

# מבוא לבינה מלאכותית

## סמסטר חורף תשפ"ו

### מטלה 1

תאריך הגשה: 25.11.25 23:55

#### הנחיות

- שאלות בנושא מטלה זו יש לשאול דרך המודל, בפורום "מטלה 1".
- הוראות להגשת המטלה מופיעים בסוף מסמך זה.
- הקבצים הנדרשים להרצת הקוד הינם:
  - heuristics.py
  - main.py
  - color\_blocks\_state.py
  - search.py
  - search\_node.py
- העבודה להגשה בזוגות בלבד אלא אם כן המגישים קיבלו אישור להגשה שאינה בזוגות.
- לפני שניגשים לממש את המטלה מומלץ לעיין רבות בהסברים וכן בקוד הקיים.
- פתרון המטלה שתגישו ייבדק מול שאר ההגשות על ידי תוכנת העתקות.
- **מי שימצא כי העתיק יכשל בקורס וכן יועבר לוועדת משמעת אוניברסיטאית.**
- הפרויקט נכתב וייבדק בשפת התכנות python3.10.
- מסמך זה בנוי באופן הבא: תיאור המטלה, בעיית החיפוש, מרחב הבעיה, שאלות המטלה, פונקציות שיש להשלים והסבר על פונקציות קיימות.

#### תיאור המטלה

כדי להרשים את חבריו, דני בנה רובוט שממייין מגדל של קוביות צבעוניות. המגדל מורכב מקוביות כך ש-2 מהפאות הצדדיות צבועות בצבע אחד ו-2 הפאות האחרות צבועות בצבע אחר (אין צבע לפאה העליונה והתחתונה ולא נשתמש בהן). בכל נקודת זמן יש צבע אחד שפונה אל החברים (החלק הקדמי) וצבע שפונה הצידה מהחברים (החלק הצדדי). חבריו של דני יכול לראות רק את החלק הקדמי של מגדל הקוביות. הרובוט יכול לסובב קוביה אחת בכל מיקום במגדל כך שצבע אחר יוצג כלפי החברים (סיבוב של 90 מעלות). כמו כן, הרובוט יכול גם להפוך בזריזות את החלק התחתית של מגדל הקוביות כך שסדר וצבע הקוביות נשמר אך בסדר הפוך. כדי להפוך את תחתית המגדל הרובוט חייב לתפוס את הקובייה התחתית ביותר ועוד קוביה במגדל. הרובוט יהפוך רק את הקוביות שנמצאות בין הקוביות שתפס. אם הרובוט תפס את הקוביה התחתית ביותר ואת הקוביה העליונה ביותר, הוא יהפוך את כל המגדל כך שהקוביה העליונה תהיה בתחתית והקוביה התחתית תהיה בראש המגדל. כמו כן, כל הסדר בין קוביות יתפנה. כדי להפחית את הסיכוי שהרובוט יפיל את המגדל, עזרו לדני לממש אלגוריתם  $A^*$  כדי למזער את כמות הפעולות המופעלות על המגדל.

#### בעיית חיפוש

בהינתן מגדל קוביות צבעוניות וסידור הצבעים הגלויים הממויין (גם קלט). עלינו למצוא את רצף המצבים המינימלי שבעזרתו הרובוט יוכל למיין את המגדל המקורי שיציג את הסידור הצבעים הגלויים במצב הסופי.

## מרחב הבעיה

רשימה בגודל  $N$  של קוביות כך שלכל קובייה יש 2 צבעים:

$$\forall i_{1 \leq i \leq N}: color_1(cube_i) \in \mathbb{Z}, color_2(cube_i) \in \mathbb{Z}$$

לכל קובייה יש צבע אחד גלוי (החלק הקדמי) וצבע אחד נסתר (החלק הצדדי)

לכל קובייה יש שני צבעים שונים:

$$\forall i_{1 \leq i \leq N}: color_1(cube_i) \neq color_2(cube_i)$$

אך יכולות להיות 2 קוביות שונות עם צבע משותף.

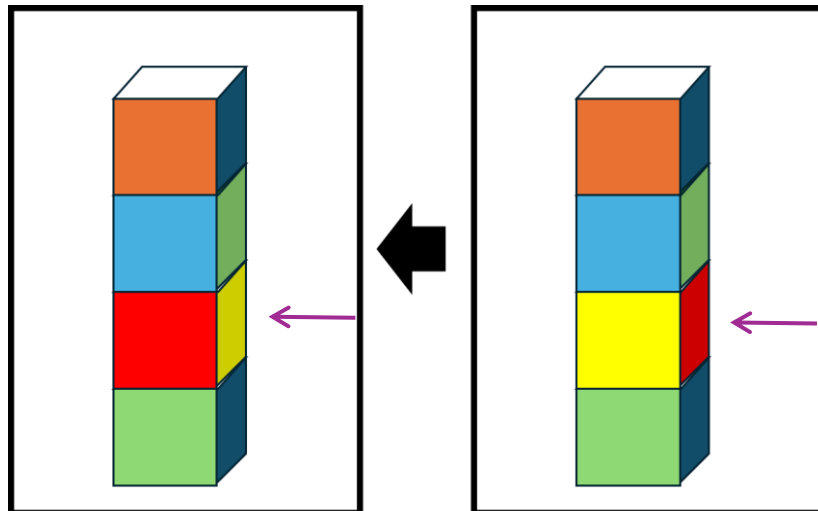
מצב התחלתי: רשימת קוביות צבעוניות לכל קובייה יש 2 צבעים ואחד הצבעים הוא הצבע הגלוי. הרשימה בנויה מזוג מספרים המסמלים את הצבעים של הקובייה כך שהמספר השמאלי הינו הצבע שמוצג והימני מוסתר. ראש המגדל הינו בצידו השמאלי של הרשימה והקובייה בתחתית המגדל בצד הימני של הרשימה.

מצב סופי: סדר הצבעים הגלויים של הקוביות.

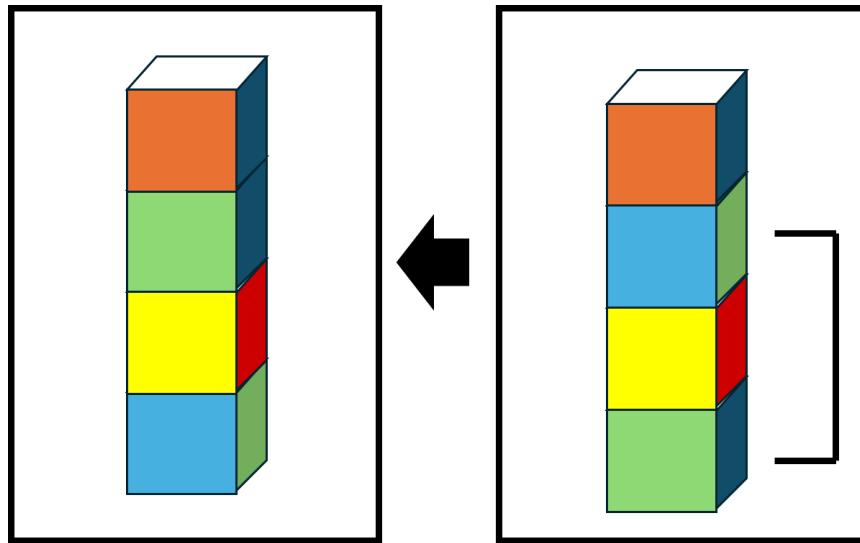
אופרטורים:

1. Spin-סיבוב של קובייה יחידה (90 מעלות) מבלי לשנות את סדר הקוביות במגדל.

דוגמא לפעולת סיבוב של קובייה (הקובייה השנייה מלמטה):



2. Flip- הפיכה של תת מגדל מתחתית המגדל.  
דוגמא לפעולת הפיכה:

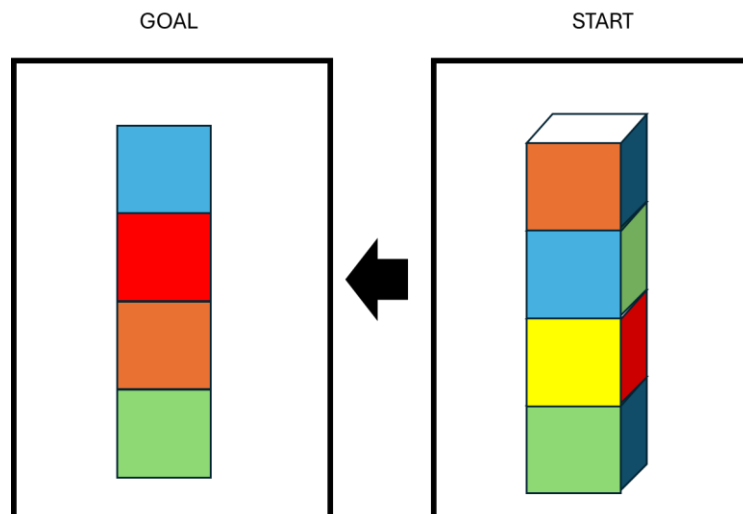


שימו לב: הסידור של כל הקוביות בתת ערימה מתהפך, בגלל שכמות הקוביות שהתהפכו אי זוגית, הקוביה הצהובה נשארה במקום. **עלות כל פעולה הינה 1.**

## שאלות המטלה

- א. עזרו לדני להשלים את הפונקציות החסרות (מצויינות בהמשך).
- ב. שנו את היוריסטיקה base heuristic כך שתחשב את היוריסטיקה הבאה- עבור כל זוג קוביות צמודות – אם קיימת קומבינציית צבעים של 2 הקוביות במצב הסופי – נוסף 0 ליוריסטיקה. אחרת, נוסף 1. קומבינציית צבעים הינה בחירת צבע מכל קוביה (סדר הקוביות לא חשוב).

**שימו לב:** הסדר של שני הצבעים אינו חשוב העיקר שקיים צימוד כזה במצב הסופי. לדוגמה:



עבור המצב ההתחלתי מימין והמצב הסופי משמאל, היוריסטיקה תחזיר  $0 + 0 + 1 = 1$ .  
זוג הקוביות העליון מוסיף 0 כי קיימת קומבינציה צבעים של כתום וירוק במצב הסופי. זוג  
הקוביות הבא גם מוסיף 0 מאחר ותכלית ואדום מופיע יחד במצב הסופי. עבור 2 הקוביות  
התחתיות: לא קיים מצב סופי שבו יש זוג של {ירוק, צהוב} או {ירוק, אדום} או {כחול, צהוב}  
או {כחול, אדום} או {סדר הפוך של הצבעים הצמודים} {צהוב, ירוק} או {אדום, ירוק} או  
{צהוב, כחול} או {אדום, כחול}. לכן נוסף 1 ליוריסטיקה.

ג. שנו את היוריסטיקה `advanced_heuristic` כך שתחשב היוריסטיקה של מצב מסויים  
בבעיה. על היוריסטיקה לשפר את היוריסטיקה מסעיף ב' (ימדד בזמן ריצה) ולשמור על  
אופטימליות הפתרון.  
הזוג שהריץ את הטסטים של סעיף ג' בזמן הממוצע הקצר ביותר (ימדד בהרצת הטסטים  
במודל ואולי בטסטים נוספים רק כדי להכריע את הבונוס), יקבל **2 נקודות בונוס לציון  
הסופי**.  
שימו לב שכל הטסטים צריכים לעבור כדי שהזמן יחושב.  
בנוסף, כדי למדוד את פונקציית היוריסטיקה בצורה מדויקת יותר, הוספנו בטסטים מימוש  
של קובץ ה-`search` ושם נקרא לפונקציות היוריסטיקה (והגדרת המצבים) שלכם כדי למדוד  
את זמן הריצה עם אותו מימוש של מבנה נתונים ולוגיקת חיפוש. גם כאן ינתנו **2 נקודות  
בונוס לציון הסופי**.  
זוגות שרוצים להשתתף בבונוס צריכים לרשום את תעודות הזהות שלהם והזמן שקיבלו  
בהרצה [בקבוצ הזה](#).  
(זוגות שלא רוצים להשתתף בבונוס צריכים לרשום רק את תעודת הזהות בלי הזמן שקיבלו)

עבור הבונוסים - במידה ויהיה קשה להבדיל בין זמני הריצה (למשל כמה צוותים יקבלו זמן ריצה של  
0.0) נרץ עבור הצוותים האלו טסטים קשים יותר (מספר שונה של קוביות וצבעים).  
הטסטים הנוספים לא משפיעי על הציון והציון שאתם רואים במודל הינו הציון לעבודה ולא נבדוק  
אותכם יותר מכך.

## הפונקציות שיש להשלים למימוש כל אחד מאלגוריתמי החיפוש

`heuristics.py`

`init_goal_for_heuristics(goal_blocks)`

הפונקציה מקבלת ייצוג של המצב הסופי כמחרוזת ולא מחזירה כלום.

שימו לב: הפונקציה הזו תקרא לפני כל ריצה חדשה – תוודאו לאפס את מבני הנתונים במידה  
והשתמשתם

`base_heuristic(_color_blocks_state)`

הפונקציה מקבלת אובייקט מסוג `color_blocks_state` ומחזירה את ערך היוריסטיקה עבור מגדל  
הקוביות בקלט. (יש למחוק את השורה שמחזירה 0)

advanced\_heuristic (\_color\_blocks\_state)

הפונקציה מקבלת אובייקט מסוג color\_blocks\_state ומחזירה את ערך היוריסטיקה עבור מגדל הקוביות בקלט. (יש למחוק את השורה שמחזירה 0)

color\_blocks\_state.py

init\_goal\_for\_search(goal\_blocks)

הפונקציה מקבלת ייצוג של המצב הסופי כמחרוזת ולא מחזירה כלום.

שימו לב: הפונקציה הזו תקרא לפני כל ריצה חדשה – תוודאו לאפס את מבני הנתונים במידה והשתמשתם

is\_goal\_state(\_color\_blocks\_state)

פונקציה סטטית המקבלת מצב של מגדל קוביות ומחזירה אמת אם המצב הינו מצב סופי, אחרת תחזיר שקר.

get\_neighbors(self)

הפונקציה לא מקבלת קלט ומחזירה רשימה של סדורות (tuples) כך שבכל סדורה יש את אובייקט מסוג color\_blocks\_state המייצג את אחד השכנים ועלות המעבר אל אותו שכן.

[(color\_blocks\_state1, cost1), (color\_blocks\_state2, cost2)...]

get\_state\_str(self)

הפונקציה לא מקבלת קלט ומחזירה את ייצוג המחרוזת של מגדל הקוביות (אין צורך לשנות את הפונקציה) - פונקציה זו לא חובה למימוש. תוכלו להיעזר בה כדי לדבג את הקוד.

search.py

create\_open\_set()

הפונקציה אינה מקבלת ערכים, ומחזירה מבני נתונים מתאים לאלגוריתם החיפוש.

create\_closed\_set()

הפונקציה אינה מקבלת ערכים, ומחזירה מבני נתונים מתאים לאלגוריתם החיפוש.

add\_to\_open(vn, open\_set)

הפונקציה מקבלת כקלט את מבני הנתונים open ומצב vn ומכניסה את vn ל-open.

open\_not\_empty(open\_set)

הפונקציה מקבלת כקלט את מבני הנתונים open ומחזירה אמת אם הוא לא ריק, אחרת, מחזירה שקר.

`get_best(open_set)`

הפונקציה מקבלת כקלט את מבני הנתונים ומחזירה את הקודקוד הטוב ביותר, על פי אלגוריתם החיפוש.

`add_to_closed(vn, closed_set)`

הפונקציה מקבלת כקלט את מבני הנתונים `closed` ומצב `vn` ומכניסה את `vn` ל-`closed`.

`duplicate_in_open(vn, open_set)`

הפונקציה מקבלת כקלט את מבני הנתונים `open` ומצב `vn` ומחזירה אמת אם קיים קודקוד ב `open` עם מצב זהה ל `vn.state` וערך ה `g` שלו קטן או שווה ל `vn.g`. אם `vn.state` לא קיים ב `open` או ערך ה `g` של המצב ב `open` גדול מ `vn.g`, הפונקציה תחזיר שקר.

`duplicate_in_close(vn, closed_set)`

הפונקציה מקבלת כקלט את מבני הנתונים `closed` ומצב `vn` ומחזירה אמת אם קיים קודקוד ב `closed` עם מצב זהה ל `vn.state` וערך ה `g` שלו קטן או שווה ל `vn.g`. אם `vn.state` לא קיים ב `closed` או ערך ה `g` של המצב ב `closed` גדול מ `vn.g`, הפונקציה תחזיר שקר.

## מידע על פונקציות ממומשות

`search_node.py`

`get_neighbors(self)`

הפונקציה לא מקבלת קלט ומחזירה את כל המצבים השכנים של המצב הנוכחי.

`search.py`

`print_path(path)`

הפונקציה מקבלת מערך של קודקודים מסוג `search_node` ומדפיסה את המחרוזת המייצגת את המצב של מגדל הקוביות.

`search(start_state, heuristic, goal_state)`

הפונקציה מקבלת את המצב ההתחלתי, פונקציית היוריסטיקה ומצב סופי ומחזירה מערך של `search_node` המייצגים את המסלול האופטימלי מהמצב ההתחלתי למצב הסופי. בתא הראשון במערך יהיה המצב ההתחלתי בתא הבא יהיה השכן שלו במסלול שהוחזר וכך הלאה עד המצב הסופי בתא האחרון.

## הגשת המטלה

- יש להגיש דרך מערכת המודל את ארבעת הקבצים הבאים:
  - heuristics.py
  - color\_blocks\_state.py
  - search.py
  - search\_node.py

הגשת המטלות תתבצע ישירות מול מערכת המודל בצורה אלקטרונית.

שימו לב: ישנה אפשרות להגיש את המטלה התכנותית מספר פעמים ובכל הגשה לקבל חיווי, כלומר תקבלו באופן מיידי את הציון לביצוע התרגיל.

ניתן לראות את הפידבק להרצה (קומפילציה, מספר טסטים שעברו, שגיאות זמן ריצה וכו'...).

לאחר סיום ההרצה יתקבלו התוצאות. ישנם מספר טסטים הבודקים את הפתרון המוגש למטלה. אם התקבלה שגיאת קומפילציה יש עליכם להעלות קובץ חדש בכדי לקבל ציון לאחר התיקונים. בקבצים המקוריים שקיבלתם עם המטלה עלולות להיות שגיאות ריצה עבור פונקציות לא ממומשות.

## טיפים לעבודה

במידה שהטסטים ב VPL לא עוברים, אין סיבה להסיק שיש בעיה בטסטים (זה יכול לקרות אך הסבירות נמוכה). יש לבדוק את הקוד שלכם קודם:

- האם הרצתם דוגמאות פשוטות ב main?
- האם וידאתם שהיוריסטיקה שלכם מחזירה את הערך שציפיתם לו?
- האם בדקתם את תקינות היוריסטיקה המתקדמת?
- אם יש לכם בעיות זמן ריצה- האם השתמשתם במבני נתונים יעילים? למשל- מה הסיבוכיות של בדיקת כפילויות ב OPEN? האם אפשר לייעל?

## בהצלחה!