-0}{5-3} = \frac{0}{2} \quad (4)$$

נקבל:



---

2. נחשב תחילה את ה- $UBC1$:

צאצא של a:

$$\frac{50}{70} + \sqrt{2} \cdot \sqrt{\frac{ln(100)}{70}} = 1.077$$

צאצא של b:

$$\frac{1}{9} + \sqrt{2} \cdot \sqrt{\frac{ln(100)}{9}} = 1.12$$

צאצא של c:

$$\frac{6}{21} + \sqrt{2} \cdot \sqrt{\frac{ln(100)}{21}} = 0.95$$

לכן נבחר את צומת b בעלת ערך $UBC1$ הכי גדול.

---

3. נרצה שצומת a או c תיבחר לכן נדרוש כי:

$UBC1(c) > UBC1(b)$ וגם $UBC1(a) > UBC1(b)$

לכן נרצה ש:

$$UBC1(b) = \frac{1}{9 + x_1} + \sqrt{2} \cdot \sqrt{\frac{ln(100 + x_1)}{9 + x_1}} < \frac{6}{21} + \sqrt{2} \cdot \sqrt{\frac{ln(100 + x_1)}{21}} = UBC1(c)$$

לכן נקבל - $x_1 \geqslant 4$ במצב זה.

$$UBC1(b) = \frac{1}{9 + x_2} + \sqrt{2} \cdot \sqrt{\frac{ln(100 + x_2)}{9 + x_2}} < \frac{50}{70} + \sqrt{2} \cdot \sqrt{\frac{ln(100 + x_2)}{70}} = UBC1(a)$$

ונקבל במצב זה- $x_2 \geqslant 1$.

לכן מספר הנימחונות המינימלי x יהיה: $x = min\{x_1, x_2\} = 1$.

שאלה פתוחה – אלגוריתם מונטה קרלו למשחקי אינפורמציה חלקית:

1. הבעיה שאלגוריתם מונטה קרלו מנסה לפתור היא כאשר המשחק ידוע רק חלקית, ויש הרבה אפשרויות וקשה לשחקן לדעת מה להחליט ואיך להתנהג, אז משלימים את המצב החלקי למצב מלא בכל האפשרויות החוקיות, והבעיה היא שיש הרבה אפשרויות להשלים שזה לוקח זמן רב, האלגוריתם מתמודד עם זאת בכך שהוא דוגם רק חלק k מכל האפשרויות.

---

2. נסביר את שני המקרים:
   a. עבור k קטן מדי: נקבל ריצה יותר מהירה, כי זמן החישוב יותר קטן עבור כל צעד חישוב, אבל הוא יהיה גם פחות מדויק, כי הרבה מידע הולך לאיבוד, כי מגבילים את זמן החישוב.
   b. עבור k גדול מדי: נקבל ריצה יותר איטית, כי נותנים הרבה זמן חישוב לכל צעד חישוב, אבל הוא יהיה יותר מדויק, כי הוא מקבל מספיק זמן בשביל לחשב כל מצב ולכן מביא יותר מידע ואז ההחלטות טובות יותר.

---

3. עבור ערך k גדול המקיים $k \geqslant |S\_complete|$, האלגוריתם דוגם k אופציות ומחשב את הממוצע של כל צעד אפשרי, ועבור ה- k שבחרנו האלגוריתם בודק את כל האפשרויות בצדורה אחידה, ולכן נוכל למדל ע"י שימוש בצמתים הסתברותיים ואלגוריתם $Expectimax$, כך שהצמתים ההסתברותיים יכילו את כל האפשרויות $S\_complete$ ולכל אפשרות יש לה הסתברות $\frac{1}{k}$ לקרות, ואז ריצת אלגוריתם $Expectimax$ עם המצתים ההסתברותיים שהגדרנו תהיה דומה לריצת אלגוריתם מונטה קרלו עם כל האפשרויות שזה $k = |S\_complete|$.

---

4. נגדיר שכל עד הזמן לא נגמר, אז ניתן לאלגוריתם לרוץ עם יותר $samples$, כאשר מתחילים מ- 1 ואחרי כל צעד בודקים אם הזמן לא נגמר נגדיל כל פעם ב- 1.

```
function RB_MonteCarlo(PartialState, Agent, D, limit):
    ////    collect all of the legal actions from the current PartialState
    Actions <- All legal actions in PartialState
    ////    collect all of S_complete so we can select random K from it
    S_complete <- All states consistent with PartialState
    ////    The initial value of K to start the anytime Algorithm with
    K <- 1
    Samples <- random any K samples from S_complete
    while limit > 0:
        ////    sample random K to run with
        Samples <- random any K samples from S_complete

        Loop for a in actions:
            Loop for s in Samples:
                v(a) <- (v(a) + RB_AlphaBeta(a(s), Agent, D, Alpha = -∞, Beta = ∞)) / K
            value = select a with max(v(a))
        K <- K + 1
    return value
```