מבוא לבינה מלאכותית – תרגיל בית 2

מגיש1: רוני רויטבורד
ת.ז.1: 313575599
מגיש2: יונתן עזיזי
ת.ז.2: 206588766
מגיש3: מרינה ינובסקי
ת.ז.3: 324515659

חלק א'

שאלה 1

נשתמש בשקופית הבאה מההרצאה על מנת לפרמל את המשחק הנתון:

State Space Search with Multiple Agents

- Formal definition is a tuple $< S, A, f, c, s_0, R >$ with
  - a discrete **state space** S
  - a set of n agents $\{1,2, \dots, n\}$
  - a set $A = \{A^i\}_{i=1}^{n}$ **of actions for each agent**: a collection of action sets A^i, one for each agent in the environment.
  - a **successor** function – if it is **deterministic** it represents the state transition function $f(a, s)$ such that $s' = f(a, s)$ stands for the state s' resulting from applying an applicable action a at state s (this definition is only applicable if agents act in turns such that only one action can be applied at a given execution step!)
  - a **cost function** $c(a, s)$ associating a **positive cost** to applying action a in s
    Type equation here.
  - a known **initial state** $s_0 \in S$
  - a function $R = \{R^i\}_{i=1}^{n}$ specifying for each agent i, a **reward function** $R_i(a, s)$ which associates a **reward** for applying action a in s
- Objective
  - Find a policy for agent i a mapping $\pi_i: S \mapsto A_i$ from states to actions A_i
  - **Optimality criteria depends on the setting and on the relationship between agents**

תחילה נגדיר את הפרמטרים הבאים:

$o \in on\_board$ כאשר: $o \in \{0,1\}$ ומתאר האם האובייקט במפה.

$d \in destination$ כאשר: $d = (x, y), 0 \le x, y \le 4$ ומתאר את מיקום יעד האובייקט במפה (במקרה שלנו, היעד של החבילה)

$b \in battery$ כאשר: $b \in N^+ \cup \{0\}$ ומתאר את מצב הסוללה של הרובוט.

$c \in credit$ כאשר $c \in N^+ \cup \{0\}$ ומתאר את הקרדיט (הניקוד) הנתון לרובוט.

$p \in position$ כאשר: $p = (x, y), 0 \le x, y \le 4$ ומתאר מיקום במפה.

כעת נגדיר את בעיית החיפוש שלנו:

- מרחב המצבים S מתואר ע"י האובייקט WarehouseEnv:

$$S = \{(C_0, C_1, P_0, P_1, R_0, R_1, c)\}$$

כאשר מתקיים:

$$C_0, C_1 \in ChargeStation[position]$$
$$P_0, P_1 \in Package[position, destination, on\_board]$$
$$R_0, R_1 \in Robot[position, battery, credit, Package]$$
$$c \in N^+ \cup \{0\}$$

המשתנה האחרון c מתאר את מספר התורות שיש לסוכן עד להגדרת סוף המשחק.

- $A = \{A^i | A^i \in \{ north, south, east, west, pickup, charge, dropoff\}^{\{1,2\}}\}$ מתאר את הפעולות האפשריות לכל סוכן.
- $f(a, s) = s'$ כאשר $a \in A \ \& \ s, s' \in S$ וגם המצב s' נובע מהפעלת הפעולה a על רובוט מסוים במצב s.

- $c(a,s) \in N^+ \cup \{0\}$ פעולת המחיר לרובוט, במקרה שלנו מדובר בסוללת הרובוט (לפי הפונקציה get_local_operators) ובמקרה שהיא ריקה (ערכה 0) המחיר יהיה 0 אך מרחב הפעולות האפשריות יהיה מוגבל.
- $s_0 \in S$ מצב התחלתי, מתואר ע"י הפונקציה generate ומאתחל את רכיבי מרחב המצבים באופן אקראי (לפי ה-seed הנבחר)
- $R = \left\{r^{\{1,2\}} | r \in N^+ \cup \{0\}\right\}$ כאשר R זאת פונקציית התגמול שלנו והיא מתארת את התגמולים האפשריים עבור אחד משני הסוכנים (רובוטים) שלנו.

שאלה 2

H(s) = 100 * robot.credit+ (reward / (pathCost + 1)) + robot.battery
Where:
The components reward and pathCost are defined as follows:

1. If the robot is carrying a package: reward = 2 * MD(Ppos, Pdest)
   pathCost = MD(Pdest, Rpos)
2. If the robot is not carrying a package: reward = 2 * MD(NPpos, NPdest)
   pathCost = MD(Rpos, NPpos) + MD(NPpos, NPdest)

Where:

- MD(a, b) is the Manhattan distance between points a and b
- Ppos is the package's position
- Pdest is the package's destination
- Rpos is the robot's position
- NPpos is the nearest package's position
- NPdest is the nearest package's destination

שאלה 3

מומש.

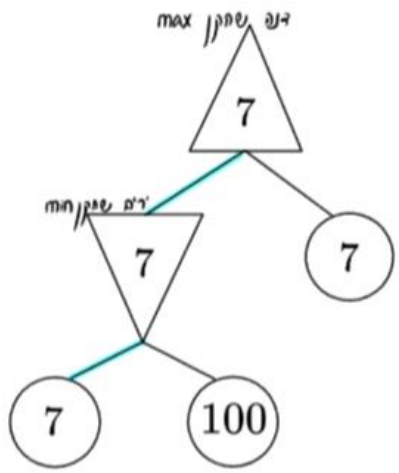
חלק ב'

שאלה 1

מכיוון שהאלגוריתם מוגבל משאבים, הוא ירוץ עד שימצה את כל המשאבים שברשותו. לכן היתרון של שימוש בהיוריסטיקה קלה לחישוב היא שינתנת לחישוב מהיר יותר ולכן נוכל לרדת עמוק יותר בעץ האסטרטגיה, ולקבל החלטה מודעת יותר, שמטיבה עימנו יותר (מבחינת האלגוריתם). לעומתה היוריסטיקה הקשה לחישוב תנצל יותר משאבים כי רצה יותר זמן, ולכן נחקור פחות רמות בעץ האסטרטגיה. עם זאת, רמת הדיוק של החישוב נמוך עבור היוריסטיקה הקלה, כי ייתכן וקיים מצב שיוביל אותנו לערך סופי טוב יותר אך מצב אחר קיבל ערך טוב ממנו עקב חוסר הדיוק של היוריסטיקה. לעומתה, היוריסטיקה הקשה לחישוב תחזיר תוצאה מדויקת יותר, כלומר שאם נלך בכיוונו הוא יטיב עימנו יותר.
כלומר, עבור היוריסטיקה קלה - יתרון הינו שאנחנו נסרוק יותר רמות בעץ ההחלטה ונבחר פיתרון מבין מרחב רחב יותר של אפשריות, אך החיסרון הוא פגיעה בדיוק ובטיב הפתרון.

שאלה 2

דנה טועה:



מאופן פעולת האלגוריתם, הצד השמאלי יפותח קודם ומכיוון שהערך הטוב ביותר שניתן יהיה לקבל מצד זה הוא 7, והצד הימני לא משפר אותו, האלגוריתם יבחר להמשיך לצד השמאלי.
בצעד הבא (של הריב), הוא יבחר כמובן ב-7.
במשחק זה - הערך האופטימלי שיכול להתקבל הינו 7, בשקלול המהלכים. ובדוגמה הנ"ל - ראינו כיצד הגענו לערך זה בשני צעדים בעוד שקיים מסלול באורך צעד אחד.

שאלה 3

שם כללי לאלגוריתם שמשפר את ביצועיו ככל שיש לו יותר זמן הוא anytime search שעובד בצורה העמוקה הדרגתית כל עוד נותר לו זמן ריצה.
אלגוריתם כזה שלמדנו בעבר ID-DFS-
נתמודד עם הגבלת הזמן באופן הבא- נריץ RB-Minmax באיטרציות גדלות על העומק, כלומר תחילה נריץ אותו על עומק 1, לאחר מכן על עומק 2 וכן הלאה, באופן הדרגתי. כאשר הזמן שברשותנו "יגמר", תוך כדי הרצת RB-Minmax על עומק n נחזיר את הפיתרון שקיבלנו מהריצה על עומק n-1 שכן הרצה זו הסתיימה במלואה באיטרציה הקודמת על העומק, ופיתרון זה הוא המדויק ביותר מבין הפתרונות שקיבלנו באיטרציות הקודמות.

שאלה 4

מומש

שאלה 5

כל סוכן רוצה לנצח ולא אכפת לו מהפעולת של סוכנים אחרים: לכל סוכן אין צורך להתחשב בפעולות של הסוכנים האחרים, ולכן כל סוכן פשוט יבחר את המהלך שממנו הוא מקבל את ה-utility המקסימלי. כלומר, בפועל כל סוכן יבחר בצעד הבא בצורה חמדנית מבין הצעדים הבאים האפשריים. עם זאת, הסוכן שלנו ימשיך לשחק כסוכן מקסימום.

```
function MinimaxA (State, Agent)
    Children = Succ(State)
    Turn = Turn(State)
    curMax = -inf
    if Turn = Agent then:
        if G(State) then return U(State, Agent)
        loop for c in children:
            v= MinimaxA(c, Agent)
            curMax = Max( c, curMax)
    else:
        Loop for c in Children:
            curMax = Max(c, curMax)
    return curMax
```

כל סוכן רוצה לפגוע בשני :הסוכן שלנו מתנהג כמו סוכן מקסימום בעוד ששאר הסוכנים מתנהגים כמו סוכני מינימום .לכן, זהו שינוי מהפסאודו-קוד שהוצג בתרגול, כי כל עוד התור לא שלנו אלא של אחד משחקני המינימום, תתבצע פעולת המינימום כפי שנדרש.

כל סוכן רוצה שהסכום שהוא יקבל יהיה מקסימלי :זאת אומרת שכל הסוכנים מתנהגים כמו סוכני מקסימום. לכן, במקום להתחשב בערכי המינימום, השחקן שלנו יתחשב בערך המקסימום .כל סוכן בתורו מבצע בחירת מקסימום על המצבים העוקבים, מה שמוביל לכך שכל צעד ייטיב עם השחקן הבא בתור.

```
function MinimaxC (State, Agent)
    if G(State) then return U(State, Agent)
    Children = Succ(State)
    curMax = -inf
    Loop for c in Children
        v = MinimaxC(c, Agent)
        curMax = Max(curMax, v)
    return curMax
```

חלק ג'

שאלה 1

מומש

שאלה 2

יתכן שהסוכן יתנהג באופן שונה מהסוכן בחלק ב:

מבחינת בחירת המהלכים - עבור מגבלת זמן זהה על שני הסוכנים, מכיוון שסוכן אלפא-בטא מבצע גיזום ענפים, הוא ינצל את הזמן העומד לרשותו בצורה טובה יותר. הוא יצליח לחשב מהלכים עד לעומק עמוק יותר, ולכן הוא יכול להחליט מהלך שונה מהמהלך שהחזיר הסוכן בחלק הקודם.

מבחינת זמן ריצה - הסוכן בחלק זה מהיר יותר מכיוון שמבצע גיזום, אופן קבלת ההחלטות לא משתנה אך הוא ישתמש בערכי אלפא ובטא כדי לא לפתוח צמתים שידוע שלא יבחרו בהם.

חלק ד'

שאלה 1

נשתמש בהסתברות שמתפלגת יוניפורמית (לכל מצב סיכוי שווה להיבחר), מכיוון שבחירה רנדומלית לחלוטין משמעותה סיכוי שווה לכל מצב עוקב להיבחר, כלומר נדמה את אופן המשחק האמיתי בצורה הטובה ביותר.

שאלה 2

נגדיר: $\alpha = -1, \beta = 1$, הערכים שעל פיהם נבצע גזימה.
נסמן- v_i: $(prob.i) * (nextState\_i\ value)$ עבור כל מצב עוקב וההסתברות המתאימה לו. חישוב v יתבצע בלולאה, כאשר האיטרציות הן על המצבים העוקבים ובכל איטרציה מתווסף ל-v הערך של החישוב הנ"ל (כל איטרציה בקוד- מחושב v_i מתאים ומתווסף לערך של הv הכולל):

$$v = \sum_{i=1}^{all\_succ} v\_i$$

גזימה בצומת ההסתברותי כאשר האב הינו צומת מקסימום:

תנאי הגזימה: לא נמשיך לחשב את הערכים של המצבים העוקבים (לא נמשיך באיטרציות) אם הערך של אלפא גדול/שווה מסכום v וההסתברות שנותרה:

$$\alpha \geq v + remaining\_prob$$

הסבר: ערך אלפא התקבל מהקריאה הרקורסיבית. אלפא מייצג את הערך הגדול ביותר עד כה שהאב מצא בחישובו. מכיוון שערך היוריסטיקה לכל היותר 1, בהינתן v שהתקבל לאחר מספר כלשהו של איטרציות- הערך המקסימלי שיתווסף אליו הינו הכפלת ההסתברות שנותרה בערך הבן המקסימלי האפשרי (כאמור 1), ולכן אם האלפא גדול מהסכום הזה- הרי שכבר לאב יש מועמד טוב יותר שייבחר ואין סיבה להמשיך בחישוב עבור המצב הנוכחי.

גזימה בצומת הסתברותי כאשר האב הוא צומת מינימום:

תנאי הגזימה: באופן שקול, נבצע גזימה כאשר

$$\beta \leq v - remaining\_prob$$

הסבר: ערך בטא התקבל מהקריאה הרקורסיבית. בטא מייצג את הערך הקטן ביותר עד כה שהאב מצא בחישובו. מכיוון שערך היוריסטיקה לכל הפחות מינוס 1, בהינתן v שהתקבל לאחר מספר כלשהו של איטרציות- הערך שיקטין אותו הכי הרבה הוא אם נוסיף אליו את הכפלת ההסתברות שנותרה בערך הבן המינימלי האפשרי (כאמור 1-), ולכן אם הבטא קטן מהסכום הזה- הרי שכבר לאב יש מועמד טוב יותר שייבחר (האב בוחר מינימום) ואין סיבה להמשיך בחישוב עבור המצב הנוכחי.

שאלה 3

מומש

חלק ה'

שאלה 1

a. תחילה נשים לב כי השינוי המדובר הוא בשטח הפעולה (האיברים המרכיבים את S) ולא במרחב הפעולות שרובוט יכול לבצע. מכיוון שמרחב הפעולות שרובוט יכול לבצע לא משתנות, נסיק מכך שמקדם הסיעוף (המתאר את הפעולות שסוכן יכול לבצע בכל תור) לא ישתנה. בהגדלת לוח המשחק, אנו לא משפיעים על פעולות שרובוט יכול לבצע (כפי שהוגדר לפני כן) ולכן סוכן יוכל לבצע בדיוק את אותם פעולות כמו לפני כן, כלומר מקדם הסיעוף יישאר זהה. בהוספת מחסומים, ברגע שסוכן מנסה להתקדם לעבר משבצת חסומה הוא יישאר במקום ולא יזוז, מה שמדמה פעולת Park ולכן גם במקרה זה מקדם הסיעוף לא ישתנה. בשני המקרים קיבלנו כי מקדם הסיעוף החדש שקול למקדם הסיעוף ללא השינויים (וערכו 7 כפי שהגדרנו את A^i בחלק א בשאלה 1.1).

b. על מנת לפתור סעיף זה, נבין תחילה מהו מצב המשחק בו יש לנו את המספר המקסימלי של פעולות, כלומר מספר מקסימלי של משבצות ריקות. במצב זה, הרובוטים יחזיקו את החבילות שלהם בעמדת טעינה שהיא גם יעד לחבילה ולכן, אם במפה הרגילה יש 25 משבצות שמתוכן 2 תפוסות נסיק שיש לכל היותר 23 משבצות לבחור מביניהן לשים בהן בלוק. במקרה מקסימלי זה לא ניתן לבצע פעולת איסוף חבילה מכיוון שהרובוט כבר מחזיק חבילה ולכן נקבל שמקדם הסיעוף הוא: $23 + (7 - 1) = 29$.

שאלה 2

a. נבחן את סיבוכיות זמן הריצה כתלות במקדם הסיעוף של האלגוריתמים מסעיפים קודמים. לפי ההרצאות ידוע כי מתקיים:

$$ImprovedGreedy: O(b)$$
$$RB - Minimax: O(b^d)$$
$$Alpha - Beta: \left[O\left(b^{\frac{d}{2}}\right), O(b^d)\right]$$
$$Expectimax: O(b^d)$$

בהינתן כי הושמו השינויים מהסעיף b בשאלה הקודמת, מקדם הסיעוף עלה משמעותית ביחס למקדם הסיעוף המקורי ולכן על מנת לבחור באלגוריתם שישמר זמן ריצה סביר נבחר באלגוריתם שחמדן ImprovedGreedy שסיבוכיות זמן הריצה שלו תלויה באופן ליניארי במקדם הסיעוף, בשונה מהאלגוריתמים האחרים.

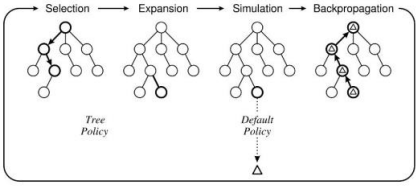
b. נציע אלגוריתם MCTS: אלגוריתם זה מציע את הצעד הבא לביצוע בהינתן הידע שהוא צבר על עץ המשחק, על ידי ביצוע סימולציות על עץ זה (חיפוש לעומק מספר מעמים שאינו תלוי בסיעוף). כלומר- גם כאן, התלות הינה לינארית בעומק. האלגוריתם בונה עץ סטטיסטיקות שממפה את עץ המשחק באופן חלקי, והעץ הזה מנחה את החיפוש של האלגוריתם כך שיחקור את האיזורים "הכי מעניינים בו" כפי שראינו בהרצאה. הערך

שמקבל מצב- הינו ממוצע הערכים שקיבל במספר סימולציות (הרצות משחק אפשריות).
המצב הבא לביצוע- המצב שממנו קיבלנו מספר נצחונות הכי גבוה במשחקים שהרצנו.
בכל אופן, כפי שראינו בהרצאה, בהינתן מקדם סיעוף גבוה, האלגוריתם יכול להריץ כ10^7 סימולציות, בניגוד מינימקס שיגיע לעומק 6, ולכן בפועל נקבל החלטות טובות יותר בהינתן אותם המשאבים תחת התנאים המוגדרים בשאלה.

Monte Carlo Tree Search (MCTS) - efficiency

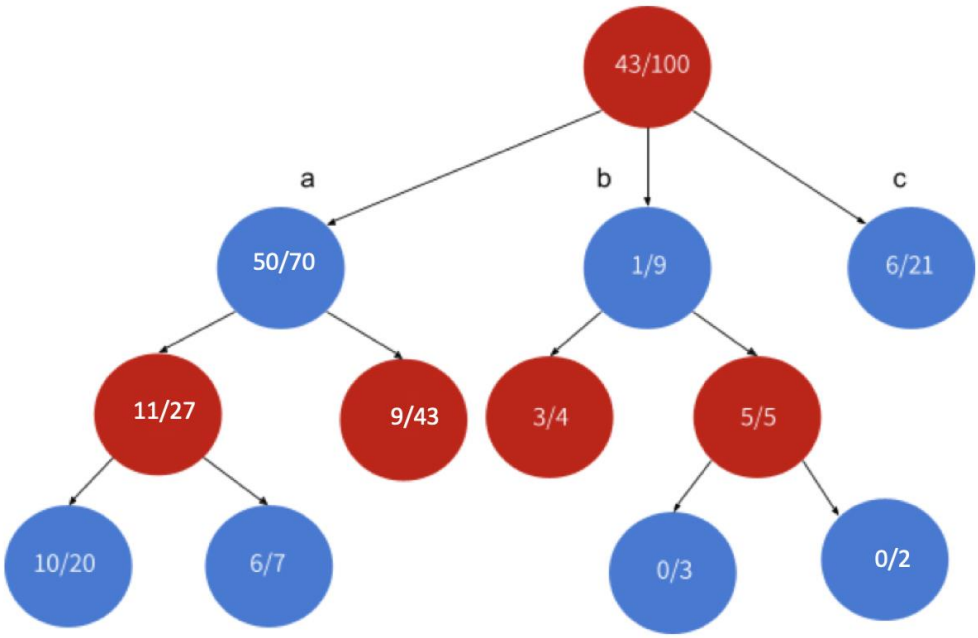
- The time to perform a rollout is linear in the depth of the tree
- This gives plenty of time to consider multiple rollouts

- For example, if
  - Branching factor b=32
  - Average game length (tree depth) is d=100
  - We can compute 10^9 moves
- `Minimax` can search 6 ply deep
- `AlphaBeta` can search up to 12 ply deep
- `MCTS` can do 10^7 rollouts



חלק ו'

1. התוצאה שיצאה היא:



2. נריץ את UCB1 על הילדים של השורש ונדע מהי הצומת הבאה שבה נבחר:

$$UCB1(a) = \frac{50}{70} + \sqrt{2} \cdot \sqrt{\frac{\ln(100)}{70}} = 1.077$$

$$UCB1(b) = \frac{1}{9} + \sqrt{2} \cdot \sqrt{\frac{\ln(100)}{9}} = 1.122$$

$$UCB1(c) = \frac{6}{21} + \sqrt{2} \cdot \sqrt{\frac{\ln(100)}{21}} = 0.947$$

נשים לב כי התוצאה המקסימלית המתקבלת היא עבור UCB1(b) ולכן הצומת הבא שיבחר בשלב ה-selection יהיה צאצא של b.

3. נבדוק מה המספר המינימלי של ניצחונות שצריך להוסיף לכל בן אחר לשורש ונבחר את המספר המינימלי מבין האפשרויות כלומר, נפתור את המשוואה הבאה:

$$\underset{x}{argmin}\big(UCB1(a_x) > UCB1(b_x), UCB1(c_x) > UCB1(b_x)\big)$$

נציין כי בעת בניית המשוואות $UCB1(a_x), UCB1(c_x)$ נוסיף נצחונות למצבים a,c בשונה מהבנייה של $UCB1(b_x)$ שבו אנו לא נוסיף נצחונות אלא נשאר עם בדיוק ניצחון יחיד. נבדוק את המקרה הראשון:

$$UCB1(a_x) > UCB1(b_x)$$

$$\frac{50+x}{70+x} + \sqrt{2}\sqrt{\frac{\ln(100+x)}{70+x}} > \frac{1}{9+x} + \sqrt{2}\sqrt{\frac{\ln(100+x)}{9+x}}$$

$$\frac{50+x}{70+x} - \frac{1}{9+x} > \sqrt{2}\left(\sqrt{\frac{\ln(100+x)}{9+x}} - \sqrt{\frac{\ln(100+x)}{70+x}}\right)$$

$$\frac{(50+x)(9+x)-(70+x)}{(70+x)(9+x)} > \sqrt{2}\left(\frac{\ln(100+x)\cdot\frac{61}{(x+9)(x+70)}}{\sqrt{\frac{\ln(100+x)}{9+x}} + \sqrt{\frac{\ln(100+x)}{70+x}}}\right)$$

$$\frac{x^2+59x+380}{(70+x)(9+x)} > \left(\frac{61\sqrt{2}\ln(100+x)}{(x+9)(x+70)\left(\sqrt{\frac{\ln(100+x)}{9+x}} + \sqrt{\frac{\ln(100+x)}{70+x}}\right)}\right)$$

$$\ldots \Rightarrow x = 1$$

נקבל כי ה-x המינימלי המקיים בחירה של צאצא מצומת אחרת מ-b הוא 1 ניצחון יחיד נוסף.
כלומר, מצאנו שעבור x=1 המשוואה מתקיימת ומכיוון שמתקיים $x \in \mathbb{N} \cup \{0\}$ והתנאי לא מתקיים עבור x=0 (ללא משחקים נוספים נקבל את אותם תוצאות של סעיף קודם) נקבל כי ה-x המינימלי שמצאנו הוא 1 ולא קיים מספר יותר קטן המקיים את התנאי ולכן נבדוק

$$UCB1(c_{x=1}) \overset{?}{\lesseqgtr} UCB1(a_{x=1})$$

$$UCB1(c_{x=1}) = \frac{7}{22} + \sqrt{2}\sqrt{\frac{\ln(101)}{22}} = 0.96591$$

$$UCB1(a_{x=1}) = \frac{51}{71} + \sqrt{2}\sqrt{\frac{\ln(101)}{71}} = 1.07887$$

מצאנו כי $UCB1(c_{x=1}) < UCB1(a_{x=1})$ ולכן אם נוסיף ניצחון ל-a נקבל שבאלגוריתם UCB1 נבחר צאצא של a.

חלק ז'

1. הבעיה במשחקי אינפורמציה חלקית היא שהסוכן יודע רק על חלק ממשתני הסביבה והוא צריך להשלים את המידע לבד. בחלק מן המשחקים, השלמת המידע יכולה להיות קבוצת קומבינציות עצומה במיוחד, כך שאלגוריתם שירצה לחשב את כל המסלולים האפשריים יצטרך לחשב מספר גדול מאוד של מסלולים ולהאריך במיוחד את סיבוכיות הזמן והמקום. אלגוריתם מונטה-קרלו למשחקי אינפורמציה חלקית פותר בעיה זו ע"י דגימה של k אפשרויות להשלמת המידע החסר בשלב מסוים במקום לרוץ על כולם, ועושה אגרגציה (ממוצע) בין האפשרויות שקיבל. בדרך זו, אנו חוסמים את סיבוכיות הזמן והמקום ע"י K כפול הסיבוכיות המקורית לפעולה ומתגברים על הבעיה שיש יותר מדי דרכים להשלים מידע.

2. אם נבחר k <u>גדול</u> מדי, כנראה שנצליח לזהות בקירוב טוב יותר את מרחב ההתפלגות של השלב הבא ובכך גם את הצעד העדיף ביותר לסוכן בשלב הנוכחי אך בזבזנו המון משאבים בשביל להגיע לתוצאה הנכונה וככל שנגדיל את k יותר כך נוותר על המטרה לשמה השתמשנו באלגוריתם זה מלכתחילה ונדגום מספר גדול מאוד של אפשרויות ונעמיס על הסיבוכיות.
אם נבחר k <u>קטן</u> מדי, אנו עלולים לפספס את המצב העדיף (סטטיסטית) לסוכן בשלב זה, ולמרות שאנו עושים ממוצע בין כל הדגימות אנו עדיין עלולים להתעלם ממצב עדיף בהרבה שלא היינו מתעלמים ממנו אם היינו מבצעים מספיק דגימות. לדוגמה, במשחקי קלפים האינפורמציה הלא ידועה היא לרוב הקלפים אצל היריב. קיימים מצבים שהקלפים של היריב יותר טובים משל הסוכן שלנו ומצבים שההפך הוא הנכון, אם נגדיר k=2 אנו עלולים לדגום בצורה לא מספקת את מרחב הפעולות של היריב ובכך להימנע מלבחור במצב הנכון עקב הממוצע שעשינו, ובכך נקבל נתונים סטטיסטיים שונים מהנתונים האמיתיים.

3. נתבונן באלג' מונטה קרלו:

```
function MonteCarloPartialInformation(PartialState, Agent, D, K):
    Actions ← All Legal actions in PartialState }  /*all possible actions to choose from*/
    S_Complete ← All states consistent with PartialState
    Sample ← Sample K random states from S_Complete  /*only K possible states are unifor
    Loop for a in Actions
        /*average the score of each action over all chosen states*/
        Loop for s in Sample
            V(a) ←V(a) + RB-AlphaBeta(a(s), Agent, D, Alpha=−∞, Beta=∞)
        V(a) ← V(a) / K
    Select a with maximal V(a)
```

נשים לב, שהקריאה אינה רקורסיבית לפונקציה הנ"ל, אלא רקורסיבית ל-RB-alphabetha.
כלומר: מתבצעת השלמת מצבים רנדומלית, פעם אחת בהתחלה, ועליהם מבוצע החישוב באופן רגיל כפי שראינו עד כה. השלמת המצבים לפיכך תתבצע בexpectimax בצומת ההסתברותי (כל בן- מהווה מצב שהושלם), כאשר ההסתברות הינה יוניפורמית (הסתברות כל בן- |S_Complete|/1). צומת זה שקול למשל למידול הטלת קובייה כפי שראינו בשיעור, והוא משלים את המצבים האפשריים. צומת זה יהיה השורש של העץ ולא נמצא אותו באף שכבה אחרת בעץ (כלומר- הוא יופיע רק בתחילת חישוב האלגוריתם) . בצומת זה כמו כן יחושב הערך של המהלך האופטימלי לביצוע (הלולאה החיצונית על המהלכים + הלולאה הפנימית על הבנים). צומת שמייצג את תור השחקן שלנו יהיה צומת מקסימום, ואחרת (תור יריב, לא התור שלנו ולא צומת הסתברותי) הצומת הינו מינימום. אלו הם הצמתים שייצגו משחק תחת מצב שהשלמנו, כפי שקורה בלולאה המקוננת עם הקריאה ל RB-AlphaBeta.

4. הרעיון: נתחיל מK ו-D קטנים, ובכל איטרציה נגדיל אותם וכך נדגום יותר אפשרויות לקבלת תוצאה מדוייקת יותר וייצוג אמיתי של העולם. במידה ונגמר הזמן להרצה- נחזיר את הערך שקיבלנו באיטרציה האחרונה שרצה באופן מלא. האלגוריתם ינסה בהינתן הזמן שעומד לרשותו להגיע לפיתרון (גם אם לא האופטימלי), ואז ינסה מאיטרציה לאיטרציה לשפר פיתרון זה. האלגוריתם המוצע מתיישב עם עיקרון אלגוריתמי הAnytime, מכיוון שהוא כמו האלגוריתמים שנלמדו בכיתה נהיה מיודע ומדוייק יותר ככל שניתן לו יותר זמן לרוץ.

```
Function AnytimeMonteCarlo(PartialState, Agent)
    D = 1
    K = 1
    While Resources are still available:
        Ans = MonteCarloPartialInformation(PartialState, Agent, D, K)
        D++
        K++
    Return Ans
```

כאשר MonteCarloPartialInformation זה האלגוריתם שהוצג בכיתה וצילום מסך שלו מופיע בדף הקודם.