שם: עמית סידס. תעודת זהות: 206768780. קורס: בינה מלאכותית 20551

השתמשתי ב- Python 2.7.16.

שאלה 1

השאלה הכי כיפית בממ"ן.

בהתחלה חשבתי שצריך לממש סוכן שמנצח את mediumClassic, אז השקעתי די הרבה ועשיתי סוכן די טוב.

הפונקציה מחשבת את המצב עבור הפעולה שהתקבלה ומחזירה את הערכה שלה עבור המצב לאחר ביצעו הפעולה. כשהערך גדול יותר, הפעולה טובה יותר (בדומה לפונקציית fitness).

הדבר הראשון שהסוכן בודק הוא האם הוא קרוב מידי לרוח כלשהי לאחר ביצוע הפעולה. אם המרחק הבטוח (safeDistance המוגדר כ-3) גדול יותר מהמרחק מאחת הרוחות, הוא מוסיף ערך שלילי די גדול (פקטור 50-) לסכום הערכת המצב וכך המהלך הזה כנראה לא ייבחר.

לאחר מכן, הוא בודק האם הצלחנו לאכול רוח בעזרת הפעולה שהתקבלה. אם כן, הוא מחזיר ערך מאוד גדול כהערכה לפעולה (פקטור 100). הבדיקה שאומרת לנו האם נאכלה רוח בעזרת הפעולה מתבצעת ע"י חישוב המרחק בין המיקום של הרוח במצב הקודם ובמצב הנוכחי. אם המרחק גדול מ-1, סימן שהרוח לא הלכה אל המיקום החדשה, אלא נאכלה והופיעה מחדש במרכז המפה.

אחרי כן, הסוכן בודק האם ישנה רוח "מפוחדת" (רוח שניתן לאכול אותה) ואם כן, הוא מחשב את המרחק אל הקרובה ביותר. הוא מוסיף ערך יחסית גדול בהתאם למרחק מהרוח המפוחדת הקרובה ביותר (פקטור 5). כך אנו קובעים כי הסוכן יעדיף לרדוף אחרי רוחות ולצוד אותן מאשר להמשיך לאכול אוכל.

לבסוף, הסוכן בודק מה האוכל הקרוב ביותר אליו ומוסיף ערך בהתאם למרחק האוכל (פקטור 1).

כיוון שהפונקציה מחזירה ערך גדול יותר לפעולות טובות יותר, בכל המקרים שבהם אנחנו רוצים מרחק קטן יותר (בכולם בעצם), הערך שמחושב הוא (אורך המפה + רוחב המפה) פחות המרחק שחושב. כך, מרחק קטן יותר יניב ערך גדול יותר. לאחר מכן, מכפילים את הערך הזה כפול הפקטור וכך מתעדפים את הפעולה שרוצים לבצע.

בכל המקרים שבהם חישבנו מרחק, ישנן 2 פונקציות המאפשרות לעשות זאת. הראשונה היא manhattanDistanceToClosestPoint והשניה היא mazeDistanceToClosestPoint. הראשונה מחשבת את מרחק מנהטן ומחזירה את הקטן ביותר. השניה מבצעת BFS עד הגעה לנקודה מסוימת, ובכך היא מתחשבת גם בקירות החוסמים את הסוכן. במבוכים שבהם אין קירות (כדוגמאת openClassic) עדיף להשתמש במרחק מנהטן (manhattanDistanceToClosestPoint), באחרים כדאי להשתמש במרחק-מבוך (mazeDistanceToClosestPoint) למרות שבשני המקרים מקבלים תוצאות די טובות (אם כי חישוב מרחק-מבוך לוקח מעט יותר זמן).

הרצתי את הסוכן 1000 פעמים, בעזרת כל אחת מהפונקציות, עבור שני המבוכים openClassic ו-mediumClassic. את קבצי הפלט של הריצות ניתן למצוא בתיקייה Q1. (הריצה בעזרת מרחק-מבוך לקחה די הרבה זמן... השניה הייתה די מהירה ולקחה בערך 15 דקות):

מבוך	פונקציית חישוב מרחק	תוצאה	קובץ פלט
openClassic	manhattanDistanceToClosestPoint	1000/1000	Q1\open_manhattan.txt
openClassic	mazeDistanceToClosestPoint	1000/1000	Q1\open_maze.txt
mediumClassic	manhattanDistanceToClosestPoint	667/1000	Q1\medium_manhattan.txt
mediumClassic	mazeDistanceToClosestPoint	814/1000	Q1\medium_maze.txt

כל הבדיקות הורצו בעזרת הפקודה הבאה (ובקוד שיניתי את שורה 130 בהתאם לפונקציית חישוב המרחק):

```
python2 pacman.py -p ReflexAgent -l <maze> -q -n 1000
```

שאלה 2

מימשתי את האלגוריתם בצורה הדומה למימוש בספר. הפונקציה calculateMinimaxState מקבלת מספר סוכן ועומק. הפונקציה מעריכה את המהלך של הסוכן הנוכחי עבור העומק שהתקבל. אם העומק הוא 0 ומספר הסוכן הוא 0, כלומר חישבנו את כל העומקים לכל הסוכנים, אז נריץ את הפונקציה evaluationFunction ונחזיר את הערך שלה (אלו הם העלים של עץ ה-minimax).

הפונקציה מורצת ע"י getActions עם הפרמטר self.depth ומספר סוכן 0. כלומר, מחשבים את עץ ה-minimax כך שהשורש שלו הוא צומת מקסימום המייצגת את המהלך של pacman ולאחר מכן ישנן k רמות (אחת לכל רוח) שמייצגות את המהלכים של כל k הרוחות. בכל הרמות הללו הצמתים הם צמתי מינימום שכן מניחים שהרוחות הן אופטימליות ויבחרו במהלך שיקטין את ניקוד השחקן. בכל קריאה לפונקציה מחושב ה-index של הסוכן הבא עבורו יש לחשב את כל המהלכים, וכאשר מגיעים לרוח האחרונה, מקטינים את ה-depth ב-1 וממשיכים לחשב עד שמגיעים לעלי העץ.

הרצתי את הסוכן MinimaxAgent 1000 פעמים על minimaxClassic עם עומק 4. ניתן למצוא את פלטי ההרצות בתיקייה Q2.

קובץ פלט	תוצאה	מאפייני הרצה
Q2\with_stop.txt	250/1000	עם המהלך Stop בתור פעולה אפשרית
Q2\no_stop.txt	691/1000	בלי המהלך Stop בתור פעולה אפשרית

שתי הבדיקות הורצו עם הפקודה:

```
python2 pacman.py -p MinimaxAgent -l minimaxClassic -a depth=4 -q -n 1000
```

שאלה 3

שידרגתי את הפונקציה calculateMinimaxState מהשאלה הקודמת והוספתי לפונקציה פרמטרים של alpha ו-beta. הם מאותחלים למינוס אינסוף ואינסוף בהתאמה ומתעדכנים במהלך הריצה בהתאם ל-value המחושב לכל מצב ובהתאם לסוג הצומת (מינימום או מקסימום). אם בשלב מסוים תנאי הגזימה מתקיים, מחזירים מפסיקים את החישוב עבור הצומת ובכך הגזימה מתבצעת.

בדומה לשאלה הקודמת, הרצתי את הסוכן AlphaBetaAgent 1000 פעמים על minimaxClassic עם עומק 4. ניתן למצוא את פלטי ההרצות בתיקייה Q3.

קובץ פלט	תוצאה	מאפייני הרצה
Q3\with_stop.txt	192/1000	עם המהלך Stop בתור פעולה אפשרית
Q3\no_stop.txt	653/1000	בלי המהלך Stop בתור פעולה אפשרית

שתי הבדיקות הורצו עם הפקודה:

```
python2 pacman.py -p AlphaBetaAgent -l minimaxClassic -a depth=4 -q -n 1000
```

שם: עמית סידס. תעודת זהות: 206768780. קורס: בינה מלאכותית 20551

שאלה 4

עדכנתי את הפונקציה המקורית calculateMinimaxState אך שיניתי את החישוב עבור צמתי מינימום (פעולות הרוחות). במקום לחשב את המינימום מכל המהלכים, סכמתי את הערכים שלהם ובסוף חישבתי ממוצע.

הרצתי את הסוכנים AlphaBetaAgent ו-ExpectimaxAgent 1000 פעמים על trappedClassic עם עומק 3.
ניתן למצוא את פלטי ההרצות בתיקייה Q4.

קובץ פלט	תוצאה	מאפייני הרצה
Q4\alphabeta.txt	0/1000	AlphaBetaAgent
Q4\expectimax.txt	500/1000	ExpectimaxAgent

שתי הבדיקות הורצו עם הפקודה:

```
python2 pacman.py -p <Agent> -l trappedClassic -a depth=3 -q -n 1000
```