פרויקט הנדסת תוכנה:
משחק דוראק

עומר לביא

323957183

בית הספר הריאלי בית בירם

2019 תשע״ט

תוכן עניינים

1. הקדמה

כחלק מפרויקט במקצוע הנדסת תוכנה נדרשתי להכיר ולהשתמש בשפת python. השתמשתי בסביבת העבודה PyCharm ובה נעזרתי בתוכנת הגרפיקה kivy. הפרויקט מטרתו משחק בשימוש AI (בינה מלאכותית). אני בחרתי את המשחק דוראק. במשחק דוראק, בניגוד למשחקי קלפים אחרים, אין מנצחים אלא רק מפסיד אחד, הנקרא "דוראק". המשחק מיועד לשניים עד שישה שחקנים, אך מקובל שמשחקים שלושה או ארבעה בלבד. מטרת המשחק היא להישאר ללא קלפים בסיומו. השחקן האחרון שנותר עם קלפים בידו הוא ה"דוראק" (המפסיד).

2. אפיון מערכת

שם המשחק: דוראק

מומחיות המערכת: חיקוי שחקן אנושי

קהל היעד: 6+

מטרות עיקריות:
המערכת הינה הדמיה של משחק של שחקן נגד מחשב חכם.

חוקי המשחק:

איך משחקים: תחילה, כל משתתף מקבל 6 קלפים מחפיסה, שבה נמצאים קלפים מכל ארבעת הסוגים עלה, תלתן, לב ויהלום, ומכל המספרים מ 6 ומעלה. לאחר מכן, לוקחים את הקלף הבא בחפיסה ומניחים אותו כלפי מעלה הסוג של קלף זה (יהלום / תלתן / עלה / לב) נקבע כ"קוזר", כלומר הסוג החזק במשחק הנוכחי. ה"קוזר" הוא שונה ממשחק למשחק, תלוי בקלף שיוצא.

המחשב בוחר את השחקן המתחיל לפי ה"קוזר" הנמוך ביותר שיש לשחקן מבלי לגלות לאף שחקן את הערך של ה"קוזר". המשתתף שנבחר להתחיל, "תוקף את המשתתף" שאחריו בתור. כלומר, מניח לפניו קלף כלשהו. על המשתמש ה"נתקף" להשיב להתקפה על ידי הנחת קלף בעל **אותו סוג** ,**ומספר גבוה יותר** מזה שהניח התוקף, על הקלף התוקף.

כאן נכנס ה"קוזר" לתמונה. הסוג של קלף ה"קוזר", יכול להשיב על התקפה של כל קלף אחר, אפילו אם הוא נמוך יותר מהקלף התוקף, אך אם הקלף התוקף הוא גם מסוג ה"קוזר", על מספרו להיות גדול יותר.

לדוגמה :סוג ה"קוזר" הוא יהלום . משתתף 1 תוקף את המשתתף שאחריו עם נסיך תלתן. המשתתף הנתקף, יכול להשיב על הנסיך תלתן עם כל קלף, לא משנה מאיזה מספר, מסוג ה"קוזר". כלומר, הוא יכול להשיב על הנסיך תלתן עם **מלך תלתן**,לדוגמה, אך הוא גם יכול להשיב עליו עם **ליהלום** ,למרות ש-7 קטן יותר מנסיך אך הוא מסוג ה"קוזר" ולכן יכול לגבור על הנסיך. לכל אורך ההתקפה, יכול המשתתף התוקף לתקוף את המשתתף הנתקף בקלפים בעלי אותו מספר כמו הקלפים בזירה. זאת אומרת, התוקף יכול להוסיף להתקפה קלף רק אם הוא נמצא כבר ב"זירה". ניתן לתקוף עד 6 קלפים.

לדוגמה :משתתף 1 תוקף את המשתתף שאחריו עם נסיך תלתן. ברגע זה, כל משתתף במשחק, כולל התוקף, יכולים להוסיף להתקפה עוד קלפים, בתנאי שהם **נסיכים בלבד.** אם המשתתף שמגן, משיב על הנסיך עם מלכה תלתן, לדוגמה, אז התוקף ושאר המשתתפים יכולים להוסיף להתקפה **גם נסיכים וגם מלכות** ,לא משנה מאיזה סוג. אם המשתתף המגן הצליח להשיב בהצלחה על כל הקלפים שתקפו אותו, ולתוקפים אין עוד קלפים להוסיף (לפי החוקים), לוקחים את כל הקלפים ושמים אותם בערימת "משומשים" בצד, וכעת השחקן שאותו תקפו, תוקף את השחקן הבא אחריו. במקרה והמשתתף המגן לא הצליח לבלום את כל ההתקפות עליו, ולא הצליח להשיב על קלף מסויים, עליו לקחת את כל הקלפים שהשתתפו בהתקפה לידו. כאשר דבר זה קורה, המשתתף המגן "מפסיד תור" והתור עובר לשחקן שאחריו, שתוקף את השחקן הבא.

חשוב: לאחר כל סיום תור, על כל אחד מהמשתתפים לקחת קלפים מהערימה הראשית עד שיש בידו 6 קלפים. כאשר החבילה הראשית נגמרת, ממשיכים לשחק כרגיל עד שלמשתתף אחד נגמרים הקלפים, והוא מוכתר כמנצח.

3. חלוקה למחלקות

3.1 מחלקת Board

מחלקה זו אחראית על לוח המשחק וביצוע המהלכים

Board
Layout

תכונות

blank1
turn_attack
turn_deffedned
sources
list_cards
pos_rectangle_list
cards_battle
list_players
computer_pos_card
list_exist_cards
list_known
start_button
cheat_flag
passed
grabbed
buttonrest
quzar
show_num_cards
lab1
lab2
lets_start_button
name_one
name_two

פעולות

__init__(**kwargs)
creat_list_cards(self, list_cards)
restart_game(self)
start_game(self)
scores(self)
remove_card(self,card)
start(self, v)
cheated(self,v)
grabbed_2(self, v)
pass_turn(self, v)
remove_from_unkown(self,card)
change_size_of_cards(self)
give_cards(self)
change_quzar(self)
prepre_battle(self)
cards_on_board(self)
who_will_start(self)
stuck_cards(self)
free_cards(self):
stuck_the_other(self)
organize_the_cards(self, player)
take_cards_from_list(self,player)
take_cards_from_board(self, player)
deffened_computer(self, player, dt)
find_not_deffedned(self)

```
find_empty_place(self)
how_much_shapes(self,card)
will_attack_me(self,card)
add_card_to_board(self, card_place, rec_place,
player,flag)
special_condition(self,card)
find_card_to_deffend(self, card, player)
build_board(self,board_none_grafic)
create_regu_list_players(self,board_none_grafic)
find_attack_option(self,player,board_none_grafic)
find_defence_option(self,player,board_none_grafic)
recurse_find_card(self, options, player,
flag,board_none_grafic)
find_this_card(self,card)
list_battle_attack(self)
attack_computer(self, player, dt)
is_full(self)
list_end(self)
check_can_be_in(self,battle_list,card)
lost_good_one(self)
attack_decision(self, battle_list,player)
is_empty(self)

print_board(self,matrix)
end_game_check(self)
```

הסבר על תכונות:

הסבר	שם תכונה
מטרת התכונה לשמור כתובת לתמונה של קלף ריק.	blank1
מטרת התכונה לסמן מי תוקף (מחשב או שחקן).	turn_attack
מטרת התכונה לסמן מי מתגונן (מחשב או שחקן).	turn_deffedned
מערך של כתובת של תמונות כל הקלפים למשחק. בעזרתם יורכבו תמונות הקלפים.	sources
מערך שמכיל את הקלפים של הקופה.	list_cards
מערך שמכיל את תמונות קלפים ריקים, מערך להצגת קלפי היריב באופן גרפי.	pos_rectangle_list
מערך שמכיל את אזור הקרב. כלומר, מערך של שישה מקומות שמכיל רשימה שגודלה בין 0-2 (כתלות במספר הקלפים). למעשה זה ייצוג לא גרפי(נתוני) של מהלכי המשחק בין השחקנים.	cards_battle
מערך שמכיל את רשימות השחקנים. כל תא הוא רשימה בפני עצמה המכילה קלפים (בהמשך יפורט על מחלקה card).	list_players
מערך שמכיל תמונות קלפים רקים לשם הסתרת הקלפים האמיתיים.	computer_pos_card

list_exist_cards	מערך שמכיל כל קלף שלא התגלה עדיין. (בפועל לא נעשה שימוש בפעולה זו, הסבר באלגוריתם).
list_known	מערך שמכיל קלפים הידועים למחשב שנמצאים אצל השחקן.
start_button	כפתור הנמצא במסך הפתיחה ומעביר את זה למסך משחק.
cheat_flag	משתנה בוליאני שקובע אם הופעל כפתור cheated.
cheated(self,v)	כפתור הנמצא במסך משחק ומשתמשים בו לראיית קלפי המחשב. כפתור זה מיועד לבדיקות.
passed	כפתור הנמצא במסך משחק ומאפשר מעבר של תור.
grabbed	כפתור הנמצא במסך משחק ומאפשר מעבר של תור ולקיחת קלפים (הפסד התגוננות).
buttonrest	כפתור הנמצא במסך משחק ומאפשר מעבר של תור.
quzar	קלף שמשמשים בו רק לקביעת סוג הקוזר.
show_num_cards	לייבל שמטרתה לציין כמה קלפים נשארו בחפישה
lab1	לייבל שמטרתה לציין מי מתקיף.
lab2	לייבל שמטרתה לציין מי מתגונן.
lets_start_button	כפתור במסך פתיחה שמטרתו להתחיל את המשחק.
name_one	תווית שמטרתה לסמן את המספר של השחקן האנושי(1).
name_two	תווית שמטרתה לסמן את המספר של המחשב(2).

הסבר על פעולות :

טענת כניסה	הסבר	שם הפעולות
	פעולה בונה של מחלקת board(הלוח)	__init__(**kwargs)
	פעולה בונה את קופת הקלפים.	creat_list_cards(self)
	הפעולה מאתחלת את המחלקה ומתחילה מחדש את המשחק.	restart_game(self)
	פעולה הבונה נתוני המחלקה ומכינה את הגרפיקה למשחק.	start_game(self)
	פעולה המעדכנת את ניקוד כל קלף במשחק לפי ערכו(מספרו ולפי האם קוזר)	scores(self)
פעולה מקבלת קלף כלשהוא.	פעולה המורידה קלפים מהרשימה שאמורה להיות הרשימה של השחקן האנושי בסוף המשחק.(זה קורה כל פעם שקלף מתגלה למחשב).	remove_card(self,card)
	פעולה המעבירה את מסך הפתיחה למסך המשחק.	start(self, v)
	פעולה המאפשרת לראות את קלפי שחקן המחשב(רק לבדיקות). פעולה המתרחשת על ידי לחיצת כפתור.	cheated(self,v)
	פעולה המאפשרת לקיחת קלפים(הפסד להתקפה) והעברת תור. פעולה המתרחשת על ידי לחיצת כפתור.	grabbed_2(self, v)
	פעולה המאפשרת העברת תור. פעולה המתרחשת על ידי לחיצת כפתור.	pass_turn(self, v)
פעולה מקבלת קלף כלשהוא.	פעולה המורידה קלפים מרשימת הקלפים הידועים לשחקן המחשב שנמצאים אצל השחקן האנושי.	remove_from_unkown(self,card)
	פעולה המעדכנת את הגודל הגרפי המתאים לקלפים.	change_size_of_cards(self)

	פעולה המחלקת קלפים לכל שחקן.	give_cards(self)
	פעולה המשנה לכל הקלפים את המשתנה הבוליאני של האם זהו קוזאר או לא.	change_quzar(self)
	פעולה ששמה בצורה גרפית מלבנים שצלעותיהם צבועות כדי ששם השחקנים ידעו איפה מתרחש הקרב ולאיפה לגרור קלפים.	prepre_battle(self)
	פעולה המוסיפה את הקלפים ללוח בצורה גרפית.	cards_on_board(self)
	פעולה הקובעת מי יתחיל(לפי הקוזאר הנמוך יותר) ומתחילה את המשחק._פעולה המתרחשת על ידי לחיצת כפתור.	who_will_start(self)
	פעולה שמקבעת את הקלפים ולא נותנת למשמש להזיז אותם.	stuck_cards(self)
	פעולה שמשחררת את הקלפים ונותנת למשתמש להזיז אותם.	free_cards(self):
	פעולה שמקבעת את הקלפים רק שלא מתאימים לקרב.	stuck_the_other(self)
פעולה המקבלת מספר שחקן.	פעולה שמסדרת את הקלפים.	organize_the_cards(self, player)
פעולה המקבלת מספר שחקן.	פעולה שלוקחת קלפים מהקופה ברגע שלשחקן יש פחות משישה קלפים.	take_cards_from_list(self,player)
פעולה המקבלת מספר שחקן.	פעולה שלוקחת קלפים מהקופה ברגע שלשחקן יש פחות משישה קלפים.	take_cards_from_board(self, player)
פעולה המקבלת מספר שחקן ומשתנה המייצג זמן.	פעולה שעוברת על הקלפים ומזמנת פעולה שקובעת איזה קלף להגן באמצעותו. כלומר פעולה שמבצעת מעבר על קלפי ההתקפה ומחפשת קלפי הגנה.	deffened_computer(self, player, dt)
	פעולה המחפשת קלף שלא מוגן. פעולה מחזירה מספר אם נמצא קלף כזה אחרת שקר.	find_not_deffedned(self)
	פעולה המחפשת מקום ריק ללא קלפים. פעולה מחזירה מספר אם נמצא מקום כזה אחרת שקר.	find_empty_place(self)

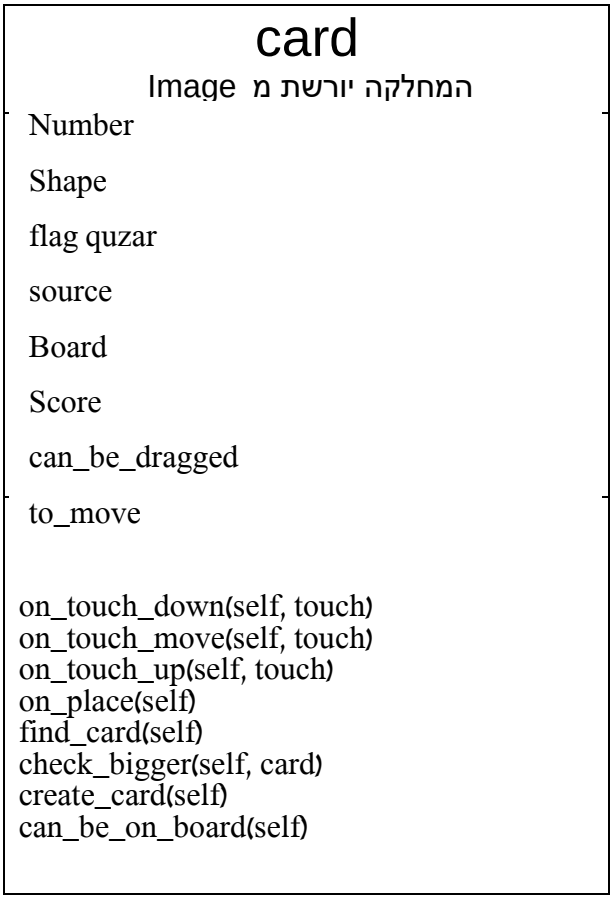
how_much_shapes(self,card)	פעולה מחזירה כתלות במספר הצורות שנספרו בחפישה ניקוד שישמש לטקטיקה. פעולה מחזירה כתלות במספר הצורות שנספרו בחפישה.	פעולה המקבלת קלף כלשהוא.
will_attack_me(self,card)	פעולה המקבלת קלף כלשהוא וקובעת אם יש לקלף הזה קלף עם אותו מספר אצל הקלפים שידועים למחשב ונמצאים אצל השחקן האנושי. כלומר, מחפשת סיכוי שתהיה התקפה נוספת. פעולה מחזירה אמת אם נמצא קלף.	פעולה המקבלת קלף כלשהוא.
add_card_to_board(self, card_place, rec_place, player,flag)	פעולה המעדכנת את מצב הקלף באופן הגרפי והנתוני ברשימת השחקן של המחשב ובקרב.	פעולה המקבלת אינדקס של קלף בחפישה של המחשב, אינדקס של מיקום בקרב ומשתנה בוליאני אם זה התקפה או הגנה.
special_condition(self,card)	פעולה המחליטה אם מדובר בקלף קוזר אם כן אז זה לא סוף המשחק הפעולה לא תאפשר שימוש בקלף. פעולה המחזירה אמת אם זה קוזר וברשימה חמישה קלפים ומטה אחרת שקר.	פעולה המקבלת קלף כלשהוא.
find_card_to_deffend(self, card, player)	פעולה המחליטה באמצעות איזה קלף להגן אם מדובר על התחלת המשחק יש לה טקטיקה מסויימת אם כבר אין קלפים בחפישה באמצעות brutfource.	פעולה המקבלת קלף ומספר שחקן.
build_board(self,board_none_grafic)	פעולה הבונה לוח משחק ללא תוספות גרפיות. פעולה מחזירה אינדקס של קלף ברשימת השחקן אם מצאה אחרת שקר.	פעולה מקבלת לוח לא גרפי ריק.
create_regu_list_players(self,board_none_grafic)	פעולה המעדכנת את חפיסות השחקנים בלוח ללא קלפים גרפיים.	פעולה מקבלת לוח לא גרפי ריק.
find_attack_option(self,player,board_none_grafic)	פעולה המחפשת את כל האופציות ההתקפה	פעולה מקבלת

לוח לא גרפי ריק ומספר שחקן.	האפשריות וכל התקפה בונה לוח חדש עם ההתקפה הספציפית בעזרת copy. פעולה מחזירה רשימה של לוחות עם כל מצבי ההתקפה האפשריים.	
פעולה מקבלת לוח לא גרפי ריק ומספר שחקן.	פעולה המחפשת את כל האופציות ההגנה האפשריות וכל התקפה בונה לוח חדש עם ההגנה הספציפית בעזרת copy. פעולה מחזירה רשימה של לוחות עם כל מצבי ההגנה האפשריים.	find_defence_option(self,player,board_none_grafic
פעולה מקבלת לוח לא גרפי ריק,מספר שחקן, משתנה בוליאני (אם זה הגנה או התקפה) ומערך אופציות של הגנה או התקפה.	פעולה רקורסיבית המשחקת את המשחק עד הסוף ממהלך ספציפי ומחפשת ניצחון. פעולה המחזירה אמת אם נמצא ניצחון אחרת שקר.	recurse_find_card(self, options, player, flag,board_none_grafic)
פעולה מקבלת קלף כלשהוא.	פעולה המחפשת את האינדקס של קלף מסויים ברשימת שחקן מחשב. פעולה המחזירה אינדקס אם נמצא הקלף אחרת שקר.	find_this_card(self,card)
	פעולה המחפשת את כל סוגי הקלפים שנצאים בקרב. פעולה המחזירה את רשימת הקלפים בקרב.	list_battle_attack(self)
פעולה המקבלת מספר שחקן וזמן.	פעולה המבצעת התקפה או לא מבצעת התקפה ומעבירה מיד תור.	attack_computer(self, player, dt)
	פעולה הבודקת האם אין מקום יותר להתקפה(האם המקום מלא). פעולה מחזירה אמת אם מלא שקר אחרת.	is_full(self)
	פעולה מנקדת לפי מצב המשחק(כמות קלפים ברשימה). פעולה מחזירה 8 אם יש יותר מ-15 קלפים 4 אם יש בין 5 ל-	list_end(self)

	15 ו-0 אם יש פחות מחמישה.	
פעולה המקבלת קלף כלשהוא ורשימה המכילה את הקלפים שנמצאים בקרב.	פעולה המחליטה האם הקלף יכול להיות חלק מההתקפה. פעולה המחזירה אמת אם יש קלף ברשימה עם אותו מספר כמו שלקלף שקיבלנו אחרת שקר.	check_can_be_in(self,battle_list,card)
	פעולה הבודקת אם השחקן האנושי איבד קלף חזק. פעולה מחזירה אמת אם יש קוזר שהגנו איתו או מספר גבוה מנסיך אחרת שקר.	lost_good_one(self)
פעולה מקבלת מספר שחקן ורשימת קלפים שבקרב.	פעולה המחליטה באמצעות איזה קלף להתקיף אם מדובר על התחלת המשחק יש לה טקטיקה מסויימת אם כבר אין קלפים בחפישה באמצעות brutfource. פעולה מחזירה אינדקס של קלף ברשימת השחקן אם מצאה אחרת שקר.	attack_decision(self, battle_list,player)
	פעולה בודקת האם הלוח ריק לחלוטין. פעולה מחזירה אמת אם אין קלף על הלוח שקר אחרת.	is_empty(self)
פעולה מקבלת מערך דו מימדי עם קלפים.	פעולה מדפיסה כל מערך דו מימדי המכיל קלפים.	print_board(self,matrix)
	פעולה הבודקת אם יש מצב ניצחון. אם למישהוא אין קלפים כשהחפישה ריקה הפעולה תחזיר אמת אחרת שקר.	end_game_check(self)

3.2 מחלקת Card

מחלקה זו אחראית על יצור קלף



הסבר על תכונות:

הסבר	שם התכונה
המספר (הערך) של הקלף	Number
סוג הצורה של הקלף	Shape
האם הקלף קוזר או לא (משתנה בוליאני)	flag quzar
כתובת של תמונה של הקלף	source
משתנה שמגדיר מה מראש יכול להיגרר על ידי האדם שמשחק	can_be_dragged
משתנה שקובע שקרה שהקלף רשאי להיגרר	to_move
הדירוג של הקלף על פי מספרו והאם הוא קוזר.	Score
מחלקת המשחק על מנת לקבל נתונים על קלפים אחרים ולבצע בדיקות.	Board

הסבר על פעולות:

טענת יציאה	טענת כניסה	הסבר	שם הפעולות
	פעולה מקבלת מיקום לחיצה של עכבר.	פעולה בודקת אם הקלף נלחץ ומעדכנת	on_touch_down(self, touch)
	פעולה מקבלת מיקום לחיצה של עכבר.	פעולה המעדכנת את מיקום הקלף כתלות בתזוזה	on_touch_move(self, touch)
	פעולה מקבלת מיקום לחיצה של עכבר.	פעולה שמטרתה לבדוק האם הקלף הגיע לקרב ואם הגיע מטרה להפסיק את גרירתו	on_touch_up(self, touch)
אם הקלף נמצא בתחום מלבן קרב והוא מתאים להיות שם תחזיר את האינדקס שלו אחרת שקר.		פעולה שמטרתה לבדוק האם הקלף נמצא בתחומי הקרב והאם הוא עומד בתנאים.	on_place(self)
הפעולה תחזיר את האינדקס של הקלף שצריך לחפש.		פעולה שמוצאת את האינדקס של הקלף ברשימת השחקן.	find_card(self)
פעולה בודקת האם הקלף חזק יותר מהקלף שהתקבל. אם כן תחזיר אמת אחרת שקר.	פעולה מקבלת קלף כלשהוא.	פעולה בודקת איזה קלף חזק יותר.	check_bigger(self, card)
פעולה המחזירה אמת אם לקלף יש קלף בקרב עם אותו מספר כמו לו אחרת שקר.		פעולה הבודקת האם לקלף יש קלף נוסף בעל אותו מספר בקרב והוא רשאי לקחת חלק בהתקפה.	can_be_on_board(self)

פעולה תחזיר קלף לא גרפי עם אותם נתונים כמו של הרגיל.		פעולה מייצרת קלף ללא גרפיקה בעל אותם נתונים כמו של הקלף הספציפי.	create_card(self)

3.3 מחלקת reg_card

מחלקה זו משמשת את AI ואחראית לשמש קלף לנתונים בלבד

reg_card
Number Shape flag quzar score
check_bigger(self, card)

הסבר על תכונות:

הסבר	שם התכונה
המספר (הערך) של הקלף	Number
סוג הצורה של הקלף	Shape
האם הקלף קוזר או לו (משתנה בוליאני)	flag quzar
הדירוג של הקלף על פי מספרו והאם הוא קוזר.	Score

3.4 מחלקת board_for_AI

מחלקה זו משמשת את AI ואחראית לשמש לוח לנתונים בלבד

board_for_AI
תכונות list_of_players battle_list
פעולות __init__(**kwargs) find_empty_place2(self) take_cards_from_board(self, player) find_need_defence(self) list_battle_attack2(self) is_full(self) is_empty(self)

הסבר על תכונות:

הסבר	שם תכונה
מערך שמכיל את אזור הקרב. כלומר, מערך של שישה מקומות שמכיל רשימה שגודלה בין 0-2(כתלות במספר הקלפים). זה ייצוג של מהלכי המשחק בין השחקנים.	battle_list
מערך שמכיל את רשימות השחקנים. כל תא הוא רשימה בפני עצמו המכילה קלפים לא גרפיים.	list_of_players

הסבר על פעולות :

טענת יציאה	טענת כניסה	הסבר	שם הפעולות
		פעולה בונה של מחלקת none_grafic_board (הלוח ללא גרפיקה)	__init__(**kwargs)
	פעולה המקבלת מספר שחקן.	פעולה שלוקחת קלפים מהקופה ברגע שלשחקן יש פחות משישה קלפים.	take_cards_from_board(self, player)
פעולה מחזירה מספר אם נמצא קלף כזה אחרת שקר.		פעולה המחפשת קלף שלא מוגן.	find_need_defence(self)
פעולה מחזירה מספר אם נמצא מקום כזה אחרת שקר.		פעולה המחפשת מקום ריק ללא קלפים.	find_empty_place2(self)
פעולה המחזירה את רשימת הקלפים בקרב.		פעולה המחפשת את כל סוגי הקלפים שנצאים בקרב.	list_battle_attack2(self)
פעולה מחזירה אמת אם מלא שקר אחרת.		פעולה הבודקת האם אין מקום יותר להתקפה(האם המקום מלא).	is_full(self)
פעולה מחזירה אמת אם אין קלף על הלוח שקר אחרת.		פעולה בודקת האם הלוח ריק לחלוטין.	is_empty(self)

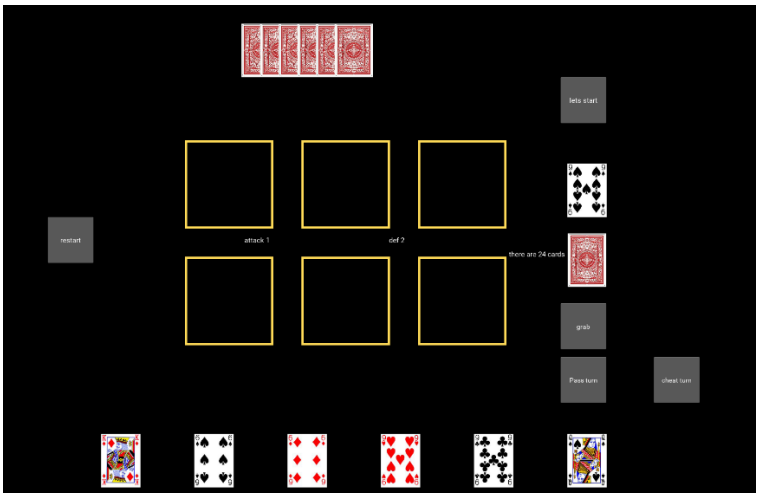
3.5 תיאור מימוש מבנה הנתונים בקוד

השתמשתי בקוד בסוג אחד של מבני נתונים: רשימות. השתמשתי ברשימות מהסיבות הבאות:

1 . על מנת להחזיק את הייצוג הגרפי של רשימות הקלפים של השחקנים, מקום המשחק (הקרב), קופת הקלפים ועוד מספר רשימות לייצוג גרפי של הקלפים. השימוש ברשימות במקרה זה נבע מהקלות בה ניתן לעבוד איתן, בנוסף לעובדה שהן שומרות על סדר האיברים שהוכנסו אליהן.

2 .על מנת להגיע בקלות לאינדקס של הקלפים ולממש צורת חיפוש קלה ויעילה רשימות הייתה הבחירה הטובה ביותר למבני הנתונים.

לדוגמא: כפי שניתן לראות בייצוג הגרפי רשימה אחת של קלפים גרפיים ותמונות של קלפים.



ובדוגמא הבאה ניתן לראות כיצד רשימה עוזרת לבנות את קופת הקלפים בצורה נוחה.

```
                                width=1000))
def creat_list_cards(self, list_cards):#Function that get list ad build in this list the cards of the game
    shapes = ['leaf', 'heart', 'diamond', 'clubs']
    num = 0
    for i in range(6, 15):#15
        for j in shapes:
            list_cards.append(Card(i, j, False, self.sources[num],self,))
            num += 1
```

3.6 חוקי ניהול המשחק

ניהול המשחק מתקיים כאשר כל שחקן על פי תורו מבצע פעולה התקפתית או הגנתית. אם השחקן מתקיף הוא מעביר את המהלך ליריב שתורו להגן. על היריב שתי אופציות להגן ושוב להעביר מהלך לשחקן התוקף או לקחת את הקלפים שעל הלוח והשחקן התוקף ממשיך לתקוף בתור חדש (לוח חדש). אם החליט להענות יהיה עוד פעם תור התוקף לבצע מהלך התקפי (אם ירצה) וכן הלאה ימשך עד שהמגן יקח את הקלפים או שהגיעו למצב של שישה התקפות מוגנות (אין מקום להתקפות) ואז הלוח מתרוקן ותור השחקן המגן להתקיף. ככה משחקים ומשלימים לשישה קלפים כל תור עד שנגמרת החפישה ומשחקים עד סיום הקלפים מהיד.

4. הסבר האלגוריתם

4.1 הסבר תיאורטי

המשחק מחולק לשני חלקים:

החלק הראשוני כאשר עוד יש חפישה והחלק השני כאשר נגמרת החפישה ועל השחקן לרוקן את חפישת הקלפים מידו לפני היריב.

בחלק הראשון נשתמש בטקטיקה כללית שבאמצעותה נוכל להביא את המחשב לחפישה הטובה ביותר שיוכל להגיע אליה בסיום המשחק ושם המהלכים כבר הם מהלכים שמובנים לניצחון. כלומר, המחשב לא בהכרח מבזבז קלפים חזקים, הוא יודע מתי לענות או לו לענות למהלך התקפי ומתי להוסיף קלפים. בעזרת חשיבה זו מטרת המחשב לחשוב כמה צעדים קדימה על מנת שבסוף קופת הקלפים המחשב יוכל למצוא מהלך שיוביל לניצחון ע"י שיטת Brute Force.

איך המחשב יודע מה לעשות ומה לו לעשות?

למחשב ציון התחלתי המוגדר כמאה כעת יתחיל להוריד את ציון זה בגלל גורמים חשובים והם:

- קלפים בעלי אותה צורה שיש בחפישה (רצון לשמר כמה סוגים של קלפים בחפישה)- פעולת def how_much_shapes(self,card). יורד מהניקוד 20 אם הקלף הוא קוזר, יורד 10 אם אין עוד סוג כזה של קלף, 5 אם יש אחד נוסף מאותה צורה, 0 אם יש שניים ואם יש יותר יוספו 5 נקודות.
- הניקוד שניתן לקלפים ככל שהקלף חזק יותר כך ניקודו גבוה יותר- הניקוד ניתן על ידי פעולת def scores(self) פעולה מחלקת ניקוד לפי ערך הקלף אך אם זה קוזר הוא יקבל עוד 9 נקודות(כדי לוודא שלקוזר הניקוד הכי גבוה).בנוסף, בגלל שיש חשיבות לערך הקלף בהורדת הנקודות יגדל ערכו פי 2.
- ידע קודם שמבוסס על זיכרון, המחשב זוכר אילו קלפים לקח השחקן אלו חלשים אצלו (אם לקח מסוג מסויים זה מראה אולי על חיסרון בו) וכמובן

בזבוז קלפים חזקים ילקח בחשבון.לדוגמא, פעולת def will_attack_me(self,card), def special_condition(self,card) . יורדו 25 נקודות אם לשחקן יש קלף מאותו סוג (אותו מספר) ויש סיכון גבוה שתהיה התקפה נוספת. פעולה עם תנאי מיוחד זה פעולה שקובעת שקוזרים לא יבוזבזו עד שנשאר פחות מ-5 קלפים.def lost_good_one(self) היא פעולה נוסף שקובעת שאם השחקן הוציא קלף גבוה(מלכה, מלך, אס או קוזר) המחשב לא ישיב ויקח.

- מצב החפישה (לקראת סיום,רק בהתחלה ועוד)-פעולת def list_end(self). הניקוד יורד 8 אם יש מעל 15 קלפים (התחלת המשחק), בין 5 ל-15 יורד 4(אמצע משחק) אחרת לא יורד בכלל(סוף משחק).
- לבסוף המחשב מקבל ציון ואם הציון גדול מ-60, פעולת המהלך הנמצאת גם בפעולת מציאת הגנה וגם ובמציאת התקפה תתרחש, אחרת המחשב אם זה הגנה ייקח את הקלפים ואם זו התקפה יועבר התור לשחקן האנושי.

כל הדברים האלה מאפשרים למחשב להחליט החלטות חשובות ובו זמנית מאפשרים לו לחשוב על המצב הסופי של המשחק, שבו בזכות הזיכרון של המחשב המחשב יוכל לדעת אם ניצחון אפשרי. אם זאת, ללא השימוש בטקטיקה גם הרצה באמצעות שימוש Brute Force לא מבטיח ניצחון לכן הטקטיקה מיועדת לחשוב כמה צעדים קדימה על מנת להבטיח ניצחון.

בחלק השני, כשהחפישה נגמרה, מסתיים השימוש בטקטיקה. למה? למחשב זיכרון חזק וכמו שהוא יודע להריץ מהלכים קדימה הוא זוכר את הקלפים שנשרפו. לכן, כל החפישה פחות הקלפים שנשרפו פחות הקלפים בידי המחשב = הקלפים ביד השחקן בסיום קופת הקלפים. ולכן מפה אפשר לתכנן מהלכים עד סוף המשחק בעזרת:

אלגוריתם(Brute Force)

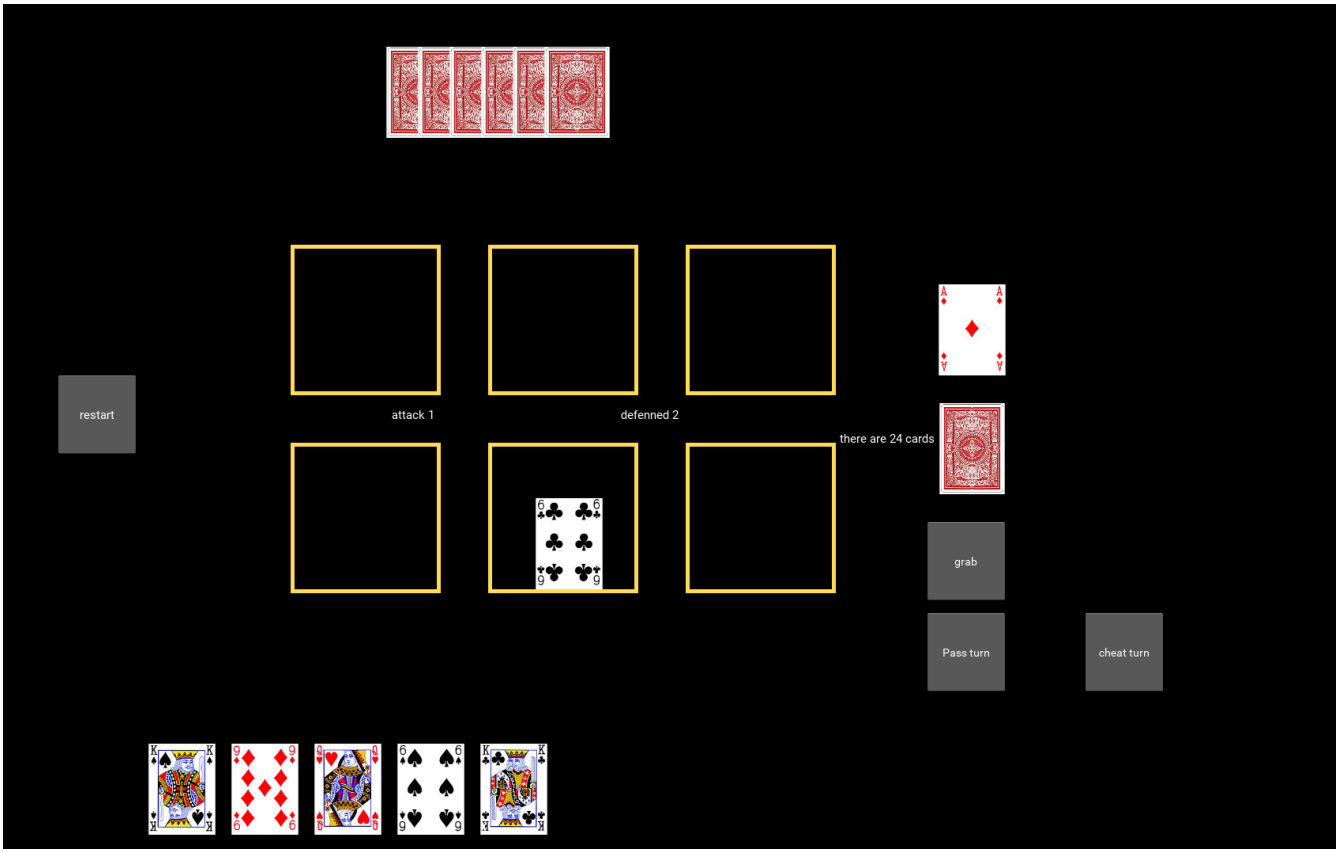
- כוח גס (Brute Force). הוא חיפוש ממצא של כל טווח המהלכים האפשריים עד למציאת המהלך שאיתו נעשה שימוש.
עם מספיק זמן וכוח חישוב, שיטה ברוטלית וישירה זו תביא בסופו של דבר לתוצאות.
- מאחר שיש רק מספר מוגבל של מהלכים (מספר קלפים מוגבל), ניתן לבחון כל מהלך בתורו בדרך Brute Force.
- הרצת המהלכים נמשכת עד השגת תנאי הניצחון.

- בגלל העובדה שמספר הקלפים מאוד קטן עם סיום המשחק הרצת המשחק עד הסוף אינה עבודה ממושכת למחשב והוא עושה זאת במהירות.
- הפעולה שממשת אלגוריתם זה היא def recurse_find_card(self, options, player, flag,board_none_grafic)

4.2 הצגת הפלט

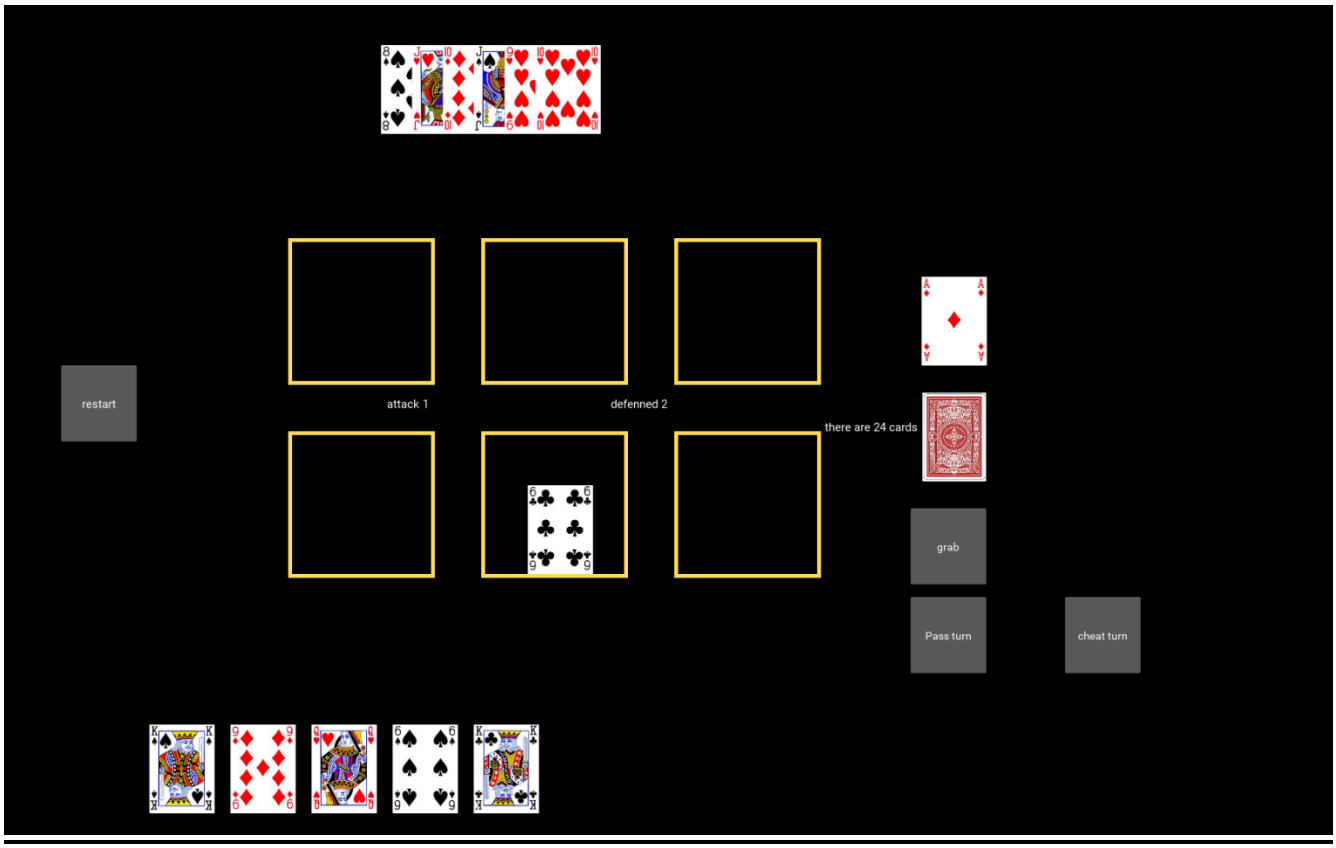
<u>פלט מהלך שחקן :</u>

התקפה של 6 תלתן



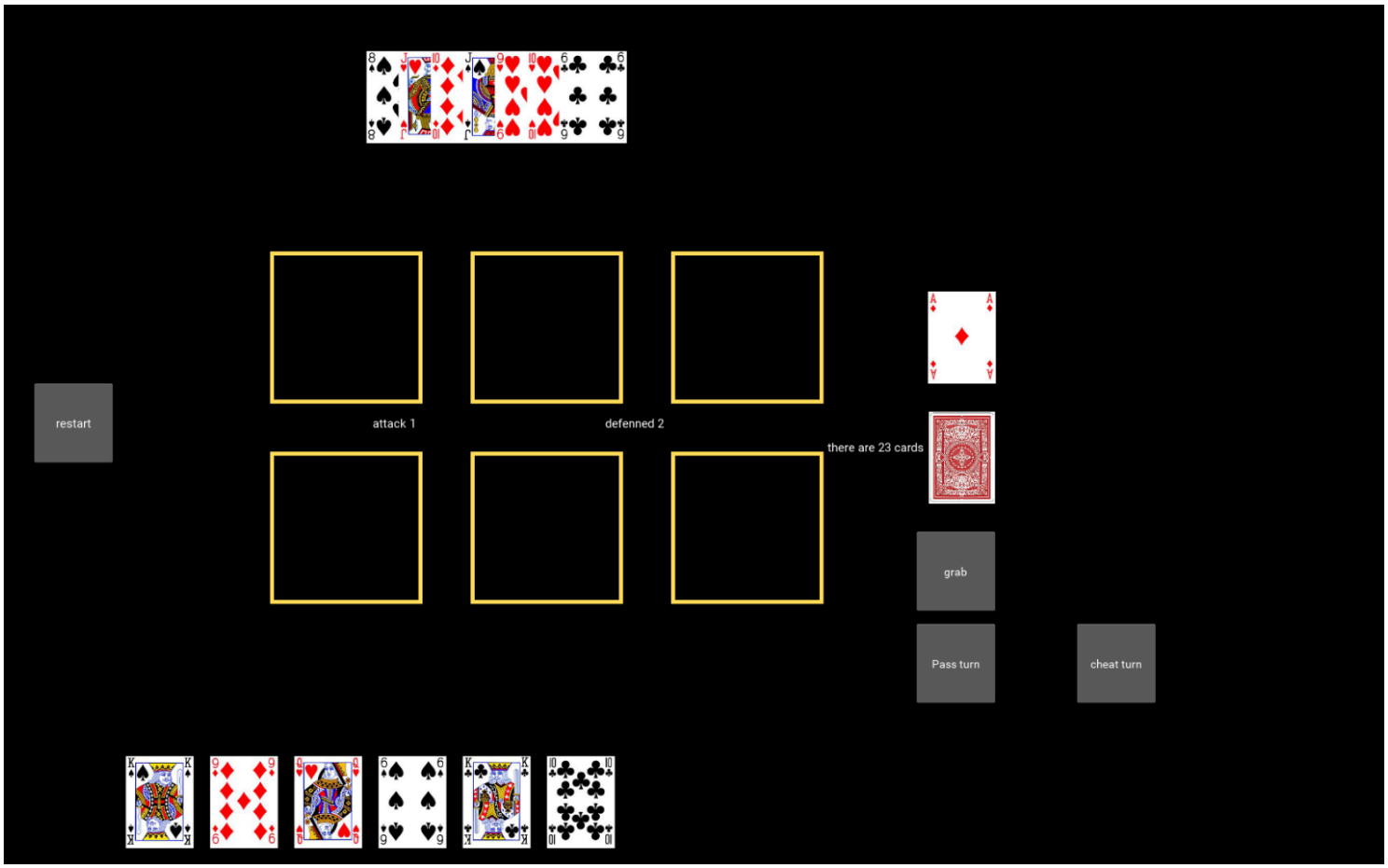
מצב 2 :

ניתן לראות את הקלפים כדי לנסות לדעת איזה החלטה יקבל המחשב.



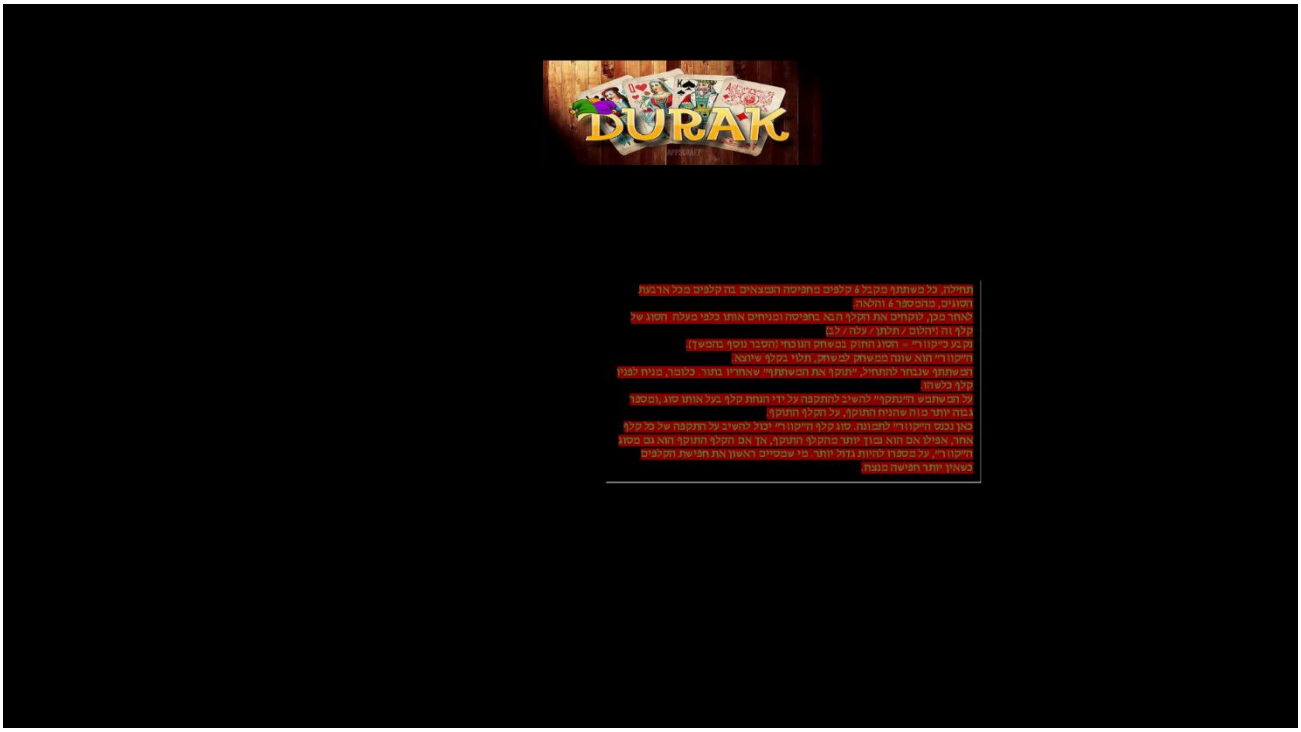
מצב 3 :

המחשב צפוי לקחת



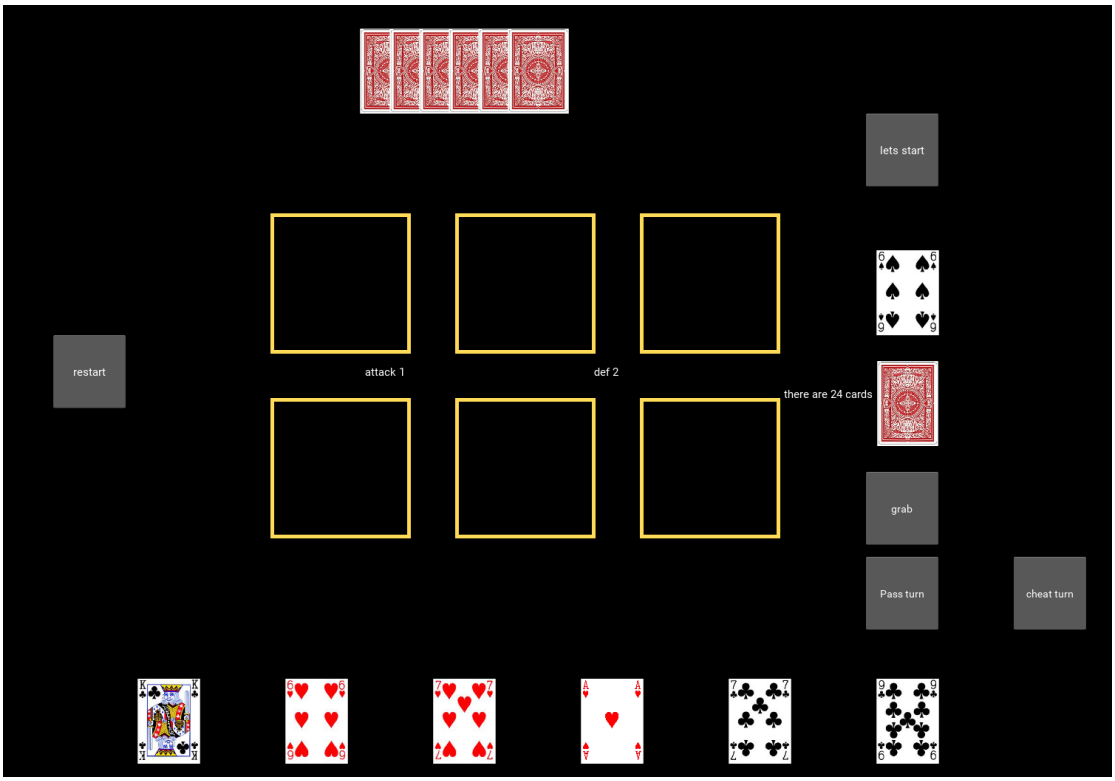
5. ממשק למשתמש

<u>מסך פתיחה</u>



לחיצה על התמונה תעלים את ההוראות ותמונה ותעביר למסך המשחק.

מסך המשחק:



ממשק העכבר:

המשחק פועל באמצעות העכבר בלבד כדי ללחוץ על הלחצנים ולגרור קלפים. אין שימוש במקלדת.

6. CODE מתועד של המערכת – הדפסת התכנית עם התיעוד

```
from kivy.app import App
from kivy.uix.button import Button
from kivy.uix.label import Label
from kivy.uix.floatlayout import Layout
from kivy.uix.image import Image
import random
from kivy.graphics import *
from kivy.animation import Animation
from kivy.core.window import Window
Window.fullscreen = 'auto'
from kivy.clock import Clock
from functools import partial
import copy
class Card(Image): #obect of card for list cards of the game
    def __init__(self,number,shape,flag_quzar,source,Board): #Build a card
that consist of shape,number ,is he quzar,source of image ,score for the Ai
and 2 values that will deide when to be dragged
        super(Card, self).__init__()
        self.shape = shape
        self.number = number
        self.flag_quzar = flag_quzar
        self.source = source
        self.Board = Board
        self.to_move = False#State if you can start move the card
        self.can_be_dragged = True#State if you can start drag the card
        self.score=0
    def on_touch_down(self,touch):  # Check if there is a card with his
shape in the battle and check on_place conditions and then u can move
him
        if self.collide_point(touch.x, touch.y) and self.can_be_dragged ==
True:
            self.to_move = True
    def on_touch_move(self, touch):  # Update position by x and y ebery
move of the card(input mouse)
        if self.to_move:
            self.x = touch.x - self.width / 2
            self.y = touch.y - self.height / 2
    def on_touch_up(self, touch):  # Unable to move when realesed, if the
conditions are good and you can put the card put him on list battle
        self.to_move = False
        if self.on_place() != False :
            self.can_be_dragged = False
    def on_place(self):  # Check if the card in batlle position and if he even
can return place that can be put
        for i in range(6):
```

```
        if (self.Board.pos_rectangle_list[i][0] < self.x <
self.Board.pos_rectangle_list[i][0] + 180 and
                    self.Board.pos_rectangle_list[i][1] < self.y <
self.Board.pos_rectangle_list[i][1] + 180):
            if (len(self.Board.cards_battle[i]) == 0 and
self.Board.turn_attack==0 and self.can_be_on_board()):
                self.Board.remove_card(self)
                self.Board.remove_from_unkown(self)
                self.can_be_dragged = False
                self.Board.cards_battle[i].append(self)
                del self.Board.list_players[0][self.find_card()]
                self.x = self.Board.pos_rectangle_list[i][0] + 40
                self.y = self.Board.pos_rectangle_list[i][1]
                self.Board.stuck_the_other()
                self.Board.organize_the_cards(0)
                return i
            elif len(self.Board.cards_battle[i]) == 1 and self.check_bigger(
self.Board.cards_battle[i][0])==True:
                self.Board.remove_card(self)
                self.Board.remove_from_unkown(self)
                self.can_be_dragged = False
                self.Board.cards_battle[i].append(self)
                del self.Board.list_players[0][self.find_card()]
                self.x = self.Board.pos_rectangle_list[i][0] + 40
                self.y = self.Board.pos_rectangle_list[i][1] + 30
                self.Board.organize_the_cards(0)
                return i
    return False
  def find_card(self):  # find which card was placed return which card it is
    for i in range(len(self.Board.list_players[0])):
        if (self.number == self.Board.list_players[0][i].number and
self.shape == self.Board.list_players[0][
            i].shape):
            return i
  def check_bigger(self, card):  # function check if the cardscan
beat(answer) the other return true if it can false for any other option
    if (card.flag_quzar == False and self.flag_quzar==False):
        if (self.shape == card.shape and self.number > card.number):
            return True
        else:
            return False
    elif(card.flag_quzar == False and self.flag_quzar==True):
        return True
    elif(self.flag_quzar == True and card.flag_quzar==True):
            if (self.number > card.number):
                return True
            else:
                return False
  def create_card(self): #Return a card but without graffics with same
values
```

```
        return reg_card(self.number,self.shape,self.flag_quzar,self.score)
    def can_be_on_board(self): #Check if the card we attacked with is good
for attacking return true if it can ele false
        counter=0
        for i in self.Board.cards_battle:
            for j in i:
                if j.number==self.number:
                    return True
                counter+=1
        if counter==0:
            return True
        return False
class reg_card(): #Card but without graffics
    def __init__(self, number, shape, flag_quzar,score):
        self.shape = shape
        self.number = number
        self.flag_quzar = flag_quzar
        self.score=score

    def check_bigger(self, card):  # function of card without graffic check if
the cardscan beat(answer) the other return true if it can false for any other
option
        if (card.flag_quzar == False and self.flag_quzar == False):
            if (self.shape == card.shape and self.number > card.number):
                return True
            else:
                return False
        elif (card.flag_quzar == False and self.flag_quzar == True):
            return True
        elif (self.flag_quzar == True and card.flag_quzar == True):
            if (self.number > card.number):
                return True
            else:
                return False
class board_for_AI(): #Board of the game but without graffics
    def __init__(self,): #Build empty list that will consist the battle
        self.battle_list=[[] for i in xrange(6)]
        self.list_of_players=[[] for i in xrange(2)]
    def find_empty_place2(self): #For the attck find empty place other return
False
        for i in range(6):
            if len(self.battle_list[i])==0:
                return i
        return False
    def find_need_defence(self): #Find in the list battle
        for i in range(6):
            if len(self.battle_list[i])==1:
                return i
        return False
    def list_battle_attack2(self): #Return the list of all attacks that
```

```
        list_battle = []
        for i in range(6):
            if len(self.battle_list[i]) == 1:
                list_battle.append(self.battle_list[i][0])
            elif len(self.battle_list[i]) == 2:
                list_battle.append(self.battle_list[i][0])
                list_battle.append(self.battle_list[i][1])
        return list_battle
    def is_full(self): #Check if there were six attack and there is no place for
more attacks return true if board is full else, false
        for i in range(6):
            if len(self.battle_list[i])==0:
                return False
        return True

    def is_empty(
            self):  # Check if there were six attack and there is no place for more
attacks return true if board is full else, false
        for i in range(6):
            if len(self.battle_list[i]) != 0:
                return False
        return True
    def take_cards_from_board(self,player): #When player can not counter
the attack he will take alll card that on board
        for i in range(6):
            for j in self.battle_list[i]:
                self.list_of_players[player].append(j)
        self.battle_list=[[] for j in xrange(6)]
# class that contains all the data structures of the game
class Board(Layout):
    # Builds the board
    def __init__(self, **kwargs): #Function that build the board of the game.
The board consist of the whole data base of the game and the graffic o the
game
        super(Board, self).__init__(**kwargs)
        self.blank1 = "blank1.jpg"
        self.turn_attack = 0  # Check who is attacking now
        self.turn_deffedned = 1  # check who is deffend now
        self.sources = ['leaf6.jpg', 'heart6.jpg',
                    'dimond6.jpg', 'tiltan6.jpg', 'leaf7.jpg', 'heart7.jpg',
'dimond7.jpg', 'tiltan7.jpg',
                    'leaf8.jpg', 'heart8.jpg',
                    'dimond8.jpg', 'tiltan8.jpg', 'leaf9.jpg', 'heart9.jpg',
'dimond9.jpg', 'tiltan9.jpg',
                    'leaf10.jpg', 'heart10.jpg',
                    'dimond10.jpg', 'tiltan10.jpg', 'leafJ.jpg', 'heartJ.jpg',
'dimondJ.jpg', 'tiltanJ.jpg',
                    'leafQ.jpg', 'heartQ.jpg',
                    'dimondQ.jpg', 'tiltanQ.jpg', 'leafK.jpg', 'heartK.jpg',
'dimondK.jpg', 'tiltanK.jpg',
```

```
                'leafA.jpg'
        ,  'heartA.jpg', 'dimondA.jpg', 'tiltanA.jpg']

        self.list_cards = []
        self.pos_rectangle_list = list()  # Position of the battle's rektangles
        self.cards_battle = [[], [], [], [], [], []]  # 2 D list that keep the card of the
current battle
        self.list_players = [[], []]  # 2D list that contain list of cards for every
player
        self.computer_pos_card = []
        self.list_exist_cards = []
        self.list_known = []
        self.start_button = Button(font_size=14, pos=(800, 800), height=150,
                                   width=400,background_normal
='image.jpg',text_color=(0,0,0,1) )
        self.add_widget(self.start_button)
        self.start_button.bind(on_press=self.start)
        self.add_widget(Image(source='instr.png',pos=(700,0),height=1000,
                                   width=1000))
    def creat_list_cards(self):#Function that get list ad build in this list the
cards of the game
        shapes = ['leaf', 'heart', 'diamond', 'clubs']
        num = 0
        for i in range(6, 15):#15
            for j in shapes:
                self.list_cards.append(Card(i, j, False, self.sources[num],self,))
                num += 1

    def restart_game(self):  # function that restart the original board's class
        self.canvas.clear()
        self.clear_widgets()
        self.__init__()


    def start_game(self):#Function that updates the data of the structures
and initializes the graphics
        self.cheat = Button(text='cheat turn', font_size=14, pos=(1400, 200),
height=100,
                            width=100)
        self.cheat_flag = False
        self.passed = Button(text='Pass turn', font_size=14, pos=(1200, 200),
height=100,
                             width=100)  # Button that when is pressed it is mean the
player end his turn
        self.cheat.bind(on_press=self.cheated)
        self.grabbed = Button(text='grab', font_size=14, pos=(1200, 315),
height=100,
                              width=100)  # Button that when is pressed it is mean the
player i grabbed the card
        self.passed.bind(on_press=self.pass_turn)
```

```
        self.grabbed.bind(on_press=self.grabbed_2)
        self.add_widget(self.grabbed)
        self.add_widget(self.passed)
        self.add_widget(self.cheat)
        self.creat_list_cards()
        self.scores()
        self.change_size_of_cards()
        self.buttonrest = Button(text="restart", pos=(100, 500))
        self.buttonrest.bind(on_press=lambda x: self.restart_game())
        self.add_widget(self.buttonrest)
        random.shuffle(self.list_cards)
        for i in range(len(self.list_cards)):
            self.list_exist_cards.append(
                reg_card(self.list_cards[i].number, self.list_cards[i].shape,
self.list_cards[i].flag_quzar,self.list_cards[i].score))
        self.give_cards()
        self.quzar = self.list_cards.pop()
        self.quzar.flag_quzar = True
        self.quzar.can_be_dragged = False
        self.quzar.x = 1200
        self.quzar.y = 600
        self.list_game = Image(source=self.blank1, x=1200, y=450,
width=115, height=115)
        self.add_widget(self.quzar)
        self.list_cards.append(self.quzar)
        self.show_num_cards = Label(text='there are ' +
str(len(self.list_cards)) + ' cards', x=1100, y=470)
        self.add_widget(self.show_num_cards)
        self.add_widget(self.list_game)
        self.change_quzar()
        self.prepre_battle()
        self.cards_on_board()
        self.stuck_cards()
        self.lab1 = Label(text='attack' + ' ' + str(self.turn_attack + 1), x=500,
y=500)
        self.add_widget(self.lab1)
        self.lab2 = Label(text='def' + ' ' + str(self.turn_deffedned + 1), x=800,
y=500)
        self.add_widget(self.lab2)
        self.lets_start_button=Button(text='lets
start',pos=(1200,800),on_press=lambda x: self.who_will_start())
        self.add_widget(self.lets_start_button)


    def scores(self): #Functions that determines points to every card
        for i in range(len(self.list_cards)):
            self.list_cards[i].score =self.list_cards[i].number
            if self.list_cards[i].flag_quzar == True:
                self.list_cards[i].score += 9
    def remove_card(self,card): #Function that changes the reminimg list of
```

cards which are not owned by the players.

```
        # This list at the end of the game will turn to the list of the player
(when there is none list of cards to take from)
        for i in range(len(self.list_exist_cards)):
            if card.number==self.list_exist_cards[i].number and
card.shape==self.list_exist_cards[i].shape:
                del self.list_exist_cards[i]
                break
    def start(self, v):  #Function that transfer the initial screen to the main
screen of the game
        self.clear_widgets()
        self.start_game()
    def cheated(self,v):  #Function  will be used to check the player cards and
to the future movements
        if self.cheat_flag==False:
            for i in range(len(self.computer_pos_card)):
                self.list_players[1][i].pos=self.computer_pos_card[i].pos
                self.remove_widget(self.computer_pos_card[i])
                self.add_widget( self.list_players[1][i])
        else:
            for i in range(len(self.computer_pos_card)):
                self.list_players[1][i].pos = self.computer_pos_card[i].pos
                self.remove_widget(self.list_players[1][i])
                self.add_widget(self.computer_pos_card[i])
        self.cheat_flag=not self.cheat_flag
    def grabbed_2(self, v):  #Function that will be called when the button
grab will be pressed.
        #This function will be enabled only when the player needs to take
cards from the board
        if  self.end_game_check()==False:
            if self.turn_deffedned == 0 and type(self.find_not_deffedned()) ==
int:
                self.take_cards_from_board(0)
                self.turn_attack=1
                self.turn_deffedned=0
                self.lab1.text = 'attack' + ' ' + str(self.turn_attack + 1)
                self.lab2.text = text = 'defenned' + ' ' + str(self.turn_deffedned +
1)
                Clock.schedule_once(partial(self.attack_computer,
self.turn_attack), 1.5)
                self.stuck_cards()
                if len(self.list_cards) != 0:
                    self.take_cards_from_list(1)
    def pass_turn(self, v):  #Function that will be called when the button pass
will be pressed.
        #The function will be called only when a turn is passed to the other
player
        if self.end_game_check()==False:
            if self.turn_attack == 0:
                if type(self.find_not_deffedned())==int:
```

```
            Clock.schedule_once(partial(self.deffened_computer, 1), 1.5)
            self.stuck_the_other()
        else:
            for i in range(6):
                for j in range(len(self.cards_battle[i])):
                    self.remove_widget(self.cards_battle[i][0])
                    del self.cards_battle[i][0]
            self.cards_battle=[[] for i in xrange(6) ]
            self.turn_attack=1
            self.turn_deffedned=0
            self.lab1.text = 'attack' + ' ' + str(self.turn_attack + 1)
            self.lab2.text = text = 'defenned' + ' ' + str(self.turn_deffedned
+ 1)
            if len(self.list_cards) != 0:
                self.take_cards_from_list(0)
            if len(self.list_cards) != 0:
                self.take_cards_from_list(1)
            Clock.schedule_once(partial(self.attack_computer,
self.turn_attack), 1.5)
    elif self.turn_deffedned == 0:
        Clock.schedule_once(partial(self.attack_computer, 1), 1.5)
def remove_from_unkown(self,card): #Function that removes cards,
that the computer remembers that the player had, when he put them on
board.
    for i in range(len(self.list_known)):
        if card.number == self.list_known[i].number and card.shape ==
self.list_known[i].shape:
            del self.list_known[i]
            break

def change_size_of_cards(self): #Function that changes the size of the
cards to the correct size
    for i in range(len(self.list_cards)):
        self.list_cards[i].width = 115
        self.list_cards[i].height = 115

def give_cards(self): #Function that divides cards to each player
    for j in range(6):
        for i in range(len(self.list_players)):
            if i == 1:
                self.remove_card(self.list_cards[0])
            self.list_players[i].append(self.list_cards[0])
            del self.list_cards[0]
            if (i != 0):
                self.list_players[i][j].can_be_dragged = False
def change_quzar(self): # Function that replaces the flag to true for
every card that have the same shape as the quzar
    for j in range(6):
        for i in range(len(self.list_players)):
            if (self.list_players[i][j].shape == self.quzar.shape):
```

```
                self.list_players[i][j].flag_quzar = True
        for i in range(len(self.list_cards)):
            if (self.list_cards[i].shape == self.quzar.shape):
                self.list_cards[i].flag_quzar = True

    def prepre_battle(self): #Function that prepares the place where the
battle of the game will occur
        x = 400
        y = 580
        for i in range(0, 3):
            self.pos_rectangle_list.append((x, y))
            with self.canvas:
                Color(255 / 255.0, 219 / 255.0, 88 / 255.0)
                Rectangle(size=(190, 190), pos=(x - 5, y - 5))
                Color(0, 0, 0)
                Rectangle(pos=self.pos_rectangle_list[i], size=(180, 180))
            x += 250
        x = 400
        y = 330
        for i in range(3, 6, 1):
            self.pos_rectangle_list.append((x, y))
            with self.canvas:
                Color(255 / 255.0, 219 / 255.0, 88 / 255.0)
                Rectangle(size=(190, 190), pos=(x - 5, y - 5))
                Color(0, 0, 0)
                Rectangle(pos=self.pos_rectangle_list[i], size=(180, 180))
            x += 250

    def cards_on_board(self): #Function that adds graphically the cards to
the layout
        y = 20
        x = 200
        for i in range(6):  # Loop for first player
            self.list_players[0][i].x = x
            self.list_players[0][i].y = y
            self.add_widget(self.list_players[0][i])
            x += 200
        y = 900
        x = 500
        for i in range(6):  # Loop for second player
            card = Image(source=self.blank1, x=x, y=y, width=115,
height=115)
            self.add_widget(card)
            self.computer_pos_card.append(card)
            x += 40
    def who_will_start(self): #Function that determines who will start
according to which player have bigger quzar
        lowest_card_player = 15
        for i in range(len(self.list_players)):
            for j in range((len(self.list_players[i]))):
```

```
            if (self.list_players[i][j].flag_quzar == True and
self.list_players[i][
                j].number < lowest_card_player):
                if i == 0:
                    self.turn_attack = 0
                    self.turn_deffedned = 1
                    lowest_card_player = self.list_players[i][j].number

                elif i==1:
                    self.turn_attack = 1
                    self.turn_deffedned =0
                    lowest_card_player = self.list_players[i][j].number
        if (self.turn_attack == 0):
            self.free_cards()
        else:
            self.stuck_cards()
        self.lab1.text = 'attack' + ' ' + str(self.turn_attack + 1)
        self.lab2.text = text = 'defenned' + ' ' + str(self.turn_deffedned + 1)
        self.attack_computer(self.turn_attack, 1)
        self.remove_widget(self.lets_start_button)
    def stuck_cards(self):  #Function that does'nt let the player to move
cards
        for i in range(len(self.list_players[0])):
            self.list_players[0][i].can_be_dragged = False

    def free_cards(self):  #Function that let the player move cards
        for i in range(len(self.list_players[0])):
            self.list_players[0][i].can_be_dragged = True

    def stuck_the_other(self): #Function that let the player move the
matching cards
        if self.is_empty() == True:
            self.free_cards()
        else:
            for j in range(len(self.list_players[0])):
                for i in range(len(self.cards_battle)):
                    for k in range(len(self.cards_battle[i])):
                        if (self.list_players[0][j].number ==
self.cards_battle[i][k].number):
                            self.list_players[0][j].can_be_dragged = True

    def organize_the_cards(self, player): #Function that organizes after
every turn
        if player == 0:
            x = 200
            y = 20
            t =105
            for i in range(len(self.list_players[0])):
                self.list_players[0][i].pos = (x, y)
                x += t
```

```
        elif player == 1:
            y = 900
            x = 500
            for i in range(len(self.list_players[player])):  # Loop for second
player
                self.computer_pos_card[i].pos = (x, y)
                x += 40
    def take_cards_from_list(self,player): #Function that gives to every
player number of cards till he has 6 in a turn (till the package of cards ends)
        while (len(self.list_cards) != 0 and len(self.list_players[player]) < 6):
            if len(self.list_cards) == 1:

self.add_widget(Label(pos=(1200,550),text=self.list_cards[0].shape))
                self.remove_widget(self.list_cards[0])
            if player==1:
                self.remove_card(self.list_cards[0])
            self.list_players[player].append(self.list_cards[0])
            del self.list_cards[0]
            if player == 0:
                k=len(self.list_players[0])-1
                self.remove_widget(self.list_players[player][k])
                self.add_widget(self.list_players[player][k])
            else:
                if len(self.list_players[1])!=0:
                    self.computer_pos_card.append(
                    Image(source=self.blank1,
x=20,y=self.computer_pos_card[len(self.computer_pos_card) - 1].y + 40,
width=115,
                          height=115))
                else:
                    self.computer_pos_card.append(
                      Image(source=self.blank1, pos=(500,900),
                       width=115,
                       height=115))

self.add_widget(self.computer_pos_card[len(self.computer_pos_card) - 1])
        self.show_num_cards.text = 'there are ' + str(len(self.list_cards)) + '
cards'
        self.organize_the_cards(player)

    def take_cards_from_board(self, player): #Function that transfers the
vards to the player that did not succeed in his turn
        for i in range(len(self.cards_battle)):
            for j in range(len(self.cards_battle[i])):
                if player==0:
                    self.list_known.append(self.cards_battle[i][0])
                self.remove_widget(self.cards_battle[i][0])
                self.list_players[player].append(self.cards_battle[i][0])
                del self.cards_battle[i][0]
                if player == 0:
```

```
            self.add_widget(self.list_players[player][len(self.list_players[0])
- 1])
        else:
            self.computer_pos_card.append(Image(source=self.blank1,
x=20,
                    y=self.computer_pos_card[len(self.computer_pos_card) -
1].y + 40,
                    width=115, height=115))

self.add_widget(self.computer_pos_card[len(self.computer_pos_card) - 1])
    self.organize_the_cards(player)
    self.cards_battle = [[], [], [], [], [], []]

  def deffened_computer(self, player, dt): #Function that performs
defensive move for the computer
    if player == 0:
        self.free_cards()
        return 0
    t = self.find_not_deffedned()
    while (type(t) == int and self.is_full() == False):
        the_card = self.find_card_to_deffend(self.cards_battle[t][0], player)
        if (type)(the_card) == int:
            self.add_card_to_board(the_card, t, player, False)
            t = self.find_not_deffedned()
        else:
            self.take_cards_from_board(player)
            self.free_cards()
            self.cards_battle = [[], [], [], [], [], []]
            t = False
            if len(self.list_cards) != 0:
                self.take_cards_from_list(0)



  def find_not_deffedned(self): #Function that seraches in the list of battle
a card without defence, returns true if finds else false
    for i in range(6):
        if len(self.cards_battle[i]) == 1:
            return i
    return False

  def find_empty_place(self): #Function that seraches in the list of battle
an empty place , returns true if finds else false
    for i in range(6):
        if len(self.cards_battle[i]) == 0:
            return i
    return False
  def how_much_shapes(self,card): #Function that returns score to the
player according to how many cards are with same shape
    if card.shape==self.list_cards[-1].shape:
```

```
            return 20
        counter=0
        for i in self.list_players[1]:
            if i.shape==card.shape and(card.number!=i.number and
card.shape!=i.shape) :
                counter+=1
        if counter==0:
            return 10
        if counter==1:
            return 5
        if counter==2:
            return 0
        return -5
    def will_attack_me(self,card): #Function that searches if the human
player has same card as the computer, returns true if finds else false
        for i in self.list_known:
            if i.number==card.number:
                return True
        return False

    def add_card_to_board(self, card_place, rec_place, player,
                          flag):  # Function that gets card place in the list,the number
of recatngle,
        # players' number and flag: true for attack false or def and put the card
in the battle
        self.remove_widget(self.computer_pos_card[card_place])
        self.list_players[player][card_place].x =
self.computer_pos_card[card_place].x
        self.list_players[player][card_place].y =
self.computer_pos_card[card_place].y
        del self.computer_pos_card[card_place]
        self.add_widget(self.list_players[player][card_place])
        if flag == True:
            anim = Animation(x=self.pos_rectangle_list[rec_place][0] + 40,
y=self.pos_rectangle_list[rec_place][1])
            anim.start(self.list_players[player][card_place])
        else:
            anim = Animation(x=self.pos_rectangle_list[rec_place][0] + 40,
                             y=self.pos_rectangle_list[rec_place][1] + 40)
            anim.start(self.list_players[player][card_place])

self.cards_battle[rec_place].append(self.list_players[player][card_place])
        del self.list_players[player][card_place]
    def special_condition(self,card): #Function that get a card
        #If it is a quzar and it is the end of the game(less than 5 cards in the
pack) return true else false
        if card.flag_quzar==True and self.list_end()!=0:
            return True
        return False
    def find_card_to_deffend(self, card, player): #Function that searches
```

card to protect from attacking card.

#when there is pack of cards to take from it searches according tactics, when there isn't it searches by brutforce.

#Returns number of card if it finds it else false.

```
    if len(self.list_cards)!=0:
        options = []
        for i in range(len(self.list_players[player])):
            if self.list_players[player][i].check_bigger(card):
                options.append(reg_card(self.list_players[player][i].number,
self.list_players[player][i].shape,
                                self.list_players[player][i].flag_quzar,
                                self.list_players[player][i].score))
        if len(options) == 0:
            return False
        score_list = []
        for i in options:
            shapes=self.how_much_shapes(i)
            removal=0
            if self.will_attack_me(i)==True:
                removal=25
            score=100-i.score*2-shapes-removal-self.list_end()
            if score>50:
                score_list.append((score,i))

        max=0
        for j in range (len(score_list)):
            if max==0:
                if score_list[j][0] > max and
self.special_condition(score_list[j][1])==False :
                    max=score_list[j]
            elif score_list[j][0]>max[0] and
self.special_condition(score_list[j][1])==False :
                max=score_list[j]
        if max==0:
            return False
        elif max[0]>50:
            return self.find_this_card(max[1])
        return False
    else:
        board_for_requrse = board_for_AI()
        self.build_board(board_for_requrse)
        self.create_regu_list_players(board_for_requrse)
        options = self.find_defence_option(1, board_for_requrse)
        print options,'zzzzzzzzzzz'
        need_board_to_win = []
        if options!=None:
            for i in options:
                if self.recurse_find_card(self.find_attack_option(1, i), 0, True,
copy.deepcopy(i)) == True:
                    print 'got one'
```

```
                self.print_board(i.battle_list)
                need_board_to_win = i
                break
        print need_board_to_win,'oooooooooooo'
        the_card=''
        if need_board_to_win == []:
            return False

        for i in range(6):
            if len(need_board_to_win.battle_list[i])!=len(self.cards_battle[i]):
                the_card=i
                break
        print the_card,'jjjjjjj'
        b=self.find_this_card(need_board_to_win.battle_list[the_card][1])
        return b
    #from here this i the AI
    def build_board(self,board_none_grafic): #Function that builds board of
the game without graphics
        board_none_grafic.battle_list=[[] for i in xrange(6)]
        for i in range(6):
            for j in self.cards_battle[i]:
                board_none_grafic.battle_list[i].append(j.create_card())
    def create_regu_list_players(self,board_none_grafic): #Function that
builds lists of the players without graphics
        for i in range(2):
            for j in self.list_players[i]:
                board_none_grafic.list_of_players[i].append(j.create_card())
    def find_attack_option(self,player,board_none_grafic): #Function that
returns list that contains none graphics board
        #and in each one of them there is different attacking option
        counter=0
        options=[]
        battle_list=board_none_grafic.list_battle_attack2()
        for i in board_none_grafic.list_of_players[player]:
            for j in battle_list:
                if board_none_grafic.is_full():
                    return options
                elif i.number == j.number:
                    v=copy.deepcopy(board_none_grafic)
                    k=board_none_grafic.find_empty_place2()
                    v.battle_list[k].append(i)
                    del v.list_of_players[player][counter]
                    options.append(v)
                    break
            if len(battle_list)==0:
                v = copy.deepcopy(board_none_grafic)
                k = board_none_grafic.find_empty_place2()

                v.battle_list[k].append(i)
                del v.list_of_players[player][counter]
```

```
            options.append(v)
        counter+=1
    return options
  def find_defence_option(self, player,board_none_grafic): #Function
that returns list that contains none graphics board
    #and in each one of them there is different defence option
    counter=0
    options = []
    card_to_def=board_none_grafic.find_need_defence()
    if type (card_to_def)==int:
        for i in board_none_grafic.list_of_players[player]:
            if i.check_bigger(board_none_grafic.battle_list[card_to_def][0]):
                k=copy.deepcopy(board_none_grafic)
                k.battle_list[card_to_def].append(i)
                del k.list_of_players[player][counter]
                options.append(k)
                counter+=1
    return options
  def recurse_find_card(self, options, player, flag,board_none_grafic):
#Function that contains brutforce AI.
    #exit conditions are human player wins returns false, or computer
wins returns true.
    #If there is no condition for exit the function will continue the
brutforce AI by placing all options recursively
    if len(board_none_grafic.list_of_players[1])==0:
        return True
    if len(board_none_grafic.list_of_players[0]) == 0:
        return False
    for i in options:
        if player==1:
            player2=0
        else:
            player2=1
        if flag==True:
            return
self.recurse_find_card(self.find_defence_option(player2,i),player2,False,i)
        else:
            j=i.find_need_defence()
            if type(j)==int:
                return self.recurse_find_card(self.find_defence_option(player,
i), player, False,i)
            else:
                return self.recurse_find_card(self.find_attack_option(player2,
i), player2, True,i)
    if len(options)==0:
        if player == 1:
            player2 = 0
        else:
            player2 = 1
        if flag==True:
```

```
            board2=[[] for i in xrange(6)]
            board_none_grafic.battle_list=board2
            return
self.recurse_find_card(self.find_attack_option(player2,board_none_grafic
), player2, True,board_none_grafic)
        else:
            board_none_grafic.take_cards_from_board(player)
            return
self.recurse_find_card(self.find_attack_option(player2,board_none_grafic
), player2, True,board_none_grafic)

    #Until here
    def find_this_card(self,card): #Function that gets a card, the function
returns the index of the card if it exists, else false
        for i in range(len(self.list_players[1])):
            if card.number==self.list_players[1][i].number and
card.shape==self.list_players[1][i].shape:
                return i
        return False
    def list_battle_attack(self): #Function that returns all cards that the battle
consist of.
        list_battle = []
        for i in range(6):
            if len(self.cards_battle[i]) == 1:
                list_battle.append(self.cards_battle[i][0])
            elif len(self.cards_battle[i]) == 2:
                list_battle.append(self.cards_battle[i][0])
                list_battle.append(self.cards_battle[i][1])
        return list_battle
    def attack_computer(self, player, dt): #Function that returns cards for
attack or clears the board and move the turn
        if player == 0:
            if self.is_empty() == True:
                self.free_cards()
            else:
                self.stuck_the_other()
            return 0
        list_battle = []
        list_battle=self.list_battle_attack()
        b=self.attack_decision(list_battle,player)
        if type(b)==int :
            t=self.find_empty_place()
            self.add_card_to_board(b,t,1,True)
            Clock.schedule_once(partial(self.deffened_computer, 0), 1.5)
        elif b==False:
            if len(self.list_cards) != 0:
                self.take_cards_from_list(0)
            if len(self.list_cards) != 0:
              self.take_cards_from_list(1)
            for i in range(6):
```

```
            for j in range(len(self.cards_battle[i])):
                self.remove_widget(self.cards_battle[i][0])
                del self.cards_battle[i][0]
        self.cards_battle = [[] for i in xrange(6)]
        self.turn_attack = 0
        self.turn_deffedned = 1
        self.lab1.text = 'attack' + ' ' + str(self.turn_attack + 1)
        self.lab2.text = text = 'defenned' + ' ' + str(self.turn_deffedned + 1)
        Clock.schedule_once(partial(self.attack_computer, 0), 1.5)

    def is_full(self): #Function that checks if the board is full, returns true if
full else false
        for i in range(len(self.cards_battle)):
            if len(self.cards_battle[i]) == 0:
                return False
        return True
    def list_end(self): #Function that returns score according the pack of
cards
        if len(self.list_cards)>15:
            return 8
        if 5<len(self.list_cards)<15:
            return 4
        return 0
    def check_can_be_in(self,battle_list,card): #Function that returns true if
card can take part in battle else false
        #Card can take part if there is a card with same number
        for i in battle_list:
            if card.number==i.number:
                return True
        return False
    def lost_good_one(self): #Function that returns true if there was loss of a
good number for the human player
        # (any quzar or non quzar card like Ace, King and queen).
        counter=0
        for j in self.cards_battle:
            if len(j)==2:
                if j[1].flag_quzar==True or j[1].number>=12 and
self.list_end()!=0:
                    return True
            counter+=1
        return False
    def attack_decision(self, battle_list,player): #Function that searches card
to attack.
        #When there is a pack of cards to take from it searches according
tactics, when there isn't it searches by brutforce.
        #Returns number of card if it finds it else false.
        if len(self.list_cards)!=0:
            if self.lost_good_one()==True:
                return False
            score_list=[]
```

```
        for i in self.list_players[1]:
            if self.is_empty():
                score=100-self.how_much_shapes(i)-self.list_end()-i.score*2
                score_list.append((score,i))
            elif self.check_can_be_in(battle_list,i):
                score = 100 - self.how_much_shapes(i) - self.list_end()  -
i.score*2
                score_list.append((score, i))
        if len(score_list)!=0:
            max = 0
            for j in range (len(score_list)):
                if max==0:
                  if score_list[j][0]>max and
self.special_condition(score_list[j][1])==False:
                    max=score_list[j]
                elif score_list[j][0] > max[0] and
self.special_condition(score_list[j][1])==False:
                  max = score_list[j]
            if max==0:
                return False
            k= self.find_this_card(max[1])
            if self.is_empty()==True:
                return k
            if self.is_empty()==False and max[0]>50:
                 return k
            return False
        else:
            return False
    elif len(self.list_players[1])!=0:
        board_for_requrse = board_for_AI()
        self.build_board(board_for_requrse)
        self.create_regu_list_players(board_for_requrse)
        options=self.find_attack_option(1,board_for_requrse)
        last_options=copy.deepcopy(board_for_requrse)
        last_options.battle_list=[[] for i in xrange(6)]
        need_board_to_win=[]
        if options!=None:
            for i in options:
                if
self.recurse_find_card(self.find_defence_option(0,i),0,False,copy.deepcop
y(i))==True:
                  need_board_to_win=i
                  break
        the_card = ''
        if need_board_to_win==[]:
             return False
        print need_board_to_win,'jjjjjjjjjjj'
        for i in range(6):
            if len(need_board_to_win.battle_list[i]) == 1:
                the_card = i
```

```
            break
        b=self.find_this_card(need_board_to_win.battle_list[the_card][0])
        print b,'ppppppp'
        return b
    def is_empty(self): #Function that if the board is empty returns true else
returns false
        for i in range(len(self.cards_battle)):
            if len(self.cards_battle[i]) != 0:
                return False
        return True
    def will_answer(self,card): #Function that gets a card and returns if this
card will be 100% defended by
        # the human player.
        for i in self.list_known:
            if i.check_bigger(card) and i.flag_quzar==False and i.number<=10:
                return 40
        return 0
    def print_board(self,matrix): #Function that prints any kind of matrix for
checking the game.
        for i in matrix:
            for j in i:
                print j.number,j.shape,'kkkkkkkkk'
            print'sssssssssss'
    def end_game_check(self): #Function that checks if there is win
condition for any player, returns true if there
        #  is else false
        if len(self.list_cards)==0:
            if len(self.list_players[0])==0 and len(self.list_players[1])==0:
                self.add_widget(Label(text='draw',pos=(800,800)))
                return True
            elif len(self.list_players[0])==0 and len(self.list_players[1])!=0:
                self.add_widget(Label(text='player won', pos=(800, 800)))
                return True
            elif len(self.list_players[0])!=0 and len(self.list_players[1])==0:
                self.add_widget(Label(text='computer won', pos=(800, 800)))
                return True
            return False
        return False

class YourApp(App): # Class that helps to build the screen that contains
the game
    def build(self): #Function that returns the Board of the game to be
shown on the screen
        self.board = Board()
        return self.board

app = YourApp()
app.run()
```

7. שימוש במערכת והרצתה

הסבר והנחיות לשימוש

על מנת להפעיל את המערכת, יש ליצור פרוייקט פייתון בסביבת העבודה PyCharm.
חשוב לבדוק שתוכנת kivy הורדה ובנוסף הפרויקט מכיל את הקבצים מהתקייה:

E:\יב\application_engineer\פרויקט דוראק\Card

בתוך התקייה קיימות התמונות שבאמצעותן ניתן להריץ את המשחק.

הוראות חשובות לממשק העכבר:

גרירת קלף מתאפשר על ידי העכבר בלבד (רק בתור השחקן ורק על הקלפים. חשוב להדגיש לגרור את הקלף ממרכזו ולהניח אותו בדיוק באזור המלבן!!). בנוסף יש ללחוץ על הכפתורים בזמן המתאים וברגע המתאים.

כפתור restart מאפשר הרצת מחדש של המשחק.

כפתור cheat מיועד לבדיקות בלבד.

כפתור pass מיועד לאחר סיום מהלך(התקפי או הגנתי).

כפתור Grab מיועד רק ללקיחת קלפים (הפסד הגנתי).

המשחק יחל רק לאחר לחיצת כפתור let's start.

8. בעיות פתוחות והצעות לשיפור

לשמחתי הרבה לא נשארו בעיות פתוחות.

לורסיה הבאה הייתי משפר שהדורק יהיה פחות תלוי בכפתורים ומידי לאחר התחלת משחק יעבור התור (למרות שזה נותן לשחקנים יותר זמן לחשוב) והייתי משנה את הצבע של המסכים.

9. נספחים

9.1 ביבליוגרפיה

- אתר fxp - https://www.fxp.co.il/showthread.php?t=3360321
- משוב מודל
- stack overflow אתר https://stackoverflow.com/