29/04/2007

Workshop in Reinforcement Learning – Final Report

Application of Reinforcement Learning to the Game of *Mancala*

אוריאל לבנטל
arielio90210 at gmail.com

אורי להב
orilahav at gmail.com

בחרנו ליישם את טכניקת ה-Reinforcement Learning במשחק Mancala. במסמך זה נציג בקצרה את המשחק עצמו, את אלגוריתם הלמידה וייצוג המשחק בו הוא משתמש ואת תהליך הלמידה. נעמוד על התוצאות אליהן הגענו בסיום, ונשים דגש על השיפורים שנעשו לאורך הדרך והסיבות להם, והקשיים בהם נתקלנו. בחלקו השני של המסמך נציג את עיקר פרטי המימוש של תהליך הלמידה והמשחק.

1. חלק ראשון – תיאור הפרויקט

1.1. תיאור קצר של הפרויקט

במסגרת הפרויקט ניסינו ליישם את שיטות ה-Reinforcement Learning שלמדנו במשחק Mancala. תחילה כתבנו תוכנת GUI למשחק זה עבור שני שחקנים אנושיים. לאחר מכן, מימשנו מספר שחקנים פשוטים. חלקו העיקרי של הפרויקט היה התאמת אלגוריתם הלמידה שלמדנו למשחק זה, ובפרט בחירת הייצוג של המשחק. לאחר מכן ניסינו להעריך את טיב התוצאות אליהן הגענו ולשפר אותן ע"י עידון הפרמטרים באלגוריתם הלמידה, התאמות נוספות בתהליך הלמידה ושיפור ייצוג המשחק.

הפרויקט כתוב ב-Matlab (כולל GUI, למרות שה-GUI הראשוני שהגשנו היה בשפת #C).

1.2. תיאור הבעיה – משחק Mancala

Mancala הוא משחק לשני שחקנים. זהו אחד המשחקים העתיקים בעולם, ומוערך כבן 7000 שנים. המילה "Mancala" באה מערבית, ופרושה – להעביר. זו הפעולה הנעשית במשחק בהעברת האבנים (שלעתים מכונות: snails, חלזונות) מגומה לגומה סביב הלוח. בלוח שתי שורות מקבילות של גומות, שש גומות בכל שורה ומיכל (גומה גדולה) בקצה השורה. למשחק יש מאות וריאציות והוא נפוץ בכל העולם, במיוחד באפריקה. אנו בחרנו להתמקד בגרסה מפורסמת אחת.

יש לציין כי Mancala הוא משחק דטרמיניסטי, ואין בו כלל אלמנטים של מזל.

מטרתנו העיקרית הייתה לנסות ולייצר שחקן Mancala ממוחשב שיהווה אתגר רציני לשחקני Mancala אנושיים. זאת תחת מגבלה של זמן חישוב, שכן, מבלי לכמת את הדברים, היינו מעוניינים שהשחקן הממוחשב יבצע את תורו בזמן סביר.

1.2.1. תמצית חוקי המשחק

בכל גומה מניחים שש אבנים. כל שחקן בתורו בוחר גומה בצידו ומחלק את כל האבנים מהגומה בה בחר, לגומות הרצופות מימין ובהמשך נגד כיוון השעון, אבן לכל גומה, כולל אבן במיכל של אותו השחקן. במידה והוא מסיים בגומה ריקה הוא לוקח למיכלו את כל האבנים שבגומת היריב המקבילה ("capture"). במידה והוא מסיים במיכל שלו, המשתתף מקבל תור נוסף. ממשיכים במשחק, והזוכה הוא זה שברשותו יותר אבנים במיכל כשאי אפשר לשחק יותר (כאשר כל הגומות בצדו של השחקן שתורו לשחק ריקות).

פירוט מלא של חוקי המשחק ניתן למצוא ב-
http://boardgames.about.com/cs/mancala/ht/play_mancala.htm

1.3. פתרונות בסיסיים

ראוי תחילה לציין, שאין ביכולתנו לפתור את המשחק באופן מלא ולהציע מהלך מנצח לכל עמדה אפשרית, מאחר ועץ ההחלטות במשחק הוא עצום (אורך משחק של 30 תורות יניב עץ בעל 6^{30} עלים). לקראת המימוש של שיטת ה-Reinforcement Learning מימשנו מספר שחקנים ממוחשבים ובחנו את תוצאותיהם כנגד עצמם וכנגד משתמש אנושי.

1.3.1. שחקנים בסיסיים שמומשו

1.3.1.1. שחקן רנדומלי

שחקן זה בוחן את לוח המשחק, מזהה את האפשרויות לשחק (גומות הלוח בו יש לו אבני משחק) ובוחר ביניהן באופן רנדומלי.

1.3.1.2. שחקן היוריסטי

בנינו שחקן זה לאחר ששיחקנו במשחק פעמים רבות וניסינו לפשט מאוד את הכללים לבחירת המהלך הבא. שחקן זה בוחר תחילה את המהלך שיאפשר לו תורות נוספים רבים ככל האפשר (חישוב רקורסיבי), ואם קיימים מספר כאלה בוחר ביניהם את הראשון. במידה ולא קיימים מהלכים שיתנו תור נוסף בוחר השחקן את המהלך שיגדיל ככל שניתן את ההפרש במיכלים, ואם קיימים מספר כאלה בוחר ביניהם את האחרון. במידה ולא קיימים תורות שיגדילו את ההפרש במיכלים, בוחר השחקן את הגומה הימנית ביותר האפשרית (הקרובה יותר למיכל, במטרה לקדם את האבנים לעבר המיכל בתורות הבאים).
הבעייתיות העיקרית של שחקן זה היא התעלמותו מהמשך המשחק והעבודה שהוא כלל אינו מסתכל קדימה ולא מנתח את יכולת היריב לצבור אבנים בתורו שלו.

1.3.1.3. השחקן החמדן

שחקן זה בוחר לבצע את המהלך שיגדיל כל שניתן את ההפרש בין המיכלים. במידה וקיימים מספר מהלכים כאלה, הוא בוחר ביניהם באופן אקראי.
בנוסף לכך, במידה והשחקן מזהה מהלך מנצח (שמוביל לסיום המשחק) הוא מעדיף אותו על פני כל המהלכים האחרים, ובמקביל נמנע מלבצע מהלך מפסיד. שיפור זה הוכנס בשלבים המאוחרים של העבודה על הפרויקט ונראה כמועיל.
מדיניות זו מולבשת על עץ MinMax בעומקים 1-5 וכך למעשה נוצרים 5 שחקנים בסיסיים. על מנת לזרז את ההחלטה נעשה שימוש ב-alpha beta pruning. עומק ה-MinMax קובע את מספר הצעדים שנסתכל קדימה, כאשר הם יכולים להיות צעדים של אותו השחקן (במקרה של תור נוסף) או של היריב (כמובן שההנחה היא שהיריב פועל באותה האסטרטגיה, כלומר חמדן לגבי ההפרש בין המיכלים).
הגבלת העומק לעד 5 מהלכים קדימה נובעת משיקולי זמן חישוב.

1.3.2. השוואת יכולות השחקנים הבסיסיים

1.3.2.1. נגד עצמם

טבלה זו מרכזת את התוצאות של 7 השחקנים הבסיסיים במשחקים כנגד עצמם. בכל תא בטבלה זוג (אחוזי ניצחונות, ואחוזי תוצאות תיקו) המתאים לתוצאות 100 משחקים כאשר השחקן שבכותרת השורה המתאימה הוא הראשון והשחקן שבכותרת העמודה המתאימה הוא השחקן השני. מבנה זה של טבלת תוצאות ישמש גם בהמשך לניתוח השחקן שנוצר ע"י Reinforcement Learning.

		Random	Heuristic	Greedy				
				1	2	3	4	5
Random		42% / 4%	0% / 0%	2% / 1%	0% / 0%	0% / 0%	0% / 0%	0% / 0%
Heuristic		100%/0% *	100% / 0%	99% / 0%	100% / 0%	95% / 0%	49% / 22%	81% / 0%
Greedy	**1**	97% / 1%	11% / 1%	57% / 5%	6% / 2%	5% / 3%	6% / 1%	3% / 0%
	2	100% / 0%	80% / 0%	100% / 0%	71% / 3%	69% / 5%	32% / 7%	31% / 3%
	3	100% / 0%	86% / 4%	99% / 1%	86% / 6%	81% / 1%	40% / 5%	54% / 6%
	4	100% / 0%	100% / 0%	100% / 0%	88% / 2%	85% / 4%	64% / 8%	54% / 6%
	5	100% / 0%	100% / 0%	100% / 0%	97% / 2%	93% / 3%	98% / 1%	89% / 4%

* משחק זה הוא משחק דטרמיניסטי יחיד, מאחר ואין אלמנט רנדומי כלשהו בהגדרת השחקן ההיוריסטי.

1.3.2.2. נגד משתמש אנושי

שיחקנו מספר משחקים כנגד השחקנים הבסיסיים:

- כשחקן ראשון לא קשה לנצח את כל 7 השחקנים הבסיסיים.
- כשחקן שני פשוט לנצח את השחקן הרנדומלי, והשחקן החמדן בעומק 1. לא פשוט לנצח את יתר השחקנים, וקשה מאוד (ואולי אף בלתי אפשרי, לא הצלחנו!) לנצח את השחקן ההיוריסטי והשחקן החמדן בעומקים 4-5.

1.3.2.3. נגד שחקני אינטרנט זמינים

נעזרנו באתר http://www.rocketsnail.com/mancala/game.htm המציע משחקי Mancala בשלוש רמות שונות. בחנו 2 מהשחקנים הבסיסיים (הטובים יותר) בשני משחקים כנגד הרמה הגבוהה ביותר באתר:

- השחקן ההיוריסטי ניצח בתור שחקן ראשון.
- השחקן ההיוריסטי הפסיד בתור שחקן שני.
- השחקן החמדן בעומק 5 ניצח בתור שחקן ראשון.
- השחקן החמדן בעומק 5 הפסיד בתור שחקן שני (בפער קטן).

1.3.3. מסקנות ראשוניות מתוצאות הפתרונות הבסיסיים

תוצאות בסיסיות אלו מאפשרות להסיק מספר מסקנות שעשויות לסייע בניתוח המשחק ותוצאות המשחקים בהמשך, וחלקן מנוגדות לאינטואיציה.

1.3.3.1. קל מאוד לנצח שחקן רנדומלי

ניכר כי השחקן הרנדומלי, כצפוי, חלש באופן משמעותי מיתר השחקנים. שחקן זה שימש אותנו על מנת לאתר שגיאות בקוד תהליך הלמידה, שכן למידה בסיסית וקצרה אמורה לאפשר לנצח בקלות שחקן רנדומלי.

1.3.3.2. רצפי תורות נוספים ארוכים

השחקן ההיוריסטי הפתיע אותנו בטיבו, ובמיוחד במשחקים נגד שחקן אנושי ונגד השחקן מהאינטרנט. כאשר בוחנים את משחקיו נוכחים לדעת כי לעיתים הוא מוצא רצפי תורות אורכים מאוד (למשל 8 תורות רצופים) בהם הוא צובר יתרון משמעותי. אנו למדים מכאן שלבונוס התור הנוסף משמעות רבה.

1.3.3.3. יתרון לשחקן הראשון

במשחק של שני שחקנים רנדומליים אין יתרון משמעותי לשחקן הראשון. לעומת זאת, במשחקים של יתר השחקנים ניתן לראות כי לשחקן הראשון יתרון משמעותי. הדבר בולט במיוחד כאשר נשווה למשל את זוג המשבצות המסומנות ברקע צהוב בטבלה.
בנוסף, יתרון זה הורגש על ידינו במהלך היכרותנו הבסיסית עם המשחק כשחקנים אנושיים. (הדבר מתבטא גם בתוצאות השחקן שנוצר ע"י Reinforcement Learning, וניתנת לכך התייחסות בהמשך.)

1.3.3.4. העדרה של טרנזיטיביות ביחס "טוב מ"

אינטואיטיבית, צפינו כי אם שחקן א' טוב משחקן ב' ואילו שחקן ב' טוב משחקן ג', אז שחקן א' טוב משחקן ג'. התוצאות המרוכזות בטבלה מראות כי לא תמיד כלל זה מתקיים (ראה למשל את המשבצות המסומנות ברקע כחול בטבלה).

1.3.3.5. מונוטוניות חלקית בלבד בטיבם של השחקנים החמדנים

אינטואיטיבית, צפינו כי ככל שעומק ה-MinMax בשחקן החמדן נקבל שחקן טוב יותר. אנו רואים כי לא תמיד כך המצב אפילו במשחקים של השחקנים החמדניים בעומקים השונים כנגד עצמם (ראה את המשבצת המסומנת ברקע אדום בטבלה), ובאופן חריף יותר במשחק נגד השחקן ההיוריסטי (ראה שוב את המשבצות המסומנות ברקע כחול בטבלה). אנו נוכחים לדעת, למשל, כי חיזוי 5 מהלכים קדימה (כמובן, תחת ההנחה שהיריב פועל באותה המדיניות) הוא לעיתים רע יותר מחיזוי של 4 מהלכים.

1.4. הצגת הבעיה כבעיית Reinforcement Learning

משחק Mancala יכול להיות מתואר כשרשרת מרקוב כאשר המצבים הן עמדות הלוח (מיקומי 48 אבני המשחק בגומות ובמיכלים), והמעברים הן הפעולות המבוצעות ע"י השחקנים. פעולת היריב אינה ידועה מראש, ועל כן המעבר למצב היעד אינו דטרמיניסטי, אלא הסתברותי. בסיום המשחק אחת מבין 3 תוצאות אפשריות: ניצחון, תיקו או הפסד, והן יתורגמו למונחי reward. בכל זמן במשחק מצב הלוח ידוע לחלוטין.

נקודה נוספת שראוי לשים לה לב, הוא שלא ייתכן שבמשחק Mancala חוזר מצב המשחק לאותו מצב שהוא היה בו קודם לכן. במונחי שרשרת מרקוב, לא נצפה שיהיו קיימים מעגלים כלשהם בשרשרת.

הערה: הגדרות מדויקות יותר של מצב במשחק ומעבר בין מצבים, והשיקולים לבחירתן מוצגת בסעיפים: 1.5.2.3 ו-1.6.3.

1.5. אלגוריתם הלמידה ושיטת ייצוג המשחק

1.5.1. אלגוריתם הלמידה

בחרנו באלגוריתם Q-Learning (Watkins, 1989), המתבסס על עקרונות שיטת ה-Temporal Difference אותן למדנו בשיעורי הסדנה. כלל העדכון המרכזי של אלגוריתם זה הוא:

$$Q(s_t, a_t) \leftarrow Q(s_t, a_t) + \alpha \left[r_{t+1} + \gamma \max_a Q(s_{t+1}, a) - Q(s_t, a_t) \right]$$

היתרון באלגוריתם זה הוא ביצוע הלמידה **ללא** בניית המודל המרקובי המלא של המשחק.

rewards הוגדרו כ-0 לכל המצבים, למעט המצבים המנצחים (תם המשחק ובמיכל הימני יש יותר אבנים מאשר במיכל השמאלי) שהוגדרו כ-1, והמצבים המפסידים (תם המשחק ובמיכל הימני יש פחות אבנים מאשר במיכל השמאלי) שהוגדו כ-1-. למצבי התיקו (תם המשחק ובמיכל הימני מספר אבני משחק שווה לזה שבמיכל השמאלי) הוגדר reward 0.

1.5.2. ייצוג המשחק והפונקציה לחישוב ערכו של מצב

מאחר ובמשחק 48 אבנים ו-14 גומות שונות, מספר עמדות המשחק השונות (מצבי הלוח) הוא עצום, ולא ניתן לטפל בו באמצעות Lookup Table. כפי שלמדנו בכיתה, פתרון אפשרי לכך הוא הגדרת מספר תכונות (features) של המשחק, כך שחישוב ערכו של מצב יעשה כממוצע משוקלל בין תכונות אלו. המשקלים שישמשו בביצוע החישוב הם תוצר תהליך הלמידה. פונקצית חישוב הערך תהא פונקציה לינארית של ערכי תכונות אלו, והערכת הפונקציה לאורך הלמידה יעשה באמצעות Gradient Descent.

1.5.2.1. AfterStates

כאשר השחקן מבצע פעולה, הוא יודע בוודאות את מצב הלוח לאחר ביצוע הפעולה שלו (המשחק דטרמיניסטי). פורמלית, מצב זה (למעט במקרה של תור נוסף) אינו מופיע במודל המרקובי, שכן אינו מצב בו על השחקן לבצע פעולה (זהו תורו של היריב). יחד עם זאת סביר יותר להשתמש בלוח זה שכן הוא מתאים יותר לחישוב ערכה של הפעולה שביצענו. בספר הלימוד (Reinforcement Learning, MIT Press, 1998) מוצעת טכניקה זו המתאימה למשחקים מסוג זה, והמצבים הללו נקראים שם: AfterStates. למעשה, בהינתן מצב ופעולה, אנו מחשבים את תכונות הלוח **לאחר** ביצוע הפעולה ולא לפניה.

1.5.2.2. התכונות שנבחרו

מצאנו לנכון לחלק רעיונית את טיבו של מצב מסוים ל-3 חלקים:

1. פער נוכחי- כחלק ממצב הלוח נמצאים גם מספר האבנים במיכלים. ההפרש ביניהם (וכמובן הסימן שלו) מעידים על הפער הנוכחי במשחק. ברור שמצבים בהם הפער גדול עדיפים ממצבים בהם הפער קטן (או שלילי גדול).
2. סכנה שנשקפת- סכנה במצב נובעת מיכולתו של היריב לצבור אבני משחק, אם באמצעות תורות נוספים, ואם באמצעות ביצוע capture של אבני משחק של השחקן השני.
3. פוטנציאל להמשך- פוטנציאל להמשך הוא היכולת של השחקן לצבור אבני משחק, אם באמצעות תורות נוספים, ואם באמצעות ביצוע capture של אבני משחק של השחקן השני.

קיימת סימטריה בין תכונות המעידות על סכנה עבורי, ותכונות המעידות על פוטנציאל המשך עבור היריב. סימטריה זו נוצלה בחישוב התכונות (מבוצע חישוב תכונות על לוח המשחק הנוכחי, ולאחריו חישוב זהה על לוח משחק הפוך).

לאורך העבודה ניסינו לשפר את רשימת התכונות שבשימוש, ובסופו של דבר עבדנו עם הרשימה שלהלן:

1. האם (0 או 1) מצב הלוח מאפשר לשחקן התחתון לקבל תור נוסף?
2. האם (0 או 1) מצב הלוח מאפשר לשחקן העליון לקבל תור נוסף?
3. האם (0 או 1) מצב הלוח מאפשר לשחקן התחתון לבצע פעולה שתעבור במיכל שלו?
4. האם (0 או 1) מצב הלוח מאפשר לשחקן העליון לבצע פעולה שתעבור במיכל שלו?
5. המספר המקסימלי של אבני משחק שנמצאות מול גומה ריקה (ולכן, עשויות להיכבש) בצד העליון.
6. המספר המקסימלי של אבני משחק שנמצאות מול גומה ריקה (ולכן, עשויות להיכבש) בצד התחתון.
7. מספר הכיבושים המקסימלי שמאפשר מצב הלוח לשחקן התחתון.
8. מספר הכיבושים המקסימלי שמאפשר מצב הלוח לשחקן העליון.
9. האם (0 או 1) השחקן התחתון יכול בפעולה אחת למלא את כל גומות היריב (ובכך כמעט למנוע ממנו את האפשרות לכבוש בתור הבא)?
10. האם (0 או 1) השחקן העליון יכול בפעולה אחת למלא את כל גומות היריב (ובכך כמעט למנוע ממנו את האפשרות לכבוש בתור הבא)?
11. המספר המקסימלי של אבני משחק הניתנות לכיבוש ע"י השחקן התחתון.
12. המספר המקסימלי של אבני משחק הניתנות לכיבוש ע"י השחקן העליון.
13. מספר האופציות לשחק עבור השחקן התחתון (מעיד גם על מספר הגומות הריקות).
14. מספר האופציות לשחק עבור השחקן העליון (מעיד גם על מספר הגומות הריקות).
15. האם (0 או 1) התא הסמוך למיכל ריק בצד התחתון (תנועה ממנו מבטיחה צבירת אבן נוספת)?
16. האם (0 או 1) התא הסמוך למיכל ריק בצד העליון (תנועה ממנו מבטיחה צבירת אבן נוספת)?
17. האם השחקן התחתון יכול לנצח בתורו הבא?
18. האם השחקן העליון יכול לנצח בתורו הבא?
19. ההפרש הנוכחי בין שני המיכלים.

1.5.2.3. התאמת התכונות לתור השחקן

נקודה מהותית, שעלתה כבר בשלבים הראשונים של העבודה על הפרויקט, היא הקשר שבין השחקן המשחק לערך המצב בהתאם לערכי התכונות שלעיל. אם נבחן את התכונות בתורו של השחקן התחתון, אז סביר שלתכונות הקשורות לפוטנציאל השחקן העליון (סכנה עבור השחקן התחתון) תהיה חשיבות רבה, שכן הן מעידות על פוטנציאל מיידי (בשל השימוש ב-afterstates). לעומתן, לתכונות הקשורות לפוטנציאל של השחקן התחתון (סכנה של העליון) תהיה חשיבות פחותה, שכן לאחר חישוב הערכים (לאחר פעולת התחתון) השחקן העליון עומד לשחק ולשנות את מצב הלוח, ולכן לשנות גם את ערכי התכונות הקשורות לאיומים עליו, ואולי במידה כזאת כך שהשחקן התחתון כלל לא יוכל לממש איומים אלו. באופן מקביל, אם נבחן את התכונות בתורו של השחקן העליון ניתקל בבעיה סימטרית.
הדבר מהותי מאוד במשחק Mancala בשל אופיו הסטוכסטי: כל מהלך משפיע מאוד על הלוח, ולכן פונקציית חישוב ערכי התכונות משנה באופן קיצוני את ערכיה בעקבות מהלכים במשחק (אינטואיטיבית: זוהי פונקציה "מאוד לא רציפה").

הפתרון הראשון שהעלינו לבעיה זו היה לחשב את ערכי התכונות רק לפני תורו של השחקן התחתון, או רק לפני תורו של העליון. בצורה זו נייחס אולי חשיבות פחותה למחצית מהתכונות (אלה הקשורות לפוטנציאל או לסכנה, בהתאם לשחקן שבתורו אנו מחשבים את התכונות), אך נימנע מערבוב מסוכן, שיגרום לשגיאות בהערכת משקלן של תכונות מסוימות. גישה זו לא עמדה במבחן התוצאה – ראה סעיף "סוגיית התור הנוסף" 1.6.3.

גישה אפשרית אחרת, בה בחרנו לנקוט, היא שכפול התכונות כולן לשני מצבים: תורו של השחקן התחתון / תורו של השחקן העליון. מצב המשחק יוגדר מעתה כמצב לוח + השחקן שתורו לשחק. וקטור התכונות ייבנה משני חלקים: החלק הראשון יחושב כאשר תור השחקן התחתון לשחק ויאופס כאשר תור העליון, והחלק השני יחושב כאשר תור השחקן העליון לשחק ויאופס כאשר תור התחתון. הדבר שקול ללמידת שני וקטורי משקלים, האחד מתאים לשווי התכונות לפני תורו של התחתון והאחר לשווי התכונות לפני תורו של העליון. לאור זאת נצפה שמשקלי התכונות המודדות את פוטנציאל התחתון בחלק הוקטור המחשוב בתורו של הראשון יהיו קטנים יחסית לאלו המודדות את הסכנה עבורו, ובאופן סימטרי בחלק הוקטור השני.

תוספת השחקן להגדרת מצב המשחק מאפשרת לנו להכניס תכונה נוספת והיא ערך השחקן (תחתון = 0 ועליון = 1). נצפה ששחקן יעדיף מצבים בהם תורו לשחק על פני כאלה בהם תורו של היריב (כלומר, משקל התכונה הזו צפוי להיות שלילי).

בנוסף לכך בחרנו להוסיף גם תכונה קבועה (ערכה 1 תמיד) שעשויה לסייע בהערכת יתרונו של השחקן הראשון במשחקי Mancala.

לסיכום, הגדרנו 40 (= 19 * 2 + 1 + 1) תכונות שמתאימות לכל מצב של המשחק. ריכוז התכונות והמשקלים שנמצאו עבורם נמצא בסעיף 1.7.1.

1.5.2.4. תכונות שבוטלו במהלך שיפור הפרויקט

במהלך העבודה על הפרויקט ניסינו מספר רב של תכונות נוספות, ובסופו של דבר החלטנו להשתמש ברשימה הנ"ל. ביטול תכונות מסוימות נעשה על מנת לקצר את זמן הלמידה הנדרש (תכונות רבות ידרשו למידה ארוכה מאוד) ולהאיץ את חישוב התכונות וערכי המצבים במהלך הלמידה. התכונות העיקריות עליהן החלטנו לוותר:

1. מספר אבני משחק בכל גומה בלוח (12 תכונות)- בוטלה מאחר ונראה כי אין לזה משמעות רבה כאשר הפונקציה לינארית.
2. מספר התורות הנוספים הגדול ביותר הניתן להשגה לכל אחד מהשחקנים- למרות שאלמנט זה נמצא מאוד מוצלח במימוש השחקן ההיוריסטי, החלטנו לוותר עליו בשל הזמן הרב יחסית הנדרש לחישוב זה.
3. מספר אבני המשחק הגדול ביותר שיכול השחקן להשיג ברצף התורות הנוכחי שלו- למרות שאלמנט זה נמצא יחסית מוצלח במימוש השחקן ההיוריסטי, החלטנו לוותר עליו בשל הזמן הרב יחסית הנדרש לחישוב זה.
4. הפרש בין אבני המשחק בין שני הצדדים- בוטלה מאחר ונראה כי אין לה משמעות רבה.

1.5.2.5. נרמול ערכי התכונות

עקרונית, מאחר ואנו לומדים ב-Reinforcement Learning את משקלה של כל תכונה, אין צורך בהיותם של ערכי התכונות "מנורמלים" (במובן כלשהו). יחד עם זאת, מצאנו לנכון להביא כל תכונה לנוע בטווח (+1) – (-1), ע"י חלוקה בערכה המקסימלי האפשרי. נרמול זה סייע לנו להבין את משמעותו של משקל התכונה שנלמד.

בשלביה הראשונים של העבודה, כאשר הרגשנו שנקודה זו עשויה להיות בעייתית, ניסינו אף לבצע נרמול חכם יותר. בחנו מספר רב של משחקים וברנו את הממוצע וסטיית התקן של ערכה של כל תכונה לאורך המשחקים כולם. באופן זה חשבנו לתקנן את ערכי התכונות, וכך להבטיח שמשקל התכונה שנלמד יצביע ישירות על חשיבותה ביחס לתכונות האחרות. ניסיון זה לא הוביל לתוצאות מיידיות, והחלטנו לוותר על גישה זו. ייתכן והסיבה לכך נעוצה בכך שקשה שהמדיניויות בהן משחקים שני השחקנים במשחק משפיעות על סטטיסטיקת ערכי התכונות, ולא ניתן לחשב סטטיסטיקה זו באופן שיתאים לכלל המשחקים.

1.6. תהליך הלמידה

1.6.1. תיחול הלמידה

על מנת שלא להסתכן בפגיעה בטיב הלמידה החלטנו לאתחל את כל המשקלים הנלמדים ל-0. ייתכן ושילמנו בזמן למידה ארוך מהנדרש, וניתן היה להתחיל במשקלים שנראים סבירים יותר. כתוצאה מאתחול זה השחקן הראשוני הוא שחקן גרוע מאוד.

1.6.2. קביעת ערכי הפרמטרים

1.6.2.1. (alpha) Learning Rate

לאחר מספר הרצות של תהליך הלמידה בערכי alpha שונים, בחרנו לנקוט במדיניות הבאה: alpha יאותחל ל-0.001 (אלפית) ונקטין אותו פי 2 ארבע פעמים במהלך תהליך הלמידה (בהפרשים קבועים). הערך נקבע כך שיתאים לערך המקסימלי של ערכי התכונות (ערך גבוה מידי גרם להתבדרות של וקטור המשקלים לאינסוף). בחרנו להתחיל עם ערך גדול על מנת לייעל את תהליך הלמידה, ולהקטין אותו על מנת לנסות להבטיח את זיהוי האופטימום במדויק. השפעתו של alpha על תהליך למידת המשקלים ניכרת, וניתן לזהות את הירידה ההדרגתית בערכו בהתבוננות בגרף למידת המשקלים (ראה סעיף 1.7.2).

1.6.2.2. (epsilon) Exploration Rate

על מנת להבטיח כיסוי של מרחב המצבים כולו נדרש בתהליך הלמידה לבצע מידי פעם פעולה שאינה ע"פ המדיניות הנלמדת, אלא אקראית לחלוטין ("Epsilon Greedy"). הפרמטר epsilon קובע את ההסתברות לביצוע פעולה כזו.
במסגרת הלימוד הורדנו באופן הדרגתי את epsilon כל פעם פי 2, וסך הכל 5 פעמים במהלך הלימוד. בחרנו בערך ההתחלתי 1/10. מאחר ומשחקי ה-Mancala אורכים מספר עשרות בודדות של תורות, ערך זה מבטיח ביצוע מרובה של פעולות אקראיות בתחילת הלימוד, אך נדיר מאוד בהמשכו.
שינוי עדין נוסף שהכנסנו באלגוריתם הלומד הוא ביטול עדכון וקטור המשקלים לאחר ביצוע פעולה אקראית, על מנת למנוע עדכון שגוי.

1.6.2.3. (gamma) Discount Factor

מאחר והמשחק הוא משחק סופי, שרשרת המצבים תמיד תבוא על סיומה, ועל כן אין צורך ב-discount factor, ובחרנו gamma = 1.

1.6.3. סוגיית התור הנוסף

במשחק Mancala קיימים מהלכים שמעניקים לשחקן תור נוסף. במהלך הלמידה ניתן לטפל בנקודה זו בשתי דרכים:

1. הגדרת פעולה של שחקן כרצף תורות, היכול לכלול תור אחד או יותר. ערכם של מצב ופעולה ייבחן על סמך מצב הלוח לאחר ביצוע רצף הפעולות כולו.
2. הגדרת פעולה של שחקן כתור בודד, וטיפול בכך שמצב המשחק לאחר הפעולה עשוי להיות כזה בו תור אותו השחקן לשחק.

דרך מספר 2 מחייבת אותנו להכניס את השחקן שתורו לשחק להגדרתו של מצב במשחק, שכן בניגוד לדרך מספר 1 ישנם מצבים בהם תורו של השחקן התחתון ומצבים בהם תורו של העליון. תוספת זו מאלצת גם לשכפל את וקטור התכונות כפי שמוסבר בסעיף 1.5.2.3.

על כן, דרך מספר 1 נראתה לנו מאוד מפתה בשל פשטותה וזוהי הדרך הראשונה בה נקטנו. בחירה זו התגלתה כבעייתית: השחקן שיוצר באופן זה היה גרוע משמעותית מהשחקנים הבסיסיים שהצענו (למעט כמובן השחקן הרנדומלי), והתכנסות וקטור המשקלים הייתה מוטלת בספק. הסיבה לכך נעוצה באופי המשחק Mancala ובתהליך באופן עדכון המשקלים באלגוריתם הלימוד: בכל איטרציה מעודכנים ערכי התכונות המתאימים למצב הלוח לאחר ביצוע הפעולה החמדנית על פי ה-reward שהתקבל והערך הנוכחי שניתן להמשך (בהתאם למצב הלוח לאחר הפעולה החמדנית בצעד הבא). בעבודה על פי דרך מספר 1, ייתכן ונתגמל (לטוב או לרע) תכונות מסוימות שהיו אמנם קיימות לפני ביצוע הפעולה הבאה, אך הן לא אלו שהובילו לטיבו של המצב הבא מאחר והפעולה הבאה עשויה לכלול מספר צעדים בסיסיים. נקודה זו אינה זניחה מאחר ומשחקי Mancala (ובעיקר משחקים ברמות גבוהות) כוללים קבלת תור נוסף פעמים רבות, ואף סדרות ארוכות של תורות נוספים.

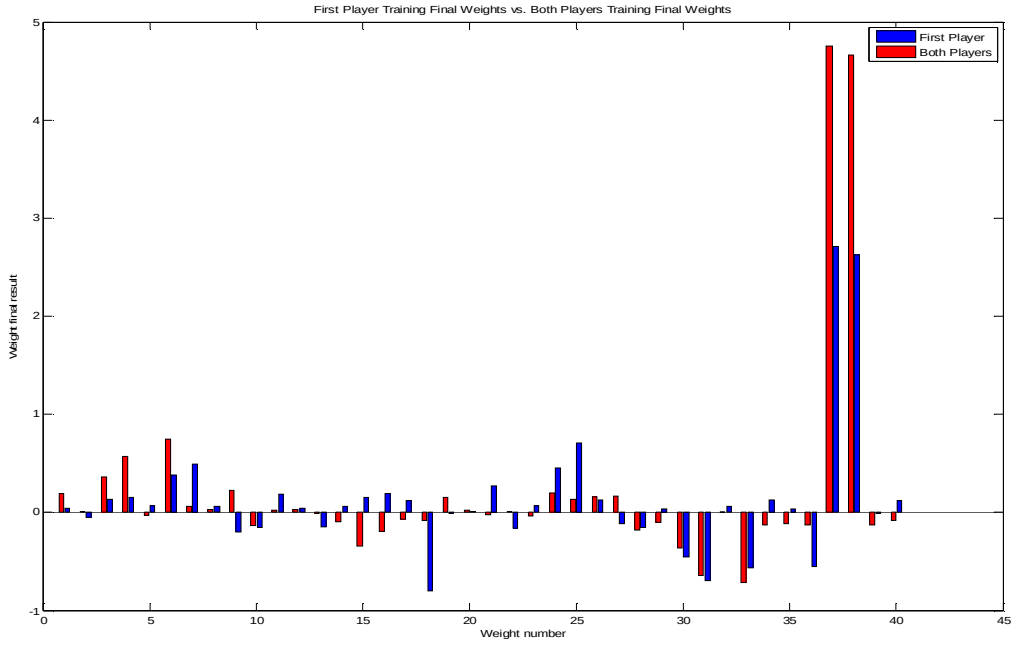
לכן, נאלצנו לבחור בדרך מספר 2, ולשכפל את וקטור התכונות.

1.6.4. למידה עבור שני השחקנים

<u>הערה:</u> נקודה זו קשורה קשר ישיר לנקודה הקודמת, החלטותינו בנושא זה והתוצאות שעל סמך קיבלנו החלטות אלו מושפעות ישירות מהגישה בה בחרנו לנקוט בסוגיית התור הנוסף.

וקטור התכונות שבחרנו מכיל סט תכונות המחושב בתורו של השחקן התחתון וסט תכונות המחושב בתורו של השחקן העליון. על מנת לכסות את מרחב המצבים באופן הומוגני היינו רוצים שדגימת המצבים בהם תורו של העליון תהיה בסדר גודל זהה לדגימת המצבים בהם תורו של התחתון. לצורך זה החלטנו לבחור אקראית את השחקן הלומד במהלך הלמידה, כך שבחלק (כמחצית) מהמשחקים יהיה השחקן הלומד השחקן התחתון ובחלק האחר יהיה זה השחקן העליון.

הבנה זו (ובעקבותיה ההחלטה הנ"ל) התקבלה בשלבים המאוחרים של העבודה על הפרויקט (עד אז הלומד היה השחקן התחתון בלבד) והיו לו השפעות חזקות על משקלי התכונות שנלמדו כפי שמשתקף בגרף הבא:



הדבר התבטא גם בשיפור ניכר בתוצאות: ב-1000 משחקים כנגד השחקן החמדן (ללא MinMax), כאשר השחקן שעובד עם המשקלים הוא השני (העליון) ניצח השחקן שנוצר ע"י למידה לשני השחקנים ב-88% ואילו זה שנוצר ע"י למידה לשחקן תחתון בלבד ניצח ב-53% בלבד.

1.6.5. צבירת מסלולים (Eligibility Traces)

באלגוריתם Q-Learning המבוסס על TD(λ) קיימת האופציה להשתמש ב- Eligibility Traces, ולעדכן בכל שלב את משקלי כלל התכונות שהובילו להגעה למצב זה בעבר, ולא רק את אלו שהיו ברגע האחרון לפני המעבר.
בספר הלימוד (Reinforcement Learning, MIT Press, 1998) מוצעות לכך שתי אופציות: accumulating traces ו-replacing traces . אנו סבורים כי במקרה של משחק Mancala בו המודל המרקובי חסר מעגלים, אין הבדל בין שתי אופציות אלו.
הפרמטר הקובע את אופן שקלול התכונות לאורך המסלול הוא lambda. Lambda=0 מבטא עבודה ללא Eligibility Traces. במסגרת העבודה על הפרויקט בחנו אופציה זו וניסינו לבצע את תהליך הלמידה בשני ערכי lambda שונים: 0 ו- 0.15. תהליך הלמידה, בשני המקרים, נראה כמתכנס, וערכי המשקלים בסיום היו שונים במעט. על מנת להכריע לגבי ערכו של lambda ערכנו סדרה של משחקים בין שני השחקנים שנוצרו, ולהלן התוצאות:

עומק עץ MinMax בשימוש	מספר משחקים	שחקן ראשון	אחוזי ניצחון של Lambda=0	אחוזי תיקו
1	1000	Lambda=0	100%	0%
1	1000	Lambda=0.15	0%	100%
2	200	Lambda=0	100%	0%
2	200	Lambda=0.15	0%	0%
3	200	Lambda=0	100%	0%
3	200	Lambda=0.15	0%	0%

הערה: על מנת להימנע ממשחק דטרמיניסטי יחיד בין השחקנים נלקחו שחקנים אקראיים מסיום תהליך הלימוד (2000 אפיזודות אחרונות). יחד עם זאת, קיבלנו תוצאות מובהקות של "הכל או כלום".

בשל יתרונו המשמעותי של השחקן הראשון, קשה להכריע מי מבין השחקנים טוב יותר. אולם, העבודה בעומק 1 (למעשה ללא MinMax) מאפשרת לראות שהשחקן שעובד ללא Eligibility Traces הוא טוב יותר (0.15 לא מצליח לנצח כשחקן ראשון ונאלץ להסתפק בתיקו). על כן בחרנו לבצע את תהליך הלימוד עם Lambda=0.
יתרה מכך, בעבודה עם Lambda=0.15 זיהינו כי השחקן הלומד נעשה פחות טוב בסיומה של הלמידה יחסית למה שהיה בתחילתה (כעבור כ-100,000 משחקים). אנו סבורים כי חוסר ההתאמה של השימוש ב-Eligibility Traces נובע מאופיו הסטוכסטי של משחק ה-Mancala, בו מצב הלוח משתנה מאוד בכל פעולה וקשה להאמין, בהינתן סט התכונות שהגדרנו, שתכונות שהיו מספר צעדים קודם הן שהובילו לרווח מאוחר יותר.

1.6.6. השחקן הלומד

בתהליך הלמידה השחקן הלומד בוחר את צעדיו החמדניים ע"פ וקטור המשקלים אותו הוא לומד. ניתן היה לשלב בבחירה זו גם עץ MinMax בעומק קטן על מנת לשפר את הבחירה. בשלבים הראשונים של העבודה על הפרויקט שיחקנו מעט עם אופציה זו, אך לא הגענו למסקנות. בסופו של דבר, משיקולי זמן חישוב החלטנו לעבוד עם שחקן לומד ללא עץ MinMax. אנו מרגישים כי נקודה זו טרם מוצתה.

1.6.7. השחקן היריב

להבנתנו, השחקן היריב בתהליך הלמידה צריך להיות בעוצמה דומה לשחקן הלומד. אחרת, השחקן הלומד לא יצליח לנצח או שתמיד ינצח, והלימוד לא יהיה אפקטיבי. על כן, בחרנו לבצע Self-Play ובכל שלב אנו לומדים כנגד השחקן שהוא תוצר השלב הקודם.
כך נוצר מצב שהרבה מהמשחקים הראשונים הם ארוכים, שכן שני השחקנים לא יודעים כיצד לשחק במשחק. ייתכן וניתן היה לקצר את הלמידה אם היינו מתחילים נגד אחד השחקנים הבסיסיים (למשל השחקן החמדן בעומק 2), אך מאחר ולא הצבנו את משך הלמידה כמטרה עיקרית, לא בדקנו לעומקה את אופציה זו.

1.6.8. משך הלמידה

מהסתכלות על התכנסות המשקלים לאורך הלמידה בחרנו להריץ את הלמידה על 500,000 משחקים, על מנת להבטיח התכנסות מלאה. במחשב הביתי (AMD 2.21 GHz) משך הלמידה כולה הוא כ-12 שעות.
בסעיף 1.7.2 הדן בהתכנסות וקטור המשקלים עולה כי ניתן היה אולי להסתפק בפחות משחקים (אולי אפילו 300,000), אך בשל בעיות בהתכנסות של מספר משקלים, בחרנו במספר המשחקים הגבוה יחסית.

1.6.8.1. שיפור ההחלטה על סוף המשחק

עקרונית, משחק ה-Mancala מסתיים כאשר לאחד השחקנים, בתורו, אין כלל אופציה לשחק (כל הגומות בצידו ריקות). המנצח הוא השחקן שבמיכלו יותר אבני משחק ברגע זה. אולם, ניתן בקלות להבין שברגע שאחד השחקנים צבר יותר ממחצית אבני משחק (24), הוא יהיה המנצח במשחק, ולסיים את המשחק בנקודה זו.
שיפור זה הביא לקיצור משמעותי בזמן הריצה של תהליך הלמידה.

1.7. תוצאות הלמידה

1.7.1. וקטור המשקלים הסופי

בסיום תהליך הלמידה הסופי (שפרטיו המלאים בסעיף הקודם) נלמד וקטור המשקלים הבא:

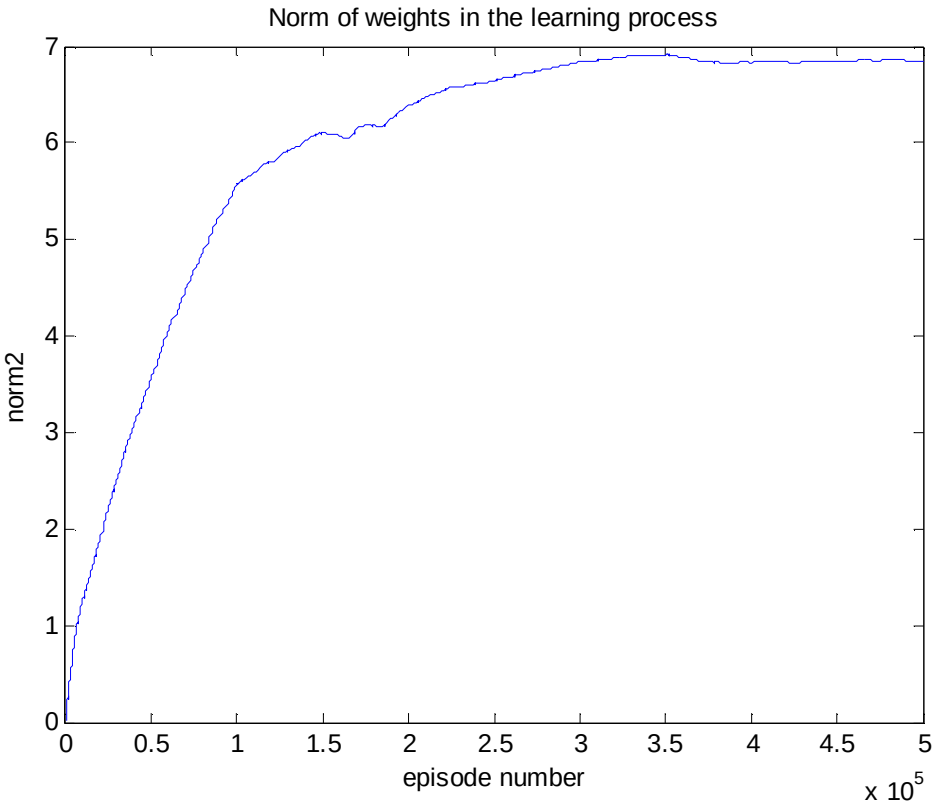
(player == 0)	board	can_get_another_move	0.1904
		is_passing_store	0.0104
		opposite_empty	0.3652
		Captures	0.5651
		can_fill_all_empties	-0.0318
		max_capture_amount	0.7498
		play_options	0.0629
		(board(6)==0)	0.0236
		can_win_in_1_turn	0.2253
	ReverseBoard	can_get_another_move	-0.1395
		is_passing_store	0.0193
		opposite_empty	0.0271
		Captures	-0.0141
		can_fill_all_empties	-0.0922
		max_capture_amount	-0.3414
		play_options	-0.1935
		(board(6)==0)	-0.0722
		can_win_in_1_turn	-0.0759
(player == 1)	board	can_get_another_move	0.1478
		is_passing_store	0.0193
		opposite_empty	-0.0241
		Captures	0.0146
		can_fill_all_empties	-0.0372
		max_capture_amount	0.1954
		play_options	0.1331
		(board(6)==0)	0.1576
		can_win_in_1_turn	0.1656
	ReverseBoard	can_get_another_move	-0.1738
		is_passing_store	-0.101
		opposite_empty	-0.367
		Captures	-0.6467
		can_fill_all_empties	-0.0005
		max_capture_amount	-0.7202
		play_options	-0.1272
		(board(6)==0)	-0.1097
		can_win_in_1_turn	-0.125
(player == 0)	delta_store		4.7509
(player == 1)	delta_store		4.6565
Player			-0.1255
Const 1			-0.0792

באופן כללי, התוצאות אליהן התכנס תהליך הלמידה מתאימות לאינטואיציה שלנו לגבי המשחק:

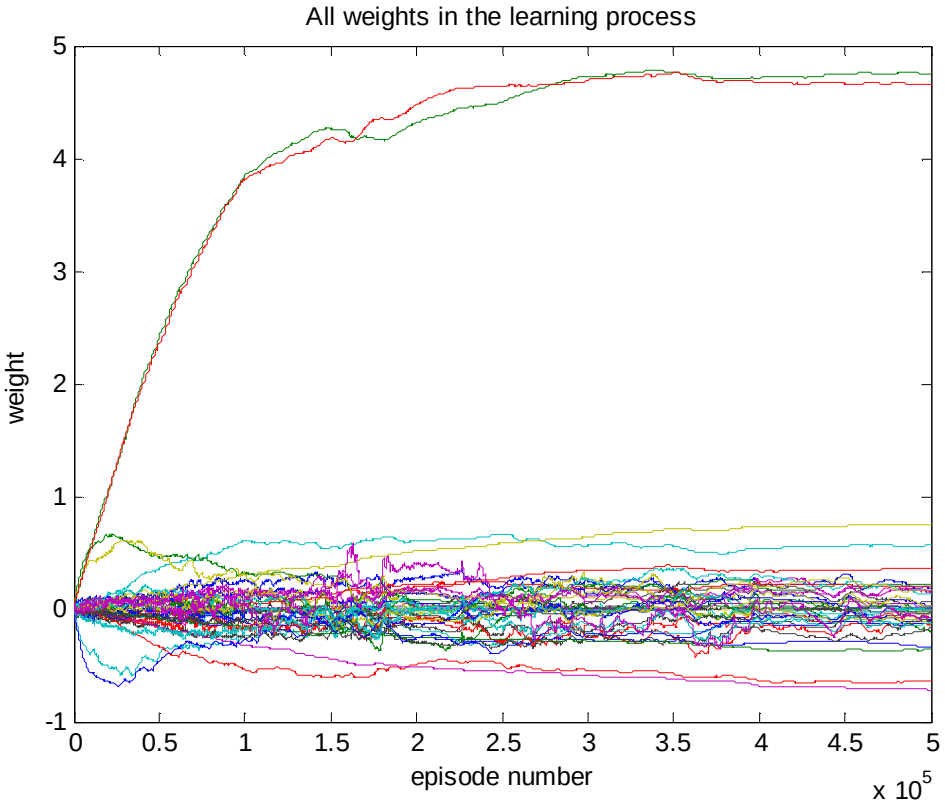
- שתי התכונות בעלות המשקלים הגבוהים ביותר הן התכונות המתארות את ההפרש הנוכחי (delta store) כאשר תור התחתון ותור העליון. משקלן הוא חיובי, כלומר יתרון לתחתון עשוי להוביל ל-reward חיובי, וכן כאשר התחתון מנצח ה-reward הוא 1.
- מרבית התכונות המתארות את הפוטנציאל של התחתון הן בערך חיובי, והתכונות המקבילות המתארות את הפוטנציאל של העליון הן בערך שלילי.
- משקלה של התכונה player (תורו של השחקן העליון/התחתון) הוא שלילי, כלומר שני השחקנים מעדיפים מצבי משחק בהם תורם לשחק.
- תכונות המתארות פוטנציאל או סכנה מיידית (כאשר התור מתאים לצד של הלוח) בעלות ערכים גדולים יותר מאלו המתארות פוטנציאל או סכנה עתידית (כאשר התור אינו מתאים לצד של הלוח).
- יש חשיבות רבה לתכונות הקשורות ב-"capture", שכן זוהי הדרך המהירה ביותר לזכות ביתרון.
- ניתן לזהות סימטריה (ברמה מסוימת) בין משקלי תכונות הלוח של השחקן התחתון בתורו ומשקלי תכונות הלוח של השחקן העליון בתורו (המספרים הללו נגדיים). אינטואיטיבית, סימטריה זו צפויה במשחק Mancala.

1.7.2. התכנסות וקטור המשקלות לאורך הלמידה

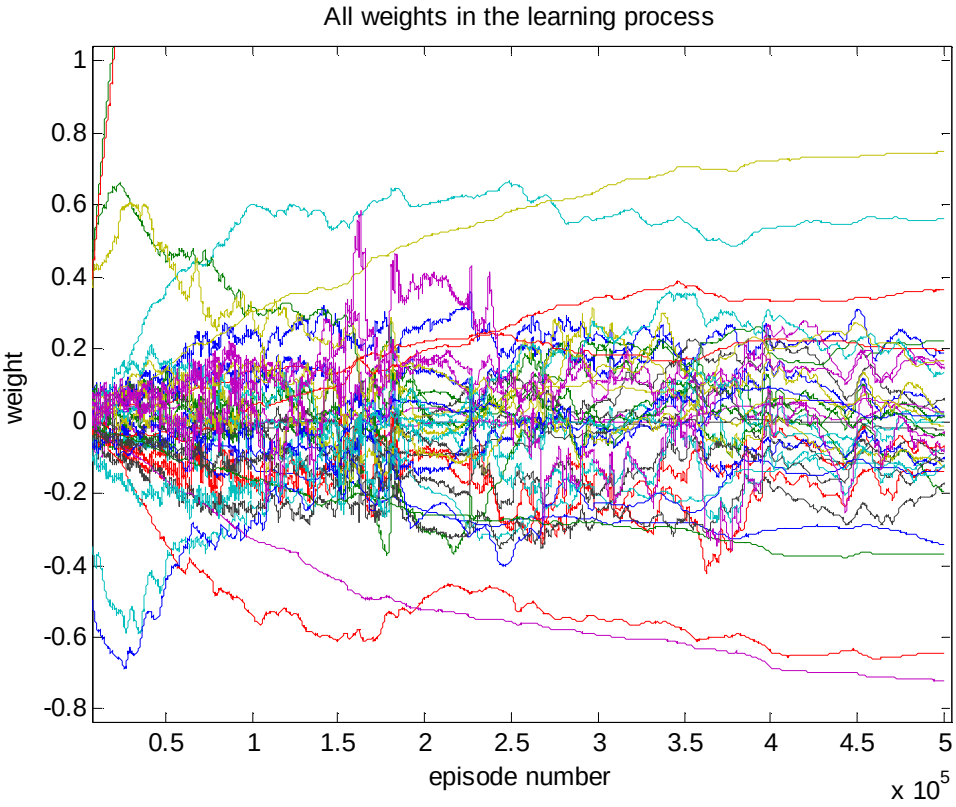
גרף זה מתאר את התכנסות הנורמה (שורש סכום הריבועים) של וקטור המשקלים ב-500,000 משחקי הלימוד:



גרף זה מתאר את התכנסות כל אחד מהמשקלים ב-500,000 משחקי הלימוד:



כאשר מתמקדים על המשקלים בעלי הערכים הקטנים:



באופן עקרוני מזוהה התכנסות המשקלים לערכים סופיים. הדבר ניכר כאשר מסתכלים על הנורמה וכן בגרף המשקלים עצמם. יחד עם זאת, ישנם מספר משקלים שעד סיום התהליך "מתנדנדים" ולא מזוהה התייצבותם בשלב כלשהו.

רשימת התכונות שלהלן כוללת את התכונות העיקריות שהמשקלים עבורן "אינם מתכנסים". אלו המשקלים שמשרעת התנודות שלהם ב-50,000 האפיזודות האחרונות הייתה גדולה מ-0.1. בהקשר זה יש לציין שלרשימה זו ערך מוגבל בלבד, מאחר וערך המשקל אינו מדד טוב להשפעת התכונה (ערכי התכונות אינם מנורמלים כהלכה), אך ייתכן ויש בה ערך:

(player == 0)	board	can_get_another_move
		is_passing_store
		can_fill_all_empties
		play_options
		(board(6)==0)
	ReverseBoard	play_options
		(board(6)==0)
(player == 1)	board	is_passing_store
		play_options
		can_win_in_1_turn
	ReverseBoard	can_get_another_move
		(board(6)==0)
Const 1		

עד כה, לא הצלחנו להבין מדוע ההתכנסות אינה מלאה.

1.7.3. שיפור השחקן לאורך הלמידה

על מנת לבחון את שיפור השחקן לאורך הלמידה בחנו את יכולות השחקן (ללא שילוב MinMax) כנגד השחקן החמדן (גם הוא בעומק 1) במספר נקודות במהלך הלמידה. בכל תא בטבלה זוג המציין את אחוזי הניצחונות ותוצאות התיקו ב-1000 משחקים.

	כשחקן ראשון	כשחקן שני
כעבור כ-10,000 משחקים	85% / 3%	60% / 4%
כעבור כ-100,000 משחקים	91% / 2%	84% / 2%
כעבור כ-200,000 משחקים	87% / 4%	82% / 3%
כעבור כ-400,000 משחקים	90% / 2%	89% / 3%
כעבור כ-500,000 משחקים	91% / 2%	89% / 2%

נבחרו שחקנים אקראית מתהליך הלמידה בטווח של 1000 משחקים סביב המספר המצוין.

ניתן לראות כי השחקן הולך ומשתפר לאורך הלמידה, דבר שמתבטא בעיקר בתפקודו כשחקן שני, בשל הקלות לנצח את השחקן החמדן בעומק 1 כאשר משחקים ראשונים. הופתענו לגלות כי אף לאחר 10,000 משחקים בלבד (רחוק מאוד מסיום ההתכנסות) מתקבל שחקן שהוא טוב מהשחקן החמדן.

בנוסף לכך שיחקנו גם בין השחקנים הללו עצמן והתוצאות מרוכזות בטבלה הבאה:

	~10,000	**~100,000**	**~200,000**	**~400,000**	**~500,000**	**average**
~10,000		44% / 5%	93% / 0%	0% / 0%	24% / 0%	40% / 1%
~100,000	87% / 0%		100% / 0%	60% / 40%	77% / 12%	81% / 13%
~200,000	93% / 0%	98% / 0%		0% / 0%	0% / 0%	48% / 0%
~400,000	84% / 1%	75% / 0%	100% / 0%		0% / 0%	65% / 0%
~500,000	100% / 0%	100% / 0%	100% / 0%	100% / 0%		100% / 0%
average	91% / 0%	79% / 1%	98% / 0%	40% / 10%	25% / 3%	

(בכל תא בטבלה זוג (אחוזי ניצחונות, ואחוזי תוצאות תיקו) המתאים לתוצאות 1000 משחקים כאשר השחקן שבכותרת השורה המתאימה הוא הראשון והשחקן שבכותרת העמודה המתאימה הוא השחקן השני)

ניכר כי השחקן האחרון טוב כשחקן ראשון מיתר השחקנים, שכן הוא ניצח את האחרים ב-100%. יחד עם זאת איננו מזהים עליה רצופה שממוצעים שמימין, ונראה כי השחקן שמתקבל כעבור 100,000 משחקים (לפני סיום ההתכנסות!) טוב מהשחקנים שמתקבלים כעבור 100,000 ו-300,000 משחקים נוספים.
כאשר בוחנים את שורת הממוצעים התחתונה מזהים שוב שהשחקן בסיום הלמידה כולה טוב מהאחרים ולו יש את אחוזי ההפסדים (הניצחונות של האחרים אותו) הנמוכים ביותר. שוב, איננו מזהים ירידה רצופה והשחקן שמתקבל כעבור 100,000 משחקים (לפני סיום ההתכנסות!) טוב מהשחקנים שמתקבלים כעבור 100,000 משחקים נוספים.

על מנת להסביר את היחלשות המסוימת שזיהינו של השחקן הלומד לאחר 100,000 משחקים, הן כשחקן ראשון והן כשחקן שני, והתחזקותו מחדש בשלבי הלמידה האחרונים, עלינו להניח כי במהלך ההתקדמות לאופטימום עובר השחקן בנקודות קיצון מקומיות, שהן טובות כמעט כמו הקיצון הגלובלי, אך הוא ממשיך לעבר הקיצון הגלובלי. התקשינו לבחון זאת במדויק, ונאלצנו להסתפק בהסבר אינטואיטיבי זה.

1.8. השחקן ThetaGreedy - מדיניות בהתאם לתוצאות הלמידה

עקרונית, לאחר למידת וקטור משקלים מתאים, עלינו לשחק באופן חמדני לגבי ערך המצב המחושב באמצעות וקטור משקלים זה. במהלך העבודה על הפרויקט נוכחנו לדעת כי השחקן שנוצר מתקשה לעיתים לסיים את המשחק, וגם כאשר יכול לנצח מעדיף להמשיך במשחק. על מנת לשפר את השחקן שינינו במעט את חישוב הערך: ערכו של מצב מנצח נקבע ל-99999, וערכו של מצב מספיד ל-99999-. ערכו של תיקו נקבע ל-0. לכן, כאשר נשחק במדיניות חמדנית לגבי ערך זה מהלך המביא לניצחון בהכרח ייבחר ובמקביל מהלך המביא להפסד לא ייבחר (אלא כשאין ברירה...).

<u>הערה:</u> אנו חושדים כי קביעת ערכו של התיקו ל-0 עשויה להיות מעט בעייתית מאחר וייתכן כי פונקצית הערך שיצרנו מחמירה או מקלה בהערכת טיבם של מצבים, וקביעת ערך התיקו כ-0 עשויה להתוות מדיניות שגויה. בחרנו בערך 0 מאחר והתקשינו לבחור ערך מתאים אחר.

פונקצית חישוב ערך זה שולבה במנגנון MinMax (זהה לזה ששימש ליצירת השחקן הבסיסי) בעומקים 1-5 וכך יצרנו 5 שחקני ThetaGreedy שהם תוצרו הסופי של תהליך הלמידה. כמו בשחקן החמדן הבסיסי, הגבלת העומק לעד 5 מהלכים קדימה נובעת משיקולי זמן חישוב.

לאחר מספר משחקים כנגד ThetaGreedy בעומקים 4 ו-5, שמנו לב כי גם לאחר התיקון של ערכי המצבים המנצחים והמפסידים קיימים מצבים בהם המחשב דוחה את הניצחון או מקדים את ההפסד. הסיבה לכך היא היותן של שתי אופציות שקולות מבחינת הערך שניתן להם, ועל מנת לפתור זאת הוספנו שינוי קטן באלגוריתם ה-MinMax הגורם לשחקן להעדיף ניצחונות מוקדמים על פני מאוחרים, והפסדים מאוחרים על פני מוקדמים. שינוי זה הביא לשיפור ביכולת השחקן שנוצר, שנצפה כנגד שחקן אנושי.

בנוסף, על מנת להימנע ממשחקים דטרמיניסטיים לחלוטין, בחרנו לעבוד עם 2000 וקטורי המשקלים האחרונים (ולא עם האחרון בלבד), כאשר לפני כל משחק מבוצעת הגרלה ונבחר וקטור יחיד מביניהם.

1.9. הערכת טיב השחקן שיוצר

1.9.1. נגד השחקנים הבסיסיים

טבלאות אלו מרכזות את תוצאות השחקן בעומקים השונים כנגד השחקנים הבסיסיים וכנגד עצמם

		Heuristic	Greedy				
			1	2	3	4	5
ThetaGreedy	1	100% / 0.0%	91% / 3.0%	69% / 3.5%	36% / 6.0%	45% / 5.0%	37% / 0.0%
	2	100% / 0.0%	100% / 0.0%	100% / 0.0%	100% / 0.0%	100% / 0.0%	100% / 0.0%
	3	100% / 0.0%	100% / 0.5%	100% / 0.0%	100% / 0.5%	99% / 0.0%	94% / 1.5%
	4	100% / 0.0%	100% / 0.0%	100% / 0.0%	94% / 2.5%	89% / 0.5%	86% / 6.0%
	5	100% / 0.0%	100% / 0.0%	100% / 0.0%	100% / 0.0%	100% / 0.0%	100% / 0.0%

		ThetaGreedy				
		1	2	3	4	5
Heuristic		100% / 0.0%	100% / 0.0%	0% / 0.0%	100% / 0.0%	100% / 0.0%
Greedy	1	9% / 1.5%	7% / 0.5%	5% / 0.0%	3% / 1.0%	2% / 0.0%
	2	76% / 0.0%	57% / 3.5%	61% / 2.5%	39% / 2.5%	37% / 8.5%
	3	92% / 2.5%	65% / 4.0%	46% / 0.5%	35% / 1.0%	71% / 7.0%
	4	100% / 0.5%	40% / 1.5%	72% / 2.0%	5% / 0.5%	3% / 3.0%
	5	99% / 0.0%	84% / 3.5%	90% / 3.0%	76% / 2.5%	72% / 2.0%
ThetaGreedy	1	42% / 58.0%	100% / 0.0%	0% / 0.0%	0% / 0.0%	0% / 0.0%
	2	100% / 0.0%	100% / 0.0%	100% / 0.0%	100% / 0.0%	100% / 0.0%
	3	100% / 0.0%	100% / 0.0%	100% / 0.0%	100% / 0.0%	82% / 0.0%
	4	100% / 0.0%	100% / 0.0%	0% / 0.0%	100% / 0.0%	0% / 0.0%
	5	100% / 0.0%	100% / 0.0%	100% / 0.0%	100% / 0.0%	100% / 0.0%

(בכל תא בטבלה זוג (אחוזי ניצחונות, ואחוזי תוצאות תיקו) המתאים לתוצאות 1000 משחקים כאשר השחקן שבכותרת השורה המתאימה הוא הראשון והשחקן שבכותרת העמודה המתאימה הוא השחקן השני)

ניתן לראות כי כשחקן ראשון ThetaGreedy בעומק 5 (או בעומק 2) אינו מנוצח ע"י השחקנים הבסיסיים.

כשחקן שני, רק ThetaGreedy בעומק 3 הצליח לפרוץ את חומות השחקן ההיורסיטי. ThetaGreedy בעומק 5 מפסיד לשחקן החמדן בעומק 5 ב-72% מהמשחקים.

נגד משתמש אנושי

שיחקנו מספר משחקים כנגד השחקן שיוצר בעומק 5:

- כשחקן ראשון לא קל לנצח.
- כשחקן שני קשה מאוד לנצח.
- במהלך המשחקים ניתן לראות כי אסטרטגיות מוכרות כמו הרעבה (מניעת אופציות למשחק עבור היריב) ופיטום (stuffing, מניעת גומות ריקות אצל היריב וכך מניעת יכולת כיבוש) מבוצעות לסירוגין ע"י השחקן שיוצר.

1.9.2. נגד שחקני אינטרנט זמינים

נעזרנו באתר http://www.rocketsnail.com/mancala/game.htm המציע משחקי Mancala בשלוש רמות שונות. בחנו את השחקן שיוצר בעומק 5 בשני משחקים כנגד הרמה הגבוהה ביותר באתר:

- ThetaGreedy בעומק 5 ניצח בתור שחקן ראשון בהפרש גדול.
- ThetaGreedy בעומק 5 הפסיד בתור שחקן שני בהפרש בינוני.

1.10. נקודות פתוחות

1.10.1. חוסר התכנסות

מספר משקלים של תכונות מסוימות אינם מתכנסים גם לאחר 500,000 משחקים. ייתכן ולתכונות אלו אין השפעה אמיתית על המשחק, אך במקרה זה היו משקלים אלו צריכים להתכנס לערך בסביבות 0.

1.11. הצעות לשיפור עתידי

1.11.1. שיפור השחקן הלומד ועבודה עם עץ MinMax בלמידה

כאמור בסעיף 1.6.6, אופציה זו לא נבדקה מספיק לעומק.

1.11.2. השחקן היריב בלמידה

כאמור בסעיף 1.6.7, אופציה זו לא נבדקה מספיק לעומק. בנוסף, ייתכן ושינוי זה יביא לשיפור התוצר הסופי, מעבר לקיצור זמן הלמידה.

1.11.3. הוספת התכונות שויתרנו עליהן

במידה וזמן החישוב לא היה מהווה מגבלה כל כך משמעותית היינו מציעים להחזיר את התכונות עליהן החלטנו לוותר (ראה סעיף 1.5.2.4). בפרט תכונה שתחשב את מספר התורות הנוספים המקסימלי שניתן לקבל עשויה להיות מועילה מאוד, וכך נוכל אולי לתפוס את כוחו של השחקן ההיוריסטי. לצורך זה מומלץ אולי לעבור לשפת תכנות מהירה יותר, ולוותר על הנוחות שבעבודה ב-Matlab.

1.11.4. למידת וקטורי משקלים למצבי משחק שונים

כשחקנים אנושיים שמנו לב שאנו לעיתים מחליפים בין אסטרטגיות משחק, בהתאם למצבנו הנוכחי. כך למשל, כאשר אנו ביתרון לא נרצה לקחת סיכונים, אלא לשמר את היתרון, סיכונים שהיינו עשויים לקחת במידה ואנו מפסידים. כדי לתרגם החלפות אלו ללמידת וקטור משקלים ניתן לבחון את האופציה ללמוד 3 וקטורי משקלים: לתחילת המשחק, למצב של יתרון גדול מ-4, ולמצב של יתרון ליריב הגדול מ-4 (וקטור תיקו, הגנה והתקפה). במידה ורוצים לשמור על לימוד של וקטור אחד, ניתן לעשות זאת ע"י שכפול התכונות ואיפוס התכונות שאינן שייכות למצב (כפי שעשינו עבור השחקן העליון והתחתון). אנו מאמינים כי שיפור כזה עשוי לייצר שחקן חזק יותר.

1.12. רקע עיקרי

במהלך העבודה נעזרנו בעיקר במקורות הבאים:

A.G. Barto and R.S., Reinforcement Learning, MIT Press, 1998.

Fritz Dooley's Mancala Center - http://fritzdooley.com/mancala/index.html

2. חלק שני- High Level Code Documentation

2.1. הוראות הפעלה

על מנת להפעיל את המשחק ולשחק כנגד השחקן שיצרנו, יש להעתיק את הספרייה כולה, להיכנס לתוכנת Matlab ובתוכה לספרייה שהועתקה ולהריץ את הקובץ **RunMancala.m**.

קובץ זה טוען את הקובץ T.mat בו שמור וקטור המשקלים.

הכפתור T-MOVE מבצע מהלך של ThetaGreedy בעומק 5.

2.2. תיאור כללי של הקבצים בפרויקט

```
Are2PoliciesEqual       - Helpful function to decide whether 2
                          policies are identical.
CalculateFeatures       - Calculates all features values for a
                          given state.
CompareBasicPolicies    - Play the basic policies against
                          themselves.
CompareSomePolicies     - Template: Play some policies against
                          themselves.
CreateBoard             - Creates a board for a new game
DisplayBoard            - Displays the board to the matlab shell.
GameOver                - Checks whether the game is over.
GetDeltaScore           - The value function of the "Greedy
                          Player."
GetScore                - Checks for the reward of the state.
GetTGreedyScore         - The value function of the "Theta Greedy
                          Player."
GetTLearningScore       - The value function of the "Theta
                          Learning Player."
HeuristicMove           - Policy function of the "Heuristic
                          Player."
MancalaGUI              - Activates the GUI
minimax                 - Implementation of the MinMax tree.
MinMaxMove              - Policy function of the "Greedy Player."
Move                    - Moves snails on the board.
OptionalMoves           - Checks for all possible moves.
Play                    - Plays a full turn for a given policy.
Play1Turn               - Plays 1 turn for a given policy.
Play2Policies           - Plays two policies against each other.
PlayMancalaGUI          - Activates the GUI
RandomBoard             - Creates a random board for testing only.
RandomMove              - Policy function of the "Random Player."
ReverseBoard            - Reverses a board
RunEpisode              - The heart of the learning process.
RunEpisodes             - Runs the whole learning process.
TGreedyMove             - Policy function of the "Theta Greedy
                          Player."
TLearningMove           - Policy function of the "Theta Learning
                          Player."
CompareLearningPolicies - Play the learning policies against
                          themselves and MinMax1.
```

2.3. מבני נתונים עיקריים

- וקטור התכונות הוא וקטור בגודל 40, של double.
- וקטור המשקלים הוא וקטור בגודל 40, של double.
- לוח המשחק הוא וקטור בגודל 14 של מספרים 48-0, כאשר התא ה-I מייצג את הגומה ה-i.
- שחקן הוא משנה בינארי: 0= שחקן תחתון, 1=שחקן עליון.

2.4. תיאור מפורט של הקבצים והפונקציות בפרויקט

Are2PoliciesEqual
Helpful function to decide whether 2 policies are identical.
Receives the number of tests, two policies and 2 argument each
and checks their results on random boards,
Prints the differences between the two policies.

CalculateFeatures
Calculates all features values for a given state.
Receives the board and the player.
Returns the vector of the calculated features.

CalculateFeatures>Calculate1SideFeatures
CALCULATEFEATURES Calculates the features of the lower side of the board.
Receives the board.
Returns the vector of the calculated 9 features
of the lower side of the board.

CompareBasicPolicies
Play the basic policies against themselves.
Play 100 games between every two basic policies.
Returns the policies, their arguments, and
the results (wins and ties)

CompareLearningPolicies
Play the learning policies against themselves and MinMax1.
Play 1000 games between different policies during the learning process.
Play also against MinMax1 player.
Receives the matrix of all learnt policies.
Returns the policies, their arguments, and
the results (wins and ties)

CompareSomePolicies
Template: Play some policies against themselves.
This is a template to be edited before any use,
used to create a matrix of results comparing between some policies.
Receives the matrix of all learnt policies.
Returns the policies, their arguments, and
the results (wins and ties)

CreateBoard
Creates a board for a new game
Returns a board of a new game.

DisplayBoard
Displays the board to the matlab shell.
Receives a board vector and display it to the shell.
The "lower player" is in the low line...

GameOver
Checks whether the game is over.
Receives the board and the current player.
Returns true iff the game is over.

GetDeltaScore
The value function of the "Greedy Player".
Receives the board.
Also receives player and one more argument, but doesn't use it.
(There are needed for the uniformity of the GetxxxxScore functions).
Returns the difference between the stores.
If the game is over returns +- 99999.

GetScore
Checks for the reward of the state.
Receives the board.
Should be called in the learning process iff the game is over.
Returns 1 if the lower player won, -1 if the upper player won,
and 0 in the case of a tie.

GetTGreedyScore
The value function of the "Theta Greedy Player".
Receives the board, the player and the theta vector.
Returns the calculated value of the state.
If the gmae is over returns +- 99999.

GetTLearningScore
The value function of the "Theta Learning Player".
Receives the board, the player and the theta vector.
Returns the calculated value of the state.
Used only in the learning process.
Similar to GetTGReedyScore except the game-over case.

HeuristicMove
Policy function of the "Heuristic Player".
Receives a board and the player.
Also receives 2 spare arguments.
(all policy functions have 2 arguments)
Returns the position of the hole recommended to play.
Shouldn't be called if the game is over and there are no possible moves.
The logic is described in the document.

HeuristicMove>MaxDeltaSnails
Finds the move which will give the highest delta snails.
Receives a board.
Returns the maximum delta snails possible for player 0
and the move which will give it.
Recursive function, in the case of extra turn.

HeuristicMove>MaxExtraTurns
Finds the move which will give the highest extra turns options
Receives a board.
Returns the maximum extra turns possible for player 0
and the move which will give it.
Recursive function, in the case of extra turn.

HeuristicMove>GivesExtraTurn
Checks whether an hole can give an extra turn
Receives a board and a position from 1 to 6.
Returns true iff moving from that hole is possible and
it gives an extra turn.
Notice that in case of 0 0 0 0 0 1, there will be no another turn,
In this case this function is wrong, in order to be simple.

MancalaGUI
Activates the GUI
Receives a gui action: 'start', 'initialize', 'move', 'move_T',
and acts accordingly.
Recursive function, object handlers call itself with different action.

MancalaGUI>SetSquares
Updates squares' styles
Updates the square styles according to the turn and the board.
A square will be a push-button iff it is of the player
who is going to play and not empty.
Also updates the strings shown on the squares according to the board,
and checks for game-over.

MinMaxMove
Policy function of the "Greedy Player
Receives a board, the player, the required MinMax depth.
Also receives an argument which is passed as is to the score function.
(all policy functions have 2 arguments)
Returns the position of the hole recommended to play.
Shouldn't be called if the game is over and there are no possible moves.

Move
Moves snails on the board.
Receives a board and a position.
Returns the board after the move from the position, and extra_turn flag.
position should be a legal hole: 1-6 or 8-14 and not empty!

Move>Move_player0
Moves snails of the lower player on the board.
Receives a board and a position.
Returns the board after the move from the position, and extra_turn flag.
position should be a legal hole on the lower side: 1-6 and not empty!

OptionalMoves
Checks for all possible moves.
Receives the board and the current player.
Returns a vector of all possible moves from this position.

Play
Plays a full turn for a given policy.
Receives a board, the player, a pointer to a policy function and its 2 arguments.
Returns the board and the player after the full turn that was chosen
by the policy function.
(a full turn is a series of moves, each giving an extra turn).
This function stops in the case of game over.

Play1Turn
Plays 1 turn for a given policy.
Receives a board, the player, a pointer to a policy function and its 2 arguments.
Returns the board and the player after move that was chosen by the policy function.
Also returns the value of the board after the move, that was calculated
by the policy function. This is in order to save some time in the learning.
Shouldn't be called if the game is over and there are no possible moves.

Play2Policies
Plays two policies against each other.
Receives the required number of games,
2 policies (functions pointers) and 2 arguments for each policy,
and also a flag to indicate who will play first.
Returns the winning and ties rate of first player.
Also puts a message to the screen of the current results.

PlayMancalaGUI
Activates the GUI
Receives a theta vector and a MinMax depth,
and activates the GUI possibly against "Theta Greedy Player'.
Calls the main GUI function: MancalaGUI

RandomBoard
Creates a random board for testing only.
Returns a random board.
For testing only.

RandomMove
Policy function of the "Random Player".
Receives a board and the player.
Also receives 2 spare arguments.
(all policy functions have 2 arguments)
Returns the position of the hole recommended to play, which is a random position.
Also returns a temp var which is 0.
(needed in order to make Play1Turn generic).
Shouldn't be called if the game is over and there are no possible moves.

ReverseBoard
Reverses a board
Receives a board.
Returns the same board reversed.

RunEpisode
The heart of the learning process.
Runs an episode of the learning process, i.e a single game.
Receives: the current theta vector (to be improved) T,
the depth of the MinMax tree used to choose the learning player
actions, the learning rate alpha, the Eligibility Traces
Factor lambda, the discount factor gamma, the random rate epsilon, the
opponent policy during the learning and its two arguments.
Returns the improved theta vector.

RunEpisodes
Runs the whole learning process.
Runs the whole learning process according to our final decided parameters.
Returns the learnt theta vector, and a matrix which contains all
of the theta vectors during the learning.

TGreedyMove
Policy function of the "Theta Greedy Player".
Receives a board, the player, the theta vector and the required MinMax depth.
Returns the position of the hole recommended to play.
Also returns the value that was calculated by GetTGreedyScore of the
next state (after playing from that position).
Shouldn't be called if the game is over and there are no possible moves.

TLearningMove
Policy function of the "Theta Learning Player".
Receives a board, the player, the theta vector and the required MinMax depth.
Returns the position of the hole recommended to play.
Also returns the value that was calculated by GetTLearningScore of the
next state (after playing from that position).
Shouldn't be called if the game is over and there are no possible moves.

minimax
Implementation of the MinMax tree.
Receives the board. a player, the required depth, a pointer to the score function and its argument.
Should be called iff the game isn't over.
Depth should be >= 1.
Returns the recommended move and its value according to the input score function.

minimax>alphabeta
Inner implementation of the MinMax tree with alpha beta pruning.
Receives the board. a player, the required depth, a pointer to the score function and its argument,
and the current alpha and beta.
Returns the recommended move and its value according to the input score function.

minimax>alpha_or_beta
Helpful function for the inner implementation of the MinMax tree with alpha beta pruning.
Receives alpha, beta and player.
Returns alpha if player == 0 and beta if player == 1.

2.5. תיאור הקשרים בין הפונקציות

Dependency Report

Run Report on Current Directory

☑ Show child functions ☐ Show parent functions (current dir. only)
☑ Show subfunctions

Built-in functions and files in toolbox/matlab are not shown

Report for Directory C:\Program Files\MATLAB\R2006b\work\Copy_of_Mancala

M-files	Children (called functions)
Are2PoliciesEqual	current dir : RandomBoard current dir : GameOver current dir : DisplayBoard
CalculateFeatures	current dir : ReverseBoard subfunction : CalculateISideFeatures
CalculateFeatures>CalculateISideFeatures	
CompareBasicPolicies	current dir : RandomMove current dir : HeuristicMove current dir : MinMaxMove current dir : Play2Policies
CompareLearningPolicies	current dir : TGreedyMove current dir : MinMaxMove current dir : Play2Policies
CompareSomePolicies	current dir : MinMaxMove current dir : RandomMove current dir : TGreedyMove current dir : Play2Policies
Contents	
CreateBoard	
DisplayBoard	
GameOver	current dir : OptionalMoves
GetDeltaScore	current dir : GameOver current dir : GetScore
GetScore	
GetTGreedyScore	current dir : GameOver current dir : GetScore current dir : CalculateFeatures
GetTLearningScore	current dir : CalculateFeatures
HeuristicMove	current dir : ReverseBoard subfunction : MaxExtraTurns subfunction : MaxDeltaSnails
HeuristicMove>MaxDeltaSnails	current dir : Move subfunction : MaxDeltaSnails
HeuristicMove>MaxExtraTurns	current dir : Move subfunction : GivesExtraTurn subfunction : MaxExtraTurns
HeuristicMove>GivesExtraTurn	
MancalaGUI	current dir : CreateBoard current dir : Move current dir : TGreedyMove recursion : MancalaGUI subfunction : SetSquares
MancalaGUI>SetSquares	current dir : GameOver
MinMaxMove	current dir : minimax current dir : GetDeltaScore

Move	current dir : ReverseBoard subfunction : Move_player0
Move>Move_player0	
OptionalMoves	
Play	current dir : GameOver current dir : Play1Turn
Play1Turn	current dir : Move
Play2Policies	current dir : CreateBoard current dir : GameOver current dir : Move
PlayMancalaGUI	current dir : MancalaGUI
RandomBoard	
RandomMove	current dir : OptionalMoves
ReverseBoard	
RunEpisode	current dir : CreateBoard current dir : Play1Turn current dir : TLearningMove current dir : GetTLearningScore current dir : CalculateFeatures current dir : GameOver current dir : Play current dir : GetScore current dir : RandomMove
RunEpisodes	current dir : TGreedyMove current dir : CalculateFeatures current dir : CreateBoard current dir : RunEpisode
TGreedyMove	current dir : minimax current dir : GetTGreedyScore
TLearningMove	current dir : minimax current dir : GetTLearningScore
minimax	subfunction : alphabeta
minimax>alphabeta	current dir : GameOver current dir : OptionalMoves current dir : Move subfunction : alphabeta subfunction : alpha_or_beta
minimax>alpha_or_beta	