הקס / HEX

מגיש: יוגב בן איבגי

ת"ז: 327871570

שם המורה: אסנת אנגלמן

בית ספר: בית הספר הריאלי העברי בחיפה

שם החלופה: למידת מכונה - Reinforcement Learning

תאריך הגשה:

תוכן עניינים:

מבוא:

בספר זה תוכלו לעיין בעבודה ותהליך הלמידה שביצעתי, ובו בזמן בתקווה ללמוד כמה דברים חדשים. הפרוייקט שפתחתי הוא משחק לוח ששמו HEX עם לוח 5X5, בעל שלוש רמות קושי: שחקן אנושי – שחקן אנושי, שחקן רנדומלי-שחקן אנושי, מילון – שחקן אנושי ורשת נוירונים – שחקן אנושי ואופציה לשחק שני אנשים (אחד על אחד).

חוקי המשחק:

שחקנים: במשחק משחקים שני שחקנים, כחול ואדום, כאשר השחקן בצבע הכחול תמיד מתחיל את המשחק.

הלוח: הלוח במשחק מורכב ממשושים שמאורגנים בצורת רשת יחודית. גבולות השחקן הכחול מוגדרים להיות צד ימין וצד שמאל וגבולות השחקן האדום מוגדרים להיות למעלה ולמטה.

מטרת המשחק: המטרה במשחק ה HEX-היא ליצור מסלול רצף חופשי של משבצות של שחקן מסוים מצד אחד של הלוח לצד השני של הלוח (בהתאם לגבולות של כל שחקן) כך שיהיה בלתי אפשרי לשחקן השני לחבר את הקצוות של המסלול עם המשבצות שלו.

מהלך המשחק: בכל תור, השחקן מסמן משבצת (ריקה) על הלוח בסימן הייחודי שלו (כחול או אדום). המשחק מתנהל עד נצחון של אחד השחקנים.

נצחון: השחקן שמצליח ליצור מסלול רצף ראשון של משבצות מצד אחד של הלוח לצד השני של הלוח, באופן שאי אפשר לחבר את הקצוות של המסלול עם המשבצות של היריב, יחשב כמנצח.

הפרויקט בעיקרו מתחלק לשניים: יישום המשחק (הלוגיקה הבסיסית והגרפיקה) ויישום למידת המכונה (עליו ארחיב בהמשך בצורה מפורטת).

למידת מכונה הוא תחום שהתחיל לצבור תאוצה בשנות התשעים, ומאז רק צמח והתפתח. התחום מאוד רחב, ורלוונטי למגוון שימושים כגון ניתוח מידע מסובך, יצירת מודלי חיזוי, זיהוי תמונה, תפעול מערכות מסובכות ועוד (הרבה מאוד) שימושים.

בפרוייקט שלי יישמתי למידת מכונה באימון ה"סוכן", שישחק במשחק. השתמשתי בשני אלגוריתמים עיקריים של לימדת מכונה, Reinforcement Learning ורשת נוירונים בנוסף השתמשתי באלגוריתם נוסף, UNION-FIND, למציאת ניצחון במשחק בצורה יעילה (לא קשור ללמידת מכונה) . כדי להשתמש באלגוריתם ולבנות את המשחק, כתבתי את הקוד בשפת התכנות העילית פייתון, שהיא אחת משפות התכנות הנפוצות בעולם. את הקוד פיתחתי בסביבת הפיתוח PyCharm.

לבניית המשחק השתמשתי בספריה pygame וליישום רשת הנוירונים השתמשתי בספריות keras, sklearn, tensorflow ו pandas.

תחילה, בחרתי בפרויקט אחר לגמרי, בסימולטור נהיגה אוטונומית. כחודש לפני הגשת הפרויקט, החלטתי לשנות המשחק תוך הבנה שלא אספיק לסיים אותו בזמן. לא היה עוד תלמיד בשכבה שעסק בנושא דומה איתו אוכל להתייעץ. בסופו של דבר הגעתי למסכנה שעליי לבחור במשחק לוח (כמו רוב התלמידים) כדי שיהיה לי עם מי להתייעץ במידת הצורך וכדי שאוכל לשמש אוזן קשבת ולייעץ לחבריי.

במהלך יצירת הפרויקט למדתי רבות על גרפיקה ופיתוח משחקי מחשב וכמובן שנפתחתי לעולם חדש ומופלא של למידת מכונה.

קריאה מהנה ומלמדת!

מבנה הפרויקט:

ספריות בהן השתמשתי:

pygame –ספרייה המשמשת לפיתוח משחקים ויישומים בסביבת Python. היא מספקת כלים ומערכות כדי ליצור משחקים ואפליקציות דו-ממדיות ותלת-ממדיות.

numpy – היא ספריית פייתון המיועדת לעיבוד מערכים נומריים בצורה יעילה. היא מציעה סוגי נתונים מתקדמים, כלים לעבודה עם מערכים מרובי מימדים (כגון מטריצות) ופעולות חשבוניות שניתן להשתמש בהם בצורה מהירה ויעילה.

random — ספרייה המאפשרת למתכנת אפשרות ליצור מספרים אקראיים בקלות וביעילות.
נעזרתי בספרייה כדי להגריל מיקומים על הלוח לצורך יישום שחקן אקראי.

tensorflow - ספרייה של בניית מודלים של למידת מכונה שפותחה ע"י גוגל, אחת מן הפופולריות ביותר כיום

keras – תת ספרייה של tensorflow, עוסקת בכל הקשור לרשתות נוירונים

Pandas – ספרייה לשימוש יעיל ונוח במסדי נתונים, בפרויקט שלי השתמשתי בספרייה כדי לייבא את מסד הנתונים שיצרתי על מנת שהרשת תלמד ממנו.

Sklearn - ספריית למידת מכונה פופולרית בפייתון, המספקת ערכת כלים פשוטה ויעילה למשימות בניהן ניתוח נתונים (כולל סיווג, רגרסיה).

תרשימי UML

המחלקה Button_hex: אחראית על תפקוד הכפתורים של המשושים במשחק

הסבר	תכונה
מיקום בציר הX ובציר הY של מרכז הכפתור	self.center
גודל הכפתור	self.size
ערך בוליאני שערכו אמת אם הכפתור נלחץ	self.clicked
הסבר	פעולה

הפעולה מזהה לחיצה של עכבר על הכפתור	def buttonPress(self):

המחלקה Hexagon: אחראית על הייצוג של המשושים

הסבר	תכונה
מיקום בציר הX ובציר הY של מרכז המשושה	self.middleX self.middleY
צבע המשושה -> כחול, אדום, לבן	self.color
בניית המשושה -> תכונות אלו אחראיות על מיקום קודקודי המשושה	self.radius self.points
האם המשושה נלקח	self.taken
כפתור	self.button
הסבר	**פעולה**
הפעולה מקבלת את מיקומו של העכבר, ומחזירה אמת במידה והעכבר נמצא על המשושה, אחרת מחזירה שקר	def mouseCollision(self, mouse_pos):

המחלקה Dictionary: מילון JSON

הסבר	תכונה
מחזרות של כתובת מיקום הקובץ (החל ממיקום תיקיית הפרויקט)	self.path
מילון	self.dic
הסבר	**פעולה**
מחזיר מילון לאחר קריאה של הקובץ החיצוני	def getDic(self):
מייצא את המילון הנתון (דריסה)	def dumpDic(self,dictionary):

המחלקה DisjointSet: שיטה יעילה לבדיקת ניצחון.

<u>למה DisjointSet</u>?

השיטה האינטואיטיבית לבדיקת ניצחון היא באמצעות מעבר רקורסיבי על הלוח במטרה למצוא מסלולים מצלע אחת לצלע הנגדית לה בלוח. דרך זו אינה יעילה בעליל, היות ולאחר כל מהלך צריך לבדוק את הניצחון.

באמצעות DisjointSet ניתן לחלק את הלוח למעיין איים/קבוצות. הקבוצה לה יש איבר בצלע אחת ובצלע הנגדית לה (צלעות ימין ושמאל עבור כחול ולמעלה ולמטה עבור אדום) תחשב מנצחת.

בהמשך אני אפרט אלגוריתם זה.

הסבר	תכונה
הגבולות של הלוח והמיקום בלוח של התאים	self.elems
מילון שמאחסן את ההורה של כל התאים	self.parent
מילון שמאחסן את הגודל של כל סט	self.size
הסבר	פעולה
הפעולה מקבלת תא ויוצרת עבורו סט חדש	def make_set(self, x):
הפעולה מקבלת תא ומחזירה את השורש של התא בעץ	def find(self, x):
מחברת בין הצמתים שמכילים את x ו y	def union(self, x, y):

המחלקה Screen: אחראית על הצגת דברים על המסך – על הגרפיקה.

הסבר	תכונה
הרזולוציה של המסך	self.RES
הגדרה של המסך (האזור עליו אנחנו מציגים דברים)	self.WIN = pygame.display.set_mode(self.RES)
הגדרה של קצב הרענון של המסך (כמה פעמים המסך מתעדכן בשניה)	self.FPS = 60
גודל הלוח בציר אחד (במקרה זה 5)	self.boardSize
רשימה של tuples המייצגים את פינות המעוין שסביב הלוח	self.boardEdges
משמשים עבור תפריט המשחק ומצבי המשחק	self.menuOption self.settingSelection

self.showMenu self.optionsPage self.aboutPage	ערכים בוליאנים המשמשים עבור בחירה במסך התפריט (איזו מסך/תפריט להציג)
*תמונות של הכפתורים בתפריט	טעינה של התמונות
self.up_arrow_pressed self.down_arrow_pressed self.escape_pressed self.enter_pressed	ערכים בוליאנים המשמשים עבור ניווט בתפריט ובמשחק
self.startingX self.startingY	משמשים עבור גלילה במסך "ABOUT"
טקסטים למסך "ABOUT"	
self.hexagons self.hexagons_Shadow	מערך דו מימדי של עצמים מסוג Hexagon המערך השני משמש עבור אפקט של צל
פעולה	הסבר
def draw_hexagon(self, hex,shadow,width=0):	הפעולה מקבלת שני משושים (אחד רגיל ואחד צל) ועובי רצוי של משושה (ברירת מחדל 0), ומציירת את המשושה (ואת הצל במידה ונתפס או במידה והעכבר חולף על פניו)
def draw_grid(self):	הפעולה מציירת את הלוח ואת המעוין שסובב אותו
def menu(self,keys)	הפעולה מקבלת רשימה של כפתורים שנלחצו כעת, ואחראית על תפקוד התפריט
def aboutText(self, pressedKey,x):	הפעולה מקבלת מחזרות של מקש שנלחץ, ומיקום בציר איקס (לצורך מיקום הטקסטים על אותו אנך) ואחראית על תפקוד מסך "ABOUT"
def gameOptions(self,pressedKey):	הפעולה מקבלת מחזרות של מקש שנלחץ, ואחראית על בחירת קושי המשחק (או חזרה למסך הקודם)
def check_keys(self,keys):	הפעולה מקבלת רשימה של מקשים שנלחצו ומחזירה מחרוזת של המקשים (למעלה, למטה, ENTER, ESCAPE)

הפעולה מקבלת מיקום בציר Y,X ומנהלת את נראות הכפתור בתפריט (המידה והוא הכפתור הנבחר, צבעו יהיה שונה)	def play(self,x,y):
	def _1v1(self,x,y):
	def quit(self,x,y):
	def average(self,x,y):
	def back(self,x,y):
	def easy(self,x,y):
	def hard(self,x,y):
	def about(self,x,y):

צביעת במסך בצבע ירוק וציור הלוח	def draw(self):
הפעולה מקבלת מחרוזת, מיקום בצירים Y,X, צבע, גודל פונט (ברירת מחדל 72) ופונט (ברירת מחדל Ariel) ומציגה כיתוב על המסך.	def showMessage(self, str, x, y, color, size=72, font="Ariel"):

המחלקה Game: אחראית על כל הקשור ללוגיקה והבאתו למסך. המחלקה יורשת מן המחלקה Screen כיוון שהיא המשך ישיר של Screen ומשתמשת בכל תכונותיה.

הסבר	תכונה
משתנה בוליאני שבא להראות למי שייך התור (לכחול או לא)	self.blueTurn
מערך דו מימדי – הצגה מספרית של הלוח (0 – ריק, 1 – כחול, 2 – אדום)	self.board
משתנים בוליאנים שבאים להראות האם נגמר המשחק והאם הנצחון שייך לכחול	self.win self.blueWin
מערך דו מימדי של אינדקסים (j,i) אפשריים בלוח	self.cells
הגדרה של גבולות הלוח	self.top_node self.bottom_node self.left_node self.right_node

יצירה של עצמים מסוג DisjointSet עבור כחול ואדום לצורך בדיקת נצחון	self.ds_red self.ds_blue
מונה של מספר הנצחונות עבור כל שחקן	self.blueWins self.redWins
הגדרה של המילונים	self.diction_1to5 self.diction_6to10 self.diction_11to15 self.diction_16to20 self.diction_21to25
מונה של מספר המהלכים	self.moves
הסבר	פעולה
איפוס של כל המשתנים לצורך משחק חדש	def newGame(self):
אחראי על המשחק. כל המשחק מתנהל דרך פעולה זו.	def gameManager(self):
במידה ומשושה מסוים נלחץ, הפעולה מעדכנת אותו (במידה ולא היה לחוץ)	def pressHex(self):
לאחר כל מהלך נבדק נצחון. הפעולה מקבלת את האינדקס האחרון ש"נכנס" ומחרוזת של השחקן (כחול או אדום) ומעדכנת את המשתנים הבוליאנים (self.win self.blueWin) האם היה נצחון	def checkWinner(self, i, j, player):
מעדכן האם היה נצחון עבור כחול או אדום	def winnerRed(self): def winnerBlue(self):
מציג את מספר הנצחונות עבור כל שחקן על המסך	def displayWins(self):
הפעולה מחזירה אינדקס רנדומלי ופנוי.	def randomIndex(self):
עורך תור רנדומלי עבור כחול/אדום	def randomBlue(self, index): def randomRed(self,index):
מחזיר אינדקס של מקום עבור שחקן חכם (מילון, Reinforcement)	def smartMove_index(self):
עורך מהלך חכם (מילון, Reinforcement)	def smartMove(self):

הפעולה מחזירה מחרוזת של הלוח (בצורה חד מימדית)	def hash(self):

המחלקה Learning: אחראית על כל הקשור לReinforcement Learning:

הסבר	תכונה
מערך numpy, הייצוג של הלוח	self.board
קבועים, לצורך ציון הלוחות באלגוריתם Reinforcement Learning, יפורט בהמשך	self.gama = 0.9 self.epsilon = 0.0001
רשימה המאחסנת את הלוחות במשחק לפי הסדר (מקום ראשון הלוח הראשון...)	self.matchList
רשימה של המילונים במשחק, לצורך יצירת קובץ CSV מן הלוחות	self.dicList
משתנה בוליאני שבא להראות למי שייך התור (לכחול או לא)	self.blueTurn
הגדרה של המילונים במשחק	self.diction_1to5 self.diction_6to10 self.diction_11to15 self.diction_16to20 self.diction_21to25
מונה של מספר המהלכים	self.moves
משתנים בוליאנים שבאים להראות האם נגמר המשחק והאם הנצחון שייך לכחול	self.blueWins self.redWins
מחרוזת שמציינת איזה שחקן ניצח	self.winner
מערך דו מימדי של אינדקסים (j,i) אפשריים בלוח	self.cells
הגדרה של גבולות הלוח	self.top_node self.bottom_node self.left_node self.right_node

יצירה של עצמים מסוג DisjointSet עבור כחול ואדום לצורך בדיקת נצחון	self.ds_red self.ds_blue
הסבר	פעולה
איפוס של כל המשתנים לצורך משחק חדש	def newGame(self):
הפעולה מחזירה מחרוזת של הלוח (בצורה חד מימדית)	def hash(self):
הפעולה מחזירה אינדקס רנדומלי ופנוי.	def randomIndex(self):
עורך תור רנדומלי עבור כחול/אדום	def randomBlue(self, index): def randomRed(self,index):
לאחר כל מהלך נבדק נצחון. הפעולה מקבלת את האינדקס האחרון ש"נכנס" ומחרוזת של השחקן (כחול או אדום) ומעדכנת את המשתנים הבוליאנים (self.win self.blueWin) האם היה נצחון	def checkWinner(self, i, j, player):
מעדכן האם היה נצחון עבור כחול או אדום	def winnerRed(self): def winnerBlue(self):
ניהול משחק של שני שחקנים רנדומליים	def randomVSrandom(self):
ניהול משחק של שחקן חכם (70 אחוז חכם) מול רנדומלי	def smart70VSrandom(self):
ניהול משחק של שחקן חכם מול רנדומלי	def smart_playerVSrandom(self):
ניהול משחק של שחקן חכם מול אנושי	def humanVSsmart_player(self):
נתינת ציון ללוח במילון	def grading(self,reward):
הפעולה מחזירה אינדקס של מקום פנוי בלוח, הבחירה נעשית בצורה "חכמה" – יפורט בהמשך	def smartMove(self):
ניהול משחק בטרמינל	def terminalGame(lr):
ניקיון של המילונים (במטרה שירוצו מהר יותר) הורדה של הערכים שבין -0.2 ל0.2	def cleanDic(lr):
הדפסה של מספר הלוחות בכל מילון ומספר הלוחות בסה"כ	def statistics(lr):
שיפור המילונים	def train_70_30(gamesNumber,lr):

בדיקה של שיעור הניצחונות (באחוזים) עבור כל שחקן.	def checkWinningRate(gamesNumber,lr):
ריצת מספר מסוים של משחקים לצורך בניית המילון.	def train(gamesNumber,lr):
מחיקה של המילונים.	def erase(lr):
המרת המילונים לקובץ CSV.	def convertTOcsv(lr):

המחלקה Main: מפעילה את המשחק ואחראית על סגירתו

אלגוריתמים:

Reinforcement learning:

למידה באמצעות חיזוקים היא למידה שמתבצעת על ידי סוכנים (Agents) אשר מנסים למקסם את הביצועים שלהם בסביבה מסויימת. האלגוריתם אינו דורש מידע קודם, ויוצר את המידע שלו בהתבסס על מידע קודם שיצר. האלגוריתם פועל ממש כמו בני אדם, השואפים ללמוד מטעויות, כך גם המודל מנסה לא לחזור על טעויות שביצע בעברו .השיטה עובדת באמצעות הענקת תמלוגים ועונשים בהתאם להתנהגות הסוכן בסביבה והביצועים שלו .ככל שניתן לסוכן שלנו ללמוד מיותר משחקים/טעויות, כך הוא יהיה "חכם" יותר.

המידע שנאסף נשמר במילון json. המפתח (Key) הוא מחרוזת שטוחה (חד מימדית) שמייצגת את הלוח הדו מימדי. הערך (Value) הוא tuple שמורכב מציון מסוים (דרך החישוב תפורט בהמשך) וממספר הפעמים שאותו לוח חזר על עצמו. את המידע אחסנתי בחמישה מילונים שונים כדי ליעל את הקוד. כל מילון אחראי על חמישה מהלכים שונים במשחק (הראשון על מהלכים 1-5 וכן הלאה).

הסבר	תכונה
מופע של עצם מסוג משחק	game
משתנה בוליאני אשר מצביע על ריצת המשחק, במידה והמשחק נסגר המשתנה יהיה False	run
מקציב את מספר הפעמים שהמסך מתרענן	clock

מילון ראשון – 614704 לוחות

מילון שני – 27904129 לוחות

מילון שלישי – 38337349 לוחות

מילון רביעי – 32493706 לוחות

מילון חמישי – 14876004 לוחות

השימוש במספר מילונים יעיל יותר משימוש במילון אחד כיוון שבשיטה של מספר מילונים אין צורך לעבור על 115 מיליון לוחות ששמורים, אלה על 23 מיליון לוחות בממוצע (הפעולה תדע לאיזה מילון לגשת באמצעות מונה של מספר הצעדים שנעשו עד מהלך מסוים במשחק).

תחילה, הסוכן (השחקן האדום) לומד ממשחקים "רנדומליים", כלומר מתנהלים מספר מסוים של משחקים (5 מיליון משחקים) שבהם השחקן הכחול והשחקן האדום בוחרים מהלכים אקראיים. נצחון עבור השחקן האדום מתקבל את הציון 1, ועבור השחקן הכחול הציון 1-. כל המלהכים "שבאמצע" מקבלים ציונים בטווח שבין 1- ל 1. ככל שהמהלך קרוב יותר לנצחון, ערכו (גודל הציון ללא סימן) יהיה גדול יותר.

לאחר הרצת המשחקים הרנדומליים, אחוזי הנצחון שעמדו על 64.3% לטובת האדום לא סיפקו אותי כלל וכלל. כדרך לשיפור המילון, הרצתי (שוב) שלושה מיליון משחקים רק בדרך קצת שונה. במקום שהסוכן (השחקן האדום) יהיה 100% רנדומלי, הוא התבסס על נתונים מהריצות הקודמות, וביצע מהלכים לפיהם.

ציון לוחות המילון תלוי במרחקן מניצחון. מהלכי המשחק נשמרים ברשימה (כל תא מציין לוח) בסדר כרונולוגי, כך שהתא הראשון מייצג את המהלך האחרון (מהלך הניצחון). הציון עבור כל לוח הוא הממוצע של הציונים הקודמים (במידה ויש) ביחד עם הציון החדש. הציון החדש הוא 0.9 (מספר קרוב לאחד) בחזקת מיקום הלוח ברשימה כפול הציון לניצחון עבור השחקן (כחול 1-, אדום 1). הכפילה בציון לנצחון נועדה כדי לקבוע את סימן הציון (פלוס - אדום או מינוס - כחול).

כיצד הסוכן יודע באיזה מהלך לבחור מתוך המילון?

הדבר עובד בדומה לאלגוריתם למציאת מקסימום במערך דו מימדי (בערך...). בהינתן לוח מסוים, הפעולה למהלך חכם (בכל איטרציה) ממקמת משושה אדום בלוח ובודקת האם הציון של הלוח "החדש" גדול מהציון המקסימלי האחרון שהיה, במידה וכן, היא שומרת את האינדקס, מחזירה את הלוח למצבו הקודם וחוזרת על הפעולות שצוינו עד שתעבור על כל המקומות הפנויים בלוח.

בחזרה לניסיון שיפור אחוזי הניצחון, לאחר הרצת 3 מיליון המשחקים, היה שיפור משמעותי בשיעורי הנצחון, 72.3 אחוזים, אך עדיין לא הסתפקתי. מזל שקיים האלגוריתם ANN או בכינויו, רשת נוירונים אשר יכול לעזור לי לשפר משמעותית את אחוזי הניצחון של הסוכן.

ANN – Artificial Neural Network:

במשחק HEX בעל לוח 5X5 קיימים סביב 420 מיליארד לוחות אפשריים (וככל שהלוח גדל, גם מספר הלוחות האפשריים גדל). קשה מאוד לכסות את המספר הרב של הלוחות באמצעות Reinforcement learning לכן רשת נוירונים היא הפתרון למציאת המהלך הנכון מבין כל הלוחות האפשריים.

רשת נוירונים היא אחד מהמודלים המשמשים ללמידת מכונה. המודל מחקה את אופן העבודה של המוח האנושי, ואת האופן בו נוירונים אנושיים (יחידת התפקוד הבסיסית של המוח האנושי) מתקשרים ומאותתים אחד לשני .

תהליך עיבוד המידע:

הנוירון מקבל מידע מנוירונים ומעביר מידע לנוירונים אחרים. החלקים ברשת שמהם מגיעים inputים נקראים קשתות, בכל קשת עובר מידע מנוירון מסוים לנוירון אחר בשכבה הבאה, ולכל אחד מהקלטים (inputs) חשיבות אחרת. הקלטים הם מספרים שהם תוצאה של תהליך העיבוד בנוירון הקודם.

לכל קשת יש מספר המכונה משקולת שמייצג את חשיבות הקלט שהגיע (קלט חשוב יותר ←משקל גדול יותר). בדרך כלל המשקולות נעות בערכים שבין 1- ל 1. המשקולת היא ממש כמו הציון במילון מאלגוריתם Reinforcement learning. כל נוירון מקבל שני ערכים מכל נוירון אחר, הקלט והמשקולת. הנוירון מבצע חישובים מתמטיים על הקלטים שקיבל כדי לחשב את הפלט שיעביר לנוירונים הבאים.

מבנה הרשת:

בכל רשת שכבת קלט, פלט, ושכבה מוסתרת (או שכבות מוסתרות). ככל שיש יותר שכבות ברשת ויותר מידע עליו הרשת מתבססת, כך הרשת תחשב ל"חזקה יותר".

לפני מימוש האלגוריתם, יצרתי dataset של קובץ csv המכיל את כל המידע מן כל המילונים, חוץ מהמונה של מספר הופעות הלוח. עמודה זו (עמודת המונים) מיותרת ולוקחת מקום חשוב בזיכרון כיוון שלרשת אין שום צורך בה.

את מסד הנתונים (dataset) חילקתי בצורה של 25% - 75%. חילוק המידע נעשה עבור הערכת ביצועי המודל (כשמאמנים את המודל על שני חלקים שונים של מסד הנתונים, ניתן לראות כיצד המודל מתמודד עם מידע חדש לגמרי).

ברשת שלי 9 שכבות (7 שכבות מוסתרות). בשכבה הראשונה 25 קלטים (כמספר התאים בלוח). מספר הנוירונים יורד (בכל שכבה מספר הנוירונים הוא חצי ממספר הנוירונים בשכבה הקודמת). הירידה היא מ256 נוירונים לנוירון אחד בודד.

מספר האפוקים (Epochs) מציין את מספר הפעמים ש"עוברים" על המידע במהלך אימון הרשת. בכל אפוק מתבצע "ערבוב" של המידע. ערבוב עוזר למנוע מהמודל ללמוד את סדר הנתונים ויכול להוביל ללמידה טובה יותר במהלך האימון.

כאמור, הרשת מחזירה ציון אחד (משקולת). בדומה למציאת התא ה"טוב ביותר" ב Reinforcement learning בהינתן לוח מסוים, הפעולה למהלך חכם (בכל איטרציה) ממקמת משושה אדום בלוח ובודקת האם הציון של הלוח "החדש" גדול מהציון המקסימלי האחרון שהיה, במידה וכן, היא שומרת את האינדקס, מחזירה את הלוח למצבו הקודם וחוזרת על הפעולות שצוינו עד שתעבור על כל המקומות הפנויים בלוח.

אלגוריתם יעיל לבדיקת ניצחון באמצעות DisjointSet:

השיטה האינטואיטיבית לבדיקת ניצחון היא באמצעות מעבר רקורסיבי על הלוח במטרה למצוא מסלולים מצלע אחת לצלע הנגדית לה בלוח. דרך זו אינה יעילה בעליל, היות ולאחר כל מהלך צריך לבדוק את הניצחון עם פעולה בעלת סיבוכיות זמן ריצה גבוהה. סיבוכיות זמן הריצה של בדיקת הניצחון בשיטה זו היא $o(4^{N^2})$. באמצעות המחלקה DisjointSet ניתן לייצר פעולה בעלת סיבוכיות של $o(1)$ (הרבה יותר יעיל).

בדרך זו, ניתן להסתכל על הלוח בצורה של איים, כלומר להסתכל על כל "קבוצה" של משושים (המחוברים אחד לשני) כאי. שחקן שמבין כל האיים שברשותו לפחות אי אחד שגובל בגבולות עבור אותו השחקן (כחול – ימין ושמאל, אדום – למעלה ולמטה) יחשב למנצח.

לאחר כל מהלך (הצבה של משושה) נוצר אי חדש. מיד לאחר יצירת האי, האלגוריתם יודע למזג בין האי החדש לאיים שבהם הוא גובל (לכדי אי אחד) ובודקת האם אותו אי ממוזג גובל בגבולות שצויינו לעיל. האלגוריתם נחשב ליעיל מאוד כיוון שאינו עובר על מאגר מסויים של נתונים (בהשוואה לדרך הראשונה שהוצעה, אינו עובר על מערך דו מימדי) והוא יודע באמצעות תנאי אחד להגיד האם ישנו מנצח או לא.

GUI ומדריך למשתמש:

המשחק:

הפרויקט עוסק במשחק הקס (HEX). גודלו של הלוח הוא 5X5 והוא מורכב ממשושים. אחזור על חוקי המשחק בקצרה: לכל שחקן יש כלי בצבע אחר (כחול או אדום). בכל תור, אחד השחקנים צובע משבצת ריקה באדום או בכחול. מטרתו של כל שחקן היא לקשר את צידי הלוח המסומנים בצבע שלו, בשרשרת רצופה של משושים בצבע זה. השחקן הראשון המצליח במשימה, ומונע בכך מן השחקן השני להשלים שרשרת משלו, מנצח. על מנת לנצח על השחקן הכחול "לחבר" בין צלע שמאל וצלע ימין, ועל השחקן האדום "לחבר" בין הצלע העליונה והצלע התחתונה.

במשחק שלי יש שלוש רמות קושי למשחק: שחקן אנושי – שחקן אנושי, שחקן רנדומלי-שחקן אנושי, מילון – שחקן אנושי ורשת נוירונים – שחקן אנושי ואופציה לשחק שני אנשים (אחד על אחד).

דרישות מינימליות:

python – גרסא 3.11

pygame – גרסא 2.5.2

tensorflow – גרסא 2.16.1

keras – גרסא 3.2.1

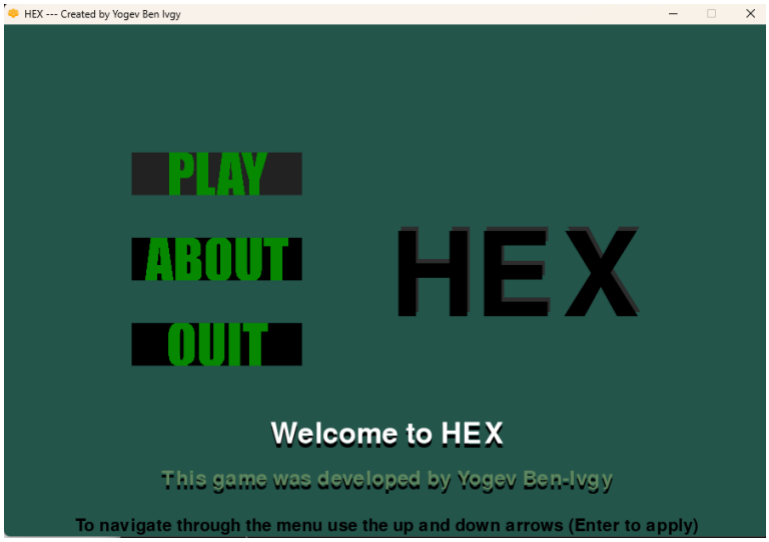
numpy – 1.26.4

Pandas – 2.2.2

Sklearn (scikit-learn) – 1.4.2

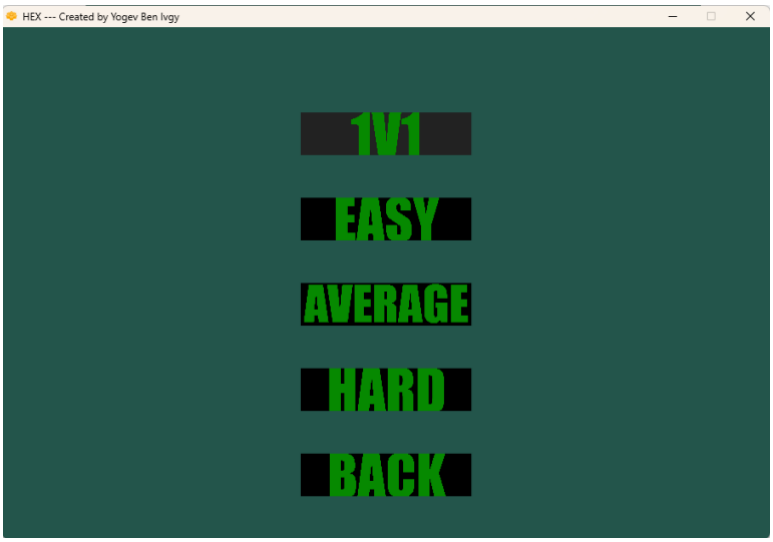
מסך הפתיחה (תפריט):

עם פתיחת המשחק, מופיע מסך הפתיחה שמשמש כתפריט. הגלילה בתפריט מתבצעת באמצעות מקשים (למעלה, למטה, enter ,esc). במסך זה שלוש אופציות:



PLAY – מוביל למסך אחר אשר אחראי על רמות הקושי במשחק. (אחד נגד אחד, קל, בינוני, קשה).

בנוסף בעזרת המקש BACK ניתן לחזור למסך הקודם.

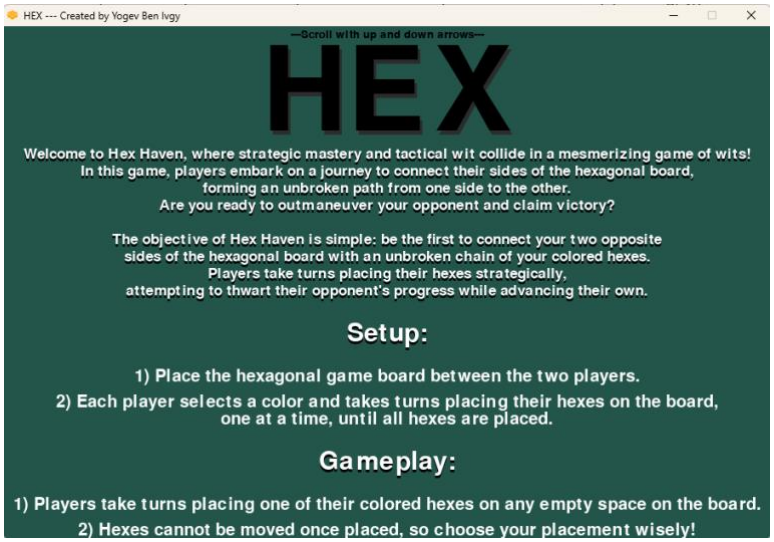


ABOUT – מוביל למסך אשר מסביר על המשחק בקווים כלליים. הניווט בו נעשה באמצעות החיצים, esc כדי לחזור אחורה

QUIT – יציאה מהמשחק וסגירתו.

המשחק עצמו:

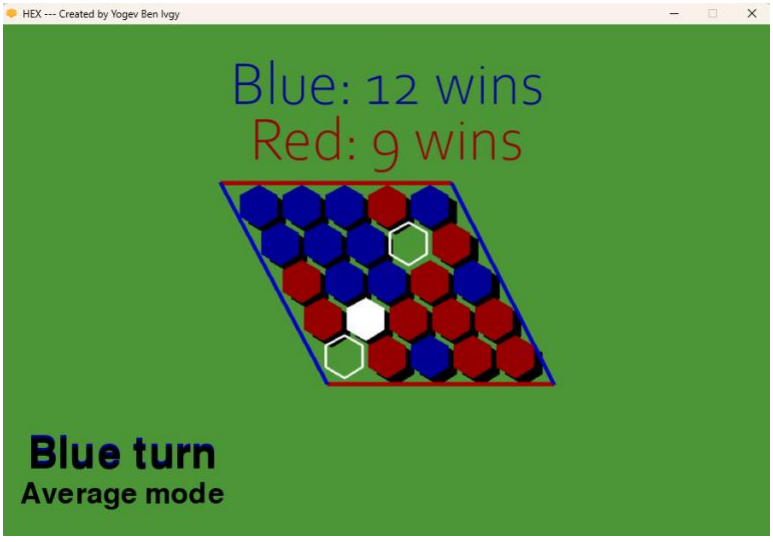
לאחר בחירת רמת הקושי המתאימה, ניתן להתחיל לשחק במשחק. במסך זה ניתן לראות את הלוח ביחד עם הגבולות של כל שחקן. בנוסף, קיימות כל מיני כתוביות חיווי. למעלה, מופיעים מספר



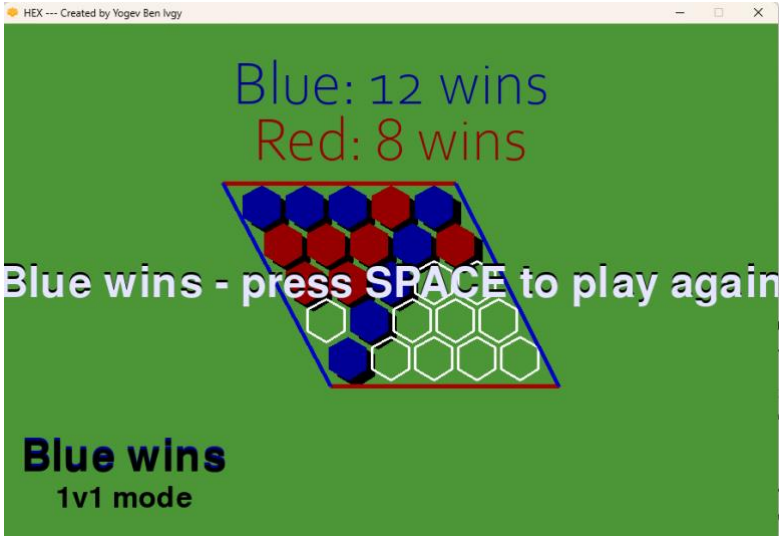
הניצחונות עבור כל שחקן, ולמטה מופיעה רמת הקושי וכתובית המראה למי שייך התור, ומי ניצח במידה והמשחק נגמר.

באמצעות לחיצה על המקש esc ניתן לחזור לתפריט של בחירת רמות הקושי במשחק (ניתן גם במהלך משחק לשנות את רמת הקושי).

הלוח מורכב מ25 משושים, אשר כל אחד מהם משמש ככפתור אשר עליו לוחצים עם המקש השמאלי בעכבר. כל משושה נצבע בהתאם לשחקן שבחר אותו (אי אפשר יהיה ללחוץ על אותו משושה בשנית). במידה והמשושה ריק והעכבר "מרחף" מעליו, המשושה יצבע בצבע לבן במטרה להראות לשחקן באיזה משושה הוא עתיד לבחור.



עם סיום המשחק, תופיע כתובית לבנה, שתפקידה להראות שהמשחק נגמר ולהציע משחק חדש (באמצעות לחיצה על המקש space).



רפלקציה:

העבודה על הפרויקט על משחק הייתה חווייה מאתגרת, מרתקת ומלמדת.

העבודה על הפרויקט הייתה מאתגרת מהרבה מובנים. נדרשתי להבין את חוקי המשחק, לפתח אלגוריתם יעיל לבדיקת ניצחון, ללמוד את הבסיס לפיתוח משחק מחשב ולתכנת ממשק משתמש יעיל ונעים לעין.

הקשיים העיקריים עבורי לא היו בתהליך כתיבת הקוד, אלא היו בתהליך יצירת תוכנית העבודה וחשיבה על הקבצים והמחלקות אשר עליי ליצור. בנוסף, המקום בזיכרון שהמילונים לקחו (7.4 ג״גה) הקשה עליי במעבר ממחשב למחשב, וגרם לי להיסחב עם כונן חיצוני או עם המחשב האישי שלי לבית הספר. כאשר רציתי לעבוד על הקוד (בעיקר על הגרפיקה) ולא נאלצתי לשחקן חכם, השתמשתי במילונים ״חלשים״, כדי שיטענו מהר וכדי שלא יצרך להתאים את הקוד למצב בו אין מילונים כלל וכלל.

הפרויקט לימד אותי רבות על למידת מכונה, לקח את רמת התכנות שלי בפייתון כמה רמות קדימה ועזר לי ללמוד על עצמי ועל הדרך שכדאי לי לתכנן את הזמן בפרויקטים עתידיים.

בזכות הפרויקט הבנתי שאני מאוד אוהב לתכנת, וישיבה ממושכת מול המחשב תוך חשיבה ממושכת גורמת לי להרגיש נפלא.

העבודה על הפרויקט חיזקה את הראייה הכוללת שלי בנוגע ליצירת קוד מכל סוג שהוא. כשבונים את המשחק ישנם תנאים ומצבים רבים שצריך לקחת בחשבון והם משפיעים על תקינות המשחק, המשאבים שהוא צורך ומהלכו. מצבים אלו גרמו לי להסתכל באופן עמוק על כל דבר שאתקל בו (לא רק בפרויקט במחשבים) ולפעול בצורה מחושבת תוך התייחסות לכל בעיה שעלולה להיווצר.

להבא, אני לא אדחה כתיבה/שיפור וייעול הקוד (ובמידה וכן, אני ארשום לעצמי תזכורת), כי בסופו של דבר שוכחים ומצטברים הרבה מאוד בעיות. בנוסף, לאחר כתיבה ועריכה של פעולות וקטעי קוד חשוב לא לשכוח לתעד, כדי לא להגיע למצב של תיעוד כל הקוד בבת אחת, וכדי להימנע ממצב שבו קיים קטע קוד שאני לא בקיא בו.

אני מאמין שביצעתי את הפרויקט על הצד הטוב ביותר שלו, למרות שתמיד יש לאן להתקדם ולשפר. יכול להיות במידה והיה לי יותר זמן, הייתי עובד על ייעול הקוד או שאולי הייתי בוחר בשפת תכנות אחרת.

אני שמח שבחרתי בפרויקט זה, ובכללי אני שמח שבחרתי ללימוד 10 יחידות לימוד של המקצוע האהוב עליי. אני סבור שבזמן הקרוב לא הייתי נחשף לעולם זה, ופרויקט זה ממש פקח את עיניי.

קוד הפרויקט:

המחלקה Button_hex:

```
import pygame
import math

class Button_hex():
    def __init__(self, x, y, size=100, rotation=30):
        self.center = (x, y)
        self.size = size
        self.rotation = rotation
        self.clicked = False

    def buttonPress(self):
        coll = False
        pos = pygame.mouse.get_pos()
        distance = math.sqrt((self.center[0] - pos[0])**2 + (self.center[1] - pos[1])**2)
        if distance <= self.size and pygame.mouse.get_pressed()[0] == 1 and not self.clicked:
            self.clicked = True
            coll = True
        if pygame.mouse.get_pressed()[0] == 0:
            self.clicked = False
        return coll
```

המחלקה Hexagon:

```
import math
import Button

class Hexagon:
    def __init__(self, x, y, radius, color=(255, 255, 255)):
        self.middleX = x
        self.middleY = y
        self.color = color
        self.radius = radius
        self.button=Button.Button_hex(x, y, 25)
        self.points = [
            (x + radius * math.cos(math.radians(30 + 60 * i)), y + radius * math.sin(math.radians(30 + 60 * i)))
            for i in range(6)
        ]
        self.taken = False

    def mouseCollision(self, mouse_pos):
        #Retruns wheres the mouse is on a hexagon
        # Mouse position is a tuple (mouse_x, mouse_y)
        mouse_x, mouse_y = mouse_pos

        # Check if the mouse position is within the bounding rectangle of the hexagon
        if (
            self.middleX - self.radius <= mouse_x <= self.middleX + self.radius and
            self.middleY - self.radius <= mouse_y <= self.middleY + self.radius
```

```
    ):
        # Check if the mouse position is inside the hexagon using ray casting algorithm
        odd_nodes = False
        j = 5
        for i in range(6):
            if (
                (self.points[i][1] < mouse_y and self.points[j][1] >= mouse_y) or
                (self.points[j][1] < mouse_y and self.points[i][1] >= mouse_y)
            ) and (
                self.points[i][0] + (mouse_y - self.points[i][1]) / (self.points[j][1] - self.points[i][1]) * (self.points[j][0]
- self.points[i][0]) < mouse_x
            ):
                odd_nodes = not odd_nodes
            j = i
        return odd_nodes
    else:
        return False
```

המחלקה Dictionary:

```
import json

class Dict():

    def __init__(self, path):
        self.path = path
        self.dic = json.load(open(self.path))


    def getDic(self):
        # Open and read the JSON file
        with open(self.path, 'r') as json_file:
            # Load JSON from the file into a dictionary
            json_dict = json.load(json_file)

        return json_dict

    def dumpDic(self,dictionary):
        #Export the given dictionary (overwrite)
        with open(self.path, 'w')as json_file:
            json.dump(dictionary, json_file)
```

המחלקה DisjointSet:

```
class DisjointSet:
    def __init__(self, elems):
        self.elems = elems #The cells on the game board and the board boundaries
        self.parent = {}  # Dictionary to store the parent of each element
        self.size = {}  # Dictionary to store the size of each set
        for elem in elems:
            # Initialize each element as a singleton set
```

```
        self.make_set(elem)

    def make_set(self, x):
        # Create a new set with a single element
        self.parent[x] = x  # The element is its own parent initially
        self.size[x] = 1  # The size of the set is initially 1

    def find(self, x):
        # Find the representative (root) of the set containing element x
        if self.parent[x] == x:
            return x
        else:
            # Path compression: Update the parent of x to the root
            self.parent[x] = self.find(self.parent[x])
            return self.parent[x]

    def union(self, x, y):
        # Union operation: Merge the sets containing elements x and y
        root_x = self.find(x)
        root_y = self.find(y)
        if root_x == root_y:
            return  # Elements are already in the same set
        elif self.size[root_x] < self.size[root_y]:
            # Attach smaller set (root_x) to the larger set (root_y)
            self.parent[root_x] = root_y
            self.size[root_y] += self.size[root_x]
        else:
            # Attach smaller set (root_y) to the larger set (root_x)
            self.parent[root_y] = root_x
            self.size[root_x] += self.size[root_y]
```

המחלקה Screen:

```
import pygame
import Hexagon

'''
This class is in charge of the game environment
'''

#colors
YELLOW = (255,232,0)
RED = (170,0,0)
RED_dark = (75,0,0)
GRAY = (45,45,45)
WHITE = (255,255,255)
GREEN = (76,150,56)
BLACK = (0,0,0)
BLUE = (0,0,200)
BLUE_dark = (0,0,75)


pygame.init()
class Screen:
```

```
def __init__(self):
    self.RES= (900,600) #Screen resolution


    self.WIN = pygame.display.set_mode(self.RES) #Set the window

    self.FPS = 60 #Refresh rate


    self.boardSize=5 #Size of the board will be 5*5

    self.boardEdges  = [(255, 185), (525, 185), (380, 420), (645, 420)]

    ''' Menu '''
    self.menuOption = "play"  # used for the menu (play, quit, options)
    self.settingSelection = "1v1"  # 1v1, easy, average , hard , back

    self.showMenu = True #Show the menu or not
    self.optionsPage=False  #Show the options page or not
    self.aboutPage = False #Show the about page or not
    self.menu_surface = pygame.Surface(self.RES, pygame.SRCALPHA)
    self.menu_surface.fill((35, 85, 75))

    #Pictures of the menu buttons
    self.play1Pic = pygame.transform.scale((pygame.image.load('images/PLAY1.png')), (200,50))
    self.play2Pic = pygame.transform.scale((pygame.image.load('images/PLAY2.png')), (200,50))
    self.quit1Pic = pygame.transform.scale((pygame.image.load('images/QUIT1.png')), (200,50))
    self.quit2Pic = pygame.transform.scale((pygame.image.load('images/QUIT2.png')), (200,50))
    self._1v1_1Pic = pygame.transform.scale((pygame.image.load('images/1V1_1.png')), (200, 50))
    self._1v1_2Pic = pygame.transform.scale((pygame.image.load('images/1V1_2.png')), (200, 50))
    self.average1Pic = pygame.transform.scale((pygame.image.load('images/AVERAGE1.png')), (200, 50))
    self.average2Pic = pygame.transform.scale((pygame.image.load('images/AVERAGE2.png')), (200, 50))
    self.back1Pic = pygame.transform.scale((pygame.image.load('images/BACK1.png')), (200, 50))
    self.back2Pic = pygame.transform.scale((pygame.image.load('images/BACK2.png')), (200, 50))
    self.easy1Pic = pygame.transform.scale((pygame.image.load('images/EASY1.png')), (200, 50))
    self.easy2Pic = pygame.transform.scale((pygame.image.load('images/EASY2.png')), (200, 50))
    self.hard1Pic = pygame.transform.scale((pygame.image.load('images/HARD1.png')), (200, 50))
    self.hard2Pic = pygame.transform.scale((pygame.image.load('images/HARD2.png')), (200, 50))
    self.about1Pic = pygame.transform.scale((pygame.image.load('images/ABOUT1.png')), (200, 50))
    self.about2Pic = pygame.transform.scale((pygame.image.load('images/ABOUT2.png')), (200, 50))

    #used for navigation
    self.up_arrow_pressed = False
    self.down_arrow_pressed = False
    self.escape_pressed = False
    self.enter_pressed = False

    #about page text
    self.startingX = 450
    self.startingY = 80

    self.about1 = "Welcome to Hex Haven, where strategic mastery and tactical wit collide in a mesmerizing
game of wits!"
    self.about2 = "In this game, players embark on a journey to connect their sides of the hexagonal board,"
    self.about3="forming an unbroken path from one side to the other."
    self.about4 = "Are you ready to outmaneuver your opponent and claim victory?"
```

```
        self.about5 = "The objective of Hex Haven is simple: be the first to connect your two opposite"
        self.about6 ="sides of the hexagonal board with an unbroken chain of your colored hexes."
        self.about7="Players take turns placing their hexes strategically,"
        self.about8 = "attempting to thwart their opponent's progress while advancing their own."
        self.about9 = "Setup:"
        self.about10 = "1) Place the hexagonal game board between the two players."
        self.about11 = "2) Each player selects a color and takes turns placing their hexes on the board,"
        self.about12 = "one at a time, until all hexes are placed."
        self.about13 = "Gameplay:"
        self.about14="1) Players take turns placing one of their colored hexes on any empty space on the board."
        self.about15 = "2) Hexes cannot be moved once placed, so choose your placement wisely!"
        self.about16 = "3) The game continues until one player successfully"
        self.about17 = "connects their two sides with an unbroken chain of their colored hexes."
        self.about18 = "Winning:"
        self.about19 = "The player who successfully connects their two sides first wins the game!"
        self.about20 = "Whether through careful planning or clever blocking,"
        self.about21 ="victory in Hex Haven requires strategic thinking and anticipation of your opponent's
moves."

        self.hexagons= [[0 for j in range(self.boardSize)] for i in range(self.boardSize)]
        for i in range(self.boardSize):
            for j in range(self.boardSize):
                self.hexagons[i][j]=(Hexagon.Hexagon(300+25*j+25*2*i,212+44*j,25))

        self.hexagons_Shadow = [[0 for j in range(self.boardSize)] for i in range(self.boardSize)]
        for i in range(self.boardSize):
            for j in range(self.boardSize):
                self.hexagons_Shadow[i][j] = (Hexagon.Hexagon(310 + 25 * j + 25 * 2 * i, 220 + 44 * j, 25))



    def draw_hexagon(self, hex,shadow,width=0):
        #Draw the given hexagon
        if hex.color == WHITE:
            width=3

        if hex.mouseCollision(pygame.mouse.get_pos()):
            pygame.draw.polygon(self.WIN, (0,0,0),shadow.points)
            width=0

        if hex.color!=WHITE:
            pygame.draw.polygon(self.WIN, (0,0,0),shadow.points)
        pygame.draw.polygon(self.WIN, hex.color,hex.points,width)


    def draw_grid(self):
        #Draw a 5x5 grid
        for i in range(self.boardSize):
            for j in range(self.boardSize):
                self.draw_hexagon(self.hexagons[i][j],self.hexagons_Shadow[i][j])

        lineW=5
        pygame.draw.line(self.WIN, RED, self.boardEdges[0], self.boardEdges[1],width=lineW)
        pygame.draw.line(self.WIN, RED, self.boardEdges[2], self.boardEdges[3],width=lineW)
        pygame.draw.line(self.WIN, BLUE, self.boardEdges[0], self.boardEdges[2],width=lineW)
        pygame.draw.line(self.WIN, BLUE, self.boardEdges[1], self.boardEdges[3],width=lineW)
```

```
  def menu(self,keys):
    pressedKey = self.check_keys(keys)
    self.WIN.blit((self.menu_surface), (0, 0))
    if not self.optionsPage and not self.aboutPage:
      self.showMessage("Welcome to HEX" , self.RES[0] / 2, self.RES[1] / 10 * 8 + 5, BLACK, 50)
      self.showMessage("Welcome to HEX" , self.RES[0] / 2, self.RES[1] / 10 * 8, WHITE, 50)

      self.showMessage("HEX", 603,295, GRAY,200)
      self.showMessage("HEX", 600,300, BLACK,200)

      self.showMessage("This game was developed by Yogev Ben-Ivgy", self.RES[0] / 2, self.RES[1] / 9 * 8 + 5,
BLACK, 36)
      self.showMessage("This game was developed by Yogev Ben-Ivgy", self.RES[0] / 2, self.RES[1] / 9 * 8,
(95,135,95), 36)

      self.showMessage("To navigate through the menu use the up and down arrows (Enter to apply)",
self.RES[0] / 2, self.RES[1] / 8 * 8 - 12 , BLACK, 30)



      if self.menuOption == "play":
        if pressedKey =="up":
          self.menuOption = "quit"
        elif pressedKey =="down":
          self.menuOption="about"

      elif self.menuOption == "about":
        if pressedKey == "up":
          self.menuOption = "play"
        elif pressedKey == "down":
          self.menuOption = "quit"

      elif self.menuOption == "quit":
        if pressedKey == "up":
          self.menuOption = "about"
        elif pressedKey == "down":
          self.menuOption = "play"
      self.play(150,150)
      self.about(150,250)
      self.quit(150,350)

      if pressedKey=="enter":
        if self.menuOption=="quit":
          return "quit"
        elif self.menuOption=="play":
          self.optionsPage=True
        elif self.menuOption=="about":
          self.aboutPage=True

    elif self.optionsPage:
      return self.gameOptions(pressedKey)
    elif self.aboutPage:
      self.aboutText(pressedKey,self.startingX)

      # return self.menuOption
    return "menu"
```

```
def aboutText(self, pressedKey,x):
    if pressedKey == "down":
        if  -100<=self.startingY<= 80:
            self.startingY-=20
    elif pressedKey == "up":
        if -120 <= self.startingY <= 60:
            self.startingY+=20


    y=self.startingY

    self.showMessage("---Scroll with up and down arrows---", self.startingX, self.startingY-70, BLACK, 20)

    self.showMessage("HEX", self.startingX+3, self.startingY +5, GRAY, 200)
    self.showMessage("HEX", self.startingX, self.startingY, BLACK, 200)

    y+=70

    self.showMessage(self.about1, self.startingX,y+ 3, BLACK, 25)
    self.showMessage(self.about1, self.startingX,y, WHITE, 25)

    y+=20

    self.showMessage(self.about2, x, y + 3, BLACK, 25)
    self.showMessage(self.about2, x, y, WHITE, 25)

    y+=20

    self.showMessage(self.about3, x, y + 3, BLACK, 25)
    self.showMessage(self.about3, x, y, WHITE, 25)

    y+=20

    self.showMessage(self.about4, x, y + 3, BLACK, 25)
    self.showMessage(self.about4, x, y, WHITE, 25)

    y += 40

    self.showMessage(self.about5, x, y + 3, BLACK, 25)
    self.showMessage(self.about5, x, y, WHITE, 25)

    y += 20

    self.showMessage(self.about6, x, y + 3, BLACK, 25)
    self.showMessage(self.about6, x, y, WHITE, 25)

    y += 20

    self.showMessage(self.about7, x, y + 3, BLACK, 25)
    self.showMessage(self.about7, x, y, WHITE, 25)

    y += 20

    self.showMessage(self.about8, x, y + 3, BLACK, 25)
    self.showMessage(self.about8, x, y, WHITE, 25)

    y += 50
```

```
self.showMessage(self.about9, x, y + 3, BLACK, 45)
self.showMessage(self.about9, x, y, WHITE, 45)

y += 50

self.showMessage(self.about10, x, y, WHITE, 30)

y += 30

self.showMessage(self.about11, x, y, WHITE, 30)

y += 20

self.showMessage(self.about12, x, y, WHITE, 30)

y += 50

self.showMessage(self.about13, x, y + 3, BLACK, 45)
self.showMessage(self.about13, x, y, WHITE, 45)

y += 50

self.showMessage(self.about14, x, y, WHITE, 30)

y += 30

self.showMessage(self.about15, x, y, WHITE, 30)

y += 30

self.showMessage(self.about16, x, y, WHITE, 30)

y += 20

self.showMessage(self.about17, x, y, WHITE, 30)

y += 50

self.showMessage(self.about18, x, y + 3, BLACK, 45)
self.showMessage(self.about18, x, y, WHITE, 45)

y += 50

self.showMessage(self.about19, x, y, WHITE, 30)

y += 20

self.showMessage(self.about20, x, y, WHITE, 30)

y += 20

self.showMessage(self.about21, x, y, WHITE, 30)
```

```
    if pressedKey=="escape":
        self.aboutPage=False
def gameOptions(self,pressedKey):
    if self.settingSelection == "1v1":
        if pressedKey == "up":
            self.settingSelection = "back"

        elif pressedKey == "down":
            self.settingSelection = "easy"

    elif self.settingSelection  == "easy":
        if pressedKey == "up":
            self.settingSelection = "1v1"

        elif pressedKey == "down":
            self.settingSelection  = "average"

    elif self.settingSelection == "average":
        if pressedKey == "up":
            self.settingSelection = "easy"

        elif pressedKey == "down":
            self.settingSelection = "hard"

    elif self.settingSelection == "hard":
        if pressedKey == "up":
            self.settingSelection = "average"

        elif pressedKey == "down":
            self.settingSelection = "back"

    elif self.settingSelection == "back":
        if pressedKey == "up":
            self.settingSelection = "hard"

        elif pressedKey == "down":
            self.settingSelection = "1v1"

    if pressedKey == "enter":
        if self.settingSelection == "1v1":
            return "1v1"
        elif self.settingSelection == "easy":
            return "random"
        elif self.settingSelection == "average":
            return "dictionary"
        elif self.settingSelection == "hard":
            return "web"
        elif self.settingSelection == "back":
            self.optionsPage = False

    self._1v1(350,100)
    self.easy(350,200)
    self.average(350,300)
    self.hard(350,400)
    self.back(350,500)
```

```
def check_keys(self,keys):
    # Check for up arrow key
    if keys[pygame.K_UP] and not self.up_arrow_pressed:
        self.up_arrow_pressed = True
        return "up"  # Return "up" if up arrow key is pressed for the first time
    elif not keys[pygame.K_UP]:
        self.up_arrow_pressed = False

    # Check for down arrow key
    if keys[pygame.K_DOWN] and not self.down_arrow_pressed:
        self.down_arrow_pressed = True
        return "down"  # Return "down" if down arrow key is pressed for the first time
    elif not keys[pygame.K_DOWN]:
        self.down_arrow_pressed = False

    # Check for enter key
    if keys[pygame.K_RETURN] and not self.enter_pressed:
        self.enter_pressed = True
        return "enter"  # Return "enter" if enter key is pressed for the first time
    elif not keys[pygame.K_RETURN]:
        self.enter_pressed = False

    # Check for escape key
    if keys[pygame.K_ESCAPE] and not self.escape_pressed:
        self.escape_pressed = True
        return "escape"  # Return "enter" if enter key is pressed for the first time
    elif not keys[pygame.K_ESCAPE]:
        self.escape_pressed = False

    return None  # Return None if neither arrow key is pressed or if they are held down

def play(self,x,y):
    if self.menuOption=="play":
        self.WIN.blit(self.play2Pic, (x, y))
    else:
        self.WIN.blit(self.play1Pic, (x, y))
def _1v1(self,x,y):
    if self.settingSelection == "1v1":
        self.WIN.blit(self._1v1_2Pic, (x, y))
    else:
        self.WIN.blit(self._1v1_1Pic, (x, y))
def quit(self,x,y):
    if self.menuOption == "quit":
        self.WIN.blit(self.quit2Pic, (x, y))
    else:
        self.WIN.blit(self.quit1Pic, (x, y))
def average(self,x,y):
    if self.settingSelection == "average":
        self.WIN.blit(self.average2Pic, (x, y))
    else:
        self.WIN.blit(self.average1Pic, (x, y))
def back(self,x,y):
    if self.settingSelection == "back":
        self.WIN.blit(self.back2Pic,(x, y))
    else:
```

```
        self.WIN.blit(self.back1Pic, (x, y))
    def easy(self,x,y):
        if self.settingSelection == "easy":
            self.WIN.blit(self.easy2Pic,(x, y))
        else:
            self.WIN.blit(self.easy1Pic, (x, y))
    def hard(self,x,y):
        if self.settingSelection == "hard":
            self.WIN.blit(self.hard2Pic,(x, y))
        else:
            self.WIN.blit(self.hard1Pic, (x, y))

    def about(self,x,y):
        if self.menuOption == "about":
            self.WIN.blit(self.about2Pic,(x, y))
        else:
            self.WIN.blit(self.about1Pic, (x, y))


    ''' Display things on the screen'''
    def draw(self):
        #Game enviorment
        self.WIN.fill(GREEN)
        self.draw_grid()


    # Shows a message on the screen
    def showMessage(self, str, x, y, color, size=72, font="Ariel"):
        FONT = pygame.font.SysFont(font, size)
        text_surface = FONT.render(str, True, color)
        text_rect = text_surface.get_rect()
        text_rect.center = (x, y)
        self.WIN.blit(text_surface, text_rect)
```

המחלקה Game:

```
import pygame

import Button
from Screen import Screen
import numpy as np
import random
import Hexagon
import Group
import Dictionary
BLUE = (0,0,150)
RED = (150,0,0)
BLACK = (0,0,0)

class Game(Screen):
    def __init__(self):
        super().__init__()

        # Initialize game state variables
```

```
        self.blueTurn = True  # Indicates if it's currently blue player's turn
        self.board = np.zeros((self.boardSize, self.boardSize))  # Represents the game board
        self.win = False  # Indicates if the game has been won
        self.blueWin = False  # Indicates if the blue player has won

        '''Winning check:'''
        # Define board positions
        self.cells = [(i, j) for i in range(self.boardSize) for j in range(self.boardSize)]
        self.top_node = (-1, 0)
        self.bottom_node = (self.boardSize, 0)
        self.left_node = (0, -1)
        self.right_node = (0, self.boardSize)

        # Initialize disjoint sets for red and blue players
        self.ds_red = Group.DisjointSet(self.cells + [self.top_node, self.bottom_node])
        self.ds_blue = Group.DisjointSet(self.cells + [self.left_node, self.right_node])

        # Connect boundary cells to corresponding boundary nodes
        for i in range(self.boardSize):
            self.ds_red.union((0, i), self.top_node)
            self.ds_red.union((self.boardSize - 1, i), self.bottom_node)
            self.ds_blue.union((i, 0), self.left_node)
            self.ds_blue.union((i, self.boardSize - 1), self.right_node)

        # Initialize win counts
        self.blueWins = 0
        self.redWins = 0


        # reinforcment learning

        self.diction_1to5JSON = Dictionary.Dict("dictionaries/diction1-5.json")
        self.diction_6to10JSON = Dictionary.Dict("dictionaries/diction6-10.json")
        self.diction_11to15JSON = Dictionary.Dict("dictionaries/diction11-15.json")
        self.diction_16to20JSON = Dictionary.Dict("dictionaries/diction16-20.json")
        self.diction_21to25JSON = Dictionary.Dict("dictionaries/diction21-25.json")

        self.diction_1to5 = self.diction_1to5JSON.dic
        self.diction_6to10 = self.diction_6to10JSON.dic
        self.diction_11to15 = self.diction_11to15JSON.dic
        self.diction_16to20 = self.diction_16to20JSON.dic
        self.diction_21to25 = self.diction_21to25JSON.dic

        self.moves = 1




    def newGame(self):
        #Reset the variables for a new game
        self.blueTurn = True
        self.board = np.zeros((self.boardSize, self.boardSize))
        self.win=False
        self.blueWin=False

        # Reset disjoint sets for red and blue players
        self.ds_red = Group.DisjointSet(self.cells + [self.top_node, self.bottom_node])
```

```
        self.ds_blue = Group.DisjointSet(self.cells + [self.left_node, self.right_node])

        # Reconnect boundary cells to corresponding boundary nodes
        for i in range(self.boardSize):
            self.ds_red.union((0, i), self.top_node)
            self.ds_red.union((self.boardSize - 1, i), self.bottom_node)
            self.ds_blue.union((i, 0), self.left_node)
            self.ds_blue.union((i, self.boardSize - 1), self.right_node)

        # Reinitialize hexagons and buttons
        self.hexagons = [[0 for j in range(self.boardSize)] for i in range(self.boardSize)]
        for i in range(self.boardSize):
            for j in range(self.boardSize):
                self.hexagons[i][j] = (Hexagon.Hexagon(300 + 25 * j + 25 * 2 * i, 212 + 44 * j, 25))

        self.moves=1

    def gameManager(self):
        keys = pygame.key.get_pressed()
        if self.showMenu:
            self.gameMode=self.menu(keys)
            if self.gameMode=="quit":
                self.showMenu=False
                return "quit"
            elif self.gameMode in ["1v1","random","dictionary"]:
                self.showMenu=False
        else:
            self.draw() #Create the game enviorment
                #Random game
            if self.gameMode=="random":
                self.showMessage("Easy mode", 140, 550, (0, 0, 0), 50)
                if not self.win:
                    if self.blueTurn:
                        self.showMessage("Blue turn", 140, 500, BLUE)
                        self.showMessage("Blue turn", 140, 503, BLACK)
                        self.pressHex()


                    if not self.blueTurn and not self.win:
                        self.randomRed(self.randomIndex())


                #Regular game --> 1v1
            #Display info on the screen
            if self.gameMode == "1v1":
                self.showMessage("1v1 mode", 140, 550, (0, 0, 0), 50)
                if not self.win:
                    self.pressHex()
                    if self.blueTurn:
                        self.showMessage("Blue turn", 140, 500, BLUE)
                        self.showMessage("Blue turn", 140, 503, BLACK)

                    if not self.blueTurn:
                        self.showMessage("Red turn", 140, 500, RED)
                        self.showMessage("Red turn", 140, 503, BLACK)
```

```
        if self.gameMode == "dictionary":
            self.showMessage("Average mode", 140, 550, (0, 0, 0), 50)
            if not self.win:
                if self.blueTurn:
                    self.showMessage("Blue turn", 140, 500, BLUE)
                    self.showMessage("Blue turn", 140, 503, BLACK)
                    self.pressHex()

                    if not self.blueTurn and not self.win:
                        self.smartMove()


        if keys[pygame.K_ESCAPE]:
            self.showMenu=True
        if self.win:
            if self.blueWin:
                self.showMessage("Blue wins", 140, 500, BLUE)
                self.showMessage("Blue wins", 140, 503, BLACK)
                self.showMessage("Blue wins - press SPACE to play again", 450, 300, BLACK)
                self.showMessage("Blue wins - press SPACE to play again", 450, 303, (230,230,255))

            else:
                self.showMessage("Red wins", 140, 500, RED)
                self.showMessage("Red wins", 140, 503,BLACK)
                self.showMessage("Red wins - press SPACE to play again", 450, 300, BLACK)
                self.showMessage("Red wins - press SPACE to play again", 450, 303, (255,230,230))

        self.displayWins()



    def pressHex(self):
        for i in range(self.boardSize):
            for j in range(self.boardSize):
                if self.hexagons[i][j].button.buttonPress():
                    if not self.hexagons[i][j].taken:

                        if self.blueTurn:
                            self.hexagons[i][j].color = BLUE
                            self.board[j][i]=1
                            self.checkWinner(j,i,"blue")

                        else:
                            self.hexagons[i][j].color = RED
                            self.board[j][i]=2
                            self.checkWinner(j, i, "red")

                        self.blueTurn=not self.blueTurn
                        self.hexagons[i][j].taken=True
                        self.moves += 1


    def checkWinner(self, i, j, player):
        code = 1 if player == 'blue' else 2
        self.board[i][j] = code
        for nei_i, nei_j in [(i + 1, j), (i + 1, j - 1), (i, j + 1), (i, j - 1), (i - 1, j), (i - 1, j + 1)]:
            if 0 <= nei_i < self.boardSize and 0 <= nei_j < self.boardSize and code == self.board[nei_i][nei_j]:
```

```
            if player == 'red':
                self.ds_red.union((nei_i, nei_j), (i, j))
            else:
                self.ds_blue.union((nei_i, nei_j), (i, j))
    if player=="blue":
        self.winnerBlue()
    if player =="red":
        self.winnerRed()




def winnerBlue(self):
    if self.ds_blue.find(self.left_node) == self.ds_blue.find(self.right_node):
        self.win=True
        self.blueWin=True
        self.blueWins+=1

def winnerRed(self):
    if self.ds_red.find(self.top_node) == self.ds_red.find(self.bottom_node):
        self.win=True
        self.redWins+=1


def displayWins(self):
    self.showMessage(f"Blue: {self.blueWins} wins",450,75, BLUE,font="Corbel")
    self.showMessage(f"Red: {self.redWins} wins",450,140, RED,font="Corbel")

'''Random moves for blue and red'''
def randomIndex(self): #Returns tuple of the random index
    i = random.randint(0, self.boardSize-1)
    j = random.randint(0, self.boardSize-1)
    while  self.hexagons[i][j].taken:
        i = random.randint(0, self.boardSize-1)
        j = random.randint(0, self.boardSize-1)
    return (i,j)


def randomBlue(self, index):
    self.hexagons[(index[0])][index[1]].color = BLUE
    self.board[index[1]][index[0]] = 1
    self.blueTurn = False
    self.hexagons[index[0]][index[1]].taken = True
    self.checkWinner(index[1],index[0],"blue")

def randomRed(self,index):
    self.hexagons[(index[0])][index[1]].color = RED
    self.board[index[1]][index[0]] = 2
    self.blueTurn=True
    self.hexagons[index[0]][index[1]].taken = True
    self.checkWinner(index[1],index[0],"red")
    self.moves+=1



def smartMove_index(self):
    #Returns the index of the cell --> dictionary
```

```
row = -1
col = -1
if not self.blueTurn:
    maxGrade = -1
    for i in range(self.boardSize):
        for j in range(self.boardSize):
            if not self.hexagons[j][i].taken:
                self.board[i][j] = 2
                stringBoard = self.hash()

                if 1 <= self.moves <= 5:
                    if stringBoard in self.diction_1to5JSON.dic and self.diction_1to5JSON.dic[stringBoard][
                        0] > maxGrade:
                        maxGrade = self.diction_1to5JSON.dic[stringBoard][0]
                        row = i
                        col = j
                        # print("diction 1")

                elif 6 <= self.moves <= 10:
                    if stringBoard in self.diction_6to10JSON.dic and self.diction_6to10JSON.dic[stringBoard][
                        0] > maxGrade:
                        maxGrade = self.diction_6to10JSON.dic[stringBoard][0]
                        row = i
                        col = j
                        # print("diction 2")


                elif 11 <= self.moves <= 15:
                    if stringBoard in self.diction_11to15JSON.dic and self.diction_11to15JSON.dic[stringBoard][
                        0] > maxGrade:
                        maxGrade = self.diction_11to15JSON.dic[stringBoard][0]
                        row = i
                        col = j
                        # print("diction 3")


                elif 16 <= self.moves <= 20:
                    if stringBoard in self.diction_16to20JSON.dic and self.diction_16to20JSON.dic[stringBoard][
                        0] > maxGrade:
                        maxGrade = self.diction_16to20JSON.dic[stringBoard][0]
                        row = i
                        col = j
                        # print("diction 4")

                else:
                    if stringBoard in self.diction_21to25JSON.dic and self.diction_21to25JSON.dic[stringBoard][
                        0] > maxGrade:
                        maxGrade = self.diction_21to25JSON.dic[stringBoard][0]
                        row = i
                        col = j
                        # print("diction 5")

                self.board[i][j] = 0
    if row == -1 and col == -1:
        print("random")
        return "random"
```

```
            self.board[row][col] = 2
            self.blueTurn = True

        return (row, col)

    def smartMove(self):
        smart = self.smartMove_index()
        if smart=="random":
            self.randomRed(self.randomIndex())
        else:
            i, j = smart
            self.board[i][j] = 2
            self.hexagons[j][i].color = RED
            self.hexagons[j][i].taken = True
            self.blueTurn = True
            self.checkWinner(i, j, "red")
            self.moves+=1


    def hash(self):
        str = np.array2string(self.board)
        str1 = ""
        for i in str:
            if i in ['1', '0', '2']:
                str1 += i
        return str1
```

המחלקה Main:

```
import Game
import pygame



def main():
    game=Game.Game() #Setting up the screen and the basic enviorment

    run = True #This is for the game loop

    clock = pygame.time.Clock()

    while run:

        clock.tick(game.FPS) #Refresh rate
        # print(game.gameMode)
        # Goes through a list of all events happening - Event handler
        for event in pygame.event.get():
            # Checks if the user quit the window
            if event.type == pygame.QUIT:
                run = False
            #Check for pressed keys
            if event.type == pygame.KEYDOWN:
                #Getting out of the menus (end game manu and start manu)
                if event.key == pygame.K_SPACE:
```

```
            if game.win:

                game.newGame()

    if game.gameManager()=="quit":
        run=False
    #     # Get mouse coordinates
    # mouse_x, mouse_y = pygame.mouse.get_pos()
    #
    # # Print mouse coordinates
    # print(f"Mouse coordinates: ({mouse_x}, {mouse_y})")
    #Update the screen
    pygame.display.flip()
  pygame.quit()


if __name__ == "__main__":
  pygame.display.set_caption('HEX --- Created by Yogev Ben Ivgy')
  pygame.display.set_icon(pygame.image.load('images/icon.webp'))
  main()
```