Poker – Texas Holdem

שם: אודי אראל

ת"ז: 325876977

שם המנחה: מור גולן

תוכן עניינים

תקציר ורציונל הפרויקט

כיום עולם משחקי המחשב הוא עולם גדול מאוד.

בימינו גם למשחקים הנחשבים למשחקי קופסא, משחקי לוח או משחקי קלפים נוצרה גרסה ממוחשבת. למשחקים אלו נוצר ביקוש גדול משום שניתן לשחק בהם מהבית באופן מידי ולא צריך לתאם את זמן המשחק והמקום עם חברי המשחק.

משחקי מחשב אלו גורמים להנאה אצל הרבה אנשים ולכן אני רוצה ליצור משחק מחשב כזה, המבוסס על משחק קלפים הקיים במציאות.

המשחק שאצור הוא טקסס הולדם, שהוא משחק קלפים השייך לקבוצת משחקי הפוקר שבהם משתמשים בקלפי קהילה, כלומר קלפים הגלויים לכל המשתתפים במשחק.

משחק הפוקר, וטקסס הולדם בפרט, קיבלו תאוצה גדולה מאוד בעשורים האחרונים והם נהיו מאוד פופולריים ברחבי העולם, גם בישראל.

תפקיד המשחק שאפתח יהיה לספק הנאה לאנשים רבים, גם לאנשים שלא מכירים את המשחק טקסס הולדם. המשחק שלי יהיה גם פתרון טוב לאנשים שיודעים לשחק ומכירים את המשחק טקסס הולדם ורוצים לשחק בו ולהנות.

בחרתי לעשות את הפרויקט על הנושא הזה מכמה סיבות:

- ראשית, אני אוהב את משחק הפוקר ונהנה מאוד לשחק בו עם חברים.
- בנוסף לכך, רציתי להתנסות עם משהו חדש שלא עשיתי קודם לכן, כמו בניית משחק רציני, ושימוש בתוכנה Unity.

מבוא ורקע כללי

משחק מרובה משתתפים, או בשפה האנגלית Multiplayer, הוא משחק וידאו בו משתתפים מספר שחקנים בו זמנית.

משחקי וידאו החלו כמשחקים טקסטואליים לשחקן בודד.

מאז הפריצה המסחרית של טכנולוגיית האינטרנט החל פיתוח ושיווק של משחקי וידאו עם פן של ריבוי משתתפים, שאפשרו למספר שחקנים לשחק יחדיו על גבי שרת משחק.

כמו שאמרתי קודם לכן, טקסס הולדם הוא משחק קלפים ששייך לסוג משחקי הפוקר שבהם משחקים עם קלפי קהילה (קלפים גלויים לכל השחקנים).

בסגנונות הכוללים קלפי קהילה נפתחים קלפי משחק הגלויים לעיני כול. כל משתתף מרכיב את היד שלו מהקומבינציה החזקה ביותר, שמשלבת את קלפיו האישיים ואת קלפי הקהילה.

טקסס הולדם הוא אחד ממשחקי הפוקר הנפוצים בבתי הקזינו ברחבי העולם ובעיקר במערב ארצות הברית. טקסס הולדם הוא גם סוג המשחק שמשחקים בדרך כלל באליפויות פוקר שונות, בהן ב"טורניר הפוקר העולמי", הנחשב לאליפות העולם בפוקר.

אליפות העולם בפוקר, או "טורניר הפוקר העולמי", נחשב לסבב המפורסם והיוקרתי ביותר של טורנירי פוקר. הסבב כולל נכון לשנת 2020, 101 טורנירים, במספר ווריאציות של פוקר, הנערכות מדי שנה בלאס וגאס שבמדינת נבדה שבארצות הברית.

המנצח בכל אחת מהתחרויות זוכה מלבד בפרס כספי גם בצמיד מוזהב, הנחשב יוקרתי מאוד בקרב שחקני פוקר. שיאן הזכיות בצמידי הסבב הוא פיל הלמות' האמריקאי שזכה ב-15 צמידים עד כה.

הטורניר המרכזי והיוקרתי ביותר מבין טורנירי הסבב הוא ה- Main Event, עם דמי כניסה בסך של 10,000 דולר, והזוכה בו נחשב באופן לא רשמי לאלוף העולם בפוקר. בשנת 2019 השתתפו באירוע המרכזי 8569 שחקנים.

בנוסף לכך, קיים משחק ידוע בשם Zynga Poker שהופץ על ידי החברה Zynga. משחק זה הופץ בשנת 2007 לפלטפורמות שונות, כמו אנדרואיד IOS ולדפדפן.

המשחק הוא מרובה משתמשים ומאפשר לשחקנים להתחרות אחד מול השני בזמן אמת ממקומות ומכשירים שונים.

מהלך משחק הטקסס הולדם הוא:

בתחילת המשחק הדילר (המחלק), שאינו משתתף במשחק, מערבב חפיסה רגילה של 52 קלפים (בלי ג'וקרים).

כל שחקן מקבל מהדילר שני קלפים שרק הוא יודע אותם. לאחר מכן, תלוי איך משחקים, לפעמים ישנו הימור התחלתי שהשחקנים חייבים להמר כדי להישאר במשחק ולפעמים לא.

אחר כך, הדילר פותח שלושה קלפי קהילה הנקראים "Flop", שלאחריו מתחיל סבב הימורים: כל שחקן יכול להמר או לומר "Check", כלומר לא להמר, בתנאי שאף אחד לא הימר לפניו. אם בוצע הימור, כל השחקנים האחרים חייבים להשוות, להעלות את ההימור או לפרוש.

לאחר שנגמר סבב ההימורים הזה, הדילר פותח עוד קלף קהילה אחד שנקרא "Turn", ונפתח עוד סבב הימורים.

כעת הדילר פותח את קלף הקהילה האחרון, שנקרא "River", כך שיש חמישה קלפי קהילה. נפתח סבב ההימורים האחרון שבסופו כל השחקנים חושפים את הקלפים שלהם והשחקן בעל היד החזקה ביותר מנצח ומרוויח את כל כספי ההימורים.

יכול גם להיווצר מצב בו כל השחקנים חוץ מאחד פרשו לפני סיום המשחק, וכאשר מצב זה קורה, כמובן שהאחד שלא פרש הוא המנצח והוא לא חייב להראות את הקלפים שלו.

מטרת הפרויקט

מה המוצר המוגמר אמור לבצע:

המוצר אמור להיות משחק טקסס הולדם למי שאוהב, משחק נוח ופשוט לשימוש המאפשר לשחק אחד מול השני בעזרת חיבור לשרת.

ברגע שכל השחקנים מתחברים לשרת, המשחק מתחיל.

בעזרת העכבר ניתן לבצע את פעולות ההימורים בעזרת לחיצה על כפתורים שונים.

המשחק יציג את ההימור שכל שחקן ביצע, יציג אם הוא פרש ובסיום המשחק יציג את המנצח ויביא לו את הכסף שמגיע לו.

דרישות מרכזיות:

דרישות פונקציונאליות:

1. פשטות למשתמש
   - משחק קל להבנה
   - קל לשימוש
2. מימוש של כל חוקי המשחק טקסס הולדם.
3. יהיה משתמש אחד שיוכל לצפות במשחק
   - הצופה יראה את הקלפים הנסתרים של כל השחקנים.
   - לצופה יהיה כתוב אחוזי ניצחון של כל שחקן.
4. שמירת הכסף של כל שחקן.

דרישות לא פונקציונאליות:

1. פשטות בחיבור – יהיה ניתן בקלות ובמהירות לשנות את כתובת השרת שהוא מאזין אליה.
2. ריצה חלקה – העברת מידע בדרך שלא תגרום למשחק להיות איטי ועם דיליי גדול בין השחקנים (דיליי של יותר מ- 5 שניות הוא דיליי גדול מדיי).
3. המשחק יוכל לרוץ על כל מחשב.
4. משחק Multiplayer – מספר שחקנים ישחקו אחד נגד השני ויהיה צופה אחד.

תרחישים במערכת:

כניסת משתמש למערכת:

המשתמש פותח את המערכת. מסך הבית מופיע, והמשתמש ילחץ על כפתור Play כדי להתחיל לשחק או לצפות (האחרון שמתחבר למשחק הוא הצופה).

שחקן:

a. המשתמש מחכה ש- 5 אנשים יכנסו והמשחק מתחיל.
b. בכל פעם שהמשחק נגמר המערכת תכריז על המנצח והוא ירוויח את כל הכסף שבקופה. לאחר מכן יתחיל משחק חדש.
c. אם משתמש יוצא מהמשחק, המשחק לא יתחיל מחדש ויחכה שעוד משתמש יכנס.

צופה:

a. המשתמש צופה במהלך המשחק, כל הקלפים של כל השחקנים גלויים לו.
b. רשומים למשתמש אחוזי ניצחון של כל שחקן.

שפות התכנות וסביבת העבודה

השפות בהן החלטתי להשתמש בפרויקט שלי הן #C ו – Python.

בשפת #C השתמשתי בשביל המשחק עצמו והלקוחות ובשפת Python השתמשתי בשביל השרת.

שפת #C (מבוטא סי שארפ) היא שפת תכנות מונחית עצמים בעיקרה שפותחה ע"י מיקרוסופט ונחשבת לאחת משפות התכנות הפופולריות בעולם. היא מיועדת לפיתוח כללי של מגוון אפליקציות בכל התחומים – מאתרי Web דרך משחקים, מאפליקציות למכשירי מובייל וטאבלטים ועד לשירותי ענן. התחביר והעקרונות שלה הם פשוטים מצד אחד אך עשירים ביכולות מצד שני.

Python הינה שפת פיתוח לכל מטרה המאפשרת כתיבה מהירה של תוכניות פשוטות ותומכת גם בפיתוח מונחה עצמים לצורך תוכניות מורכבות יותר. השפה נמצאת בשימוש נרחב בארץ ובעולם, גם כשפת צד-שרת בפיתוח, Web גם בעולם של Ops Dev כשפת סקריפטים, וגם מפתחים משתמשים בפייתון לביצוע אוטומציה ומשימות קטנות. מפתחים ואנשי אוטומציה שמגיעים משפות אחרות ימצאו את פייתון ידידותית ביותר.

בחרתי לכתוב בשתי השפות האלה מפני שאני מכיר אותן דיי טוב, יצא לי לעבוד איתן וללמוד את שתיהן במהלך השנים שלי בבית הספר, על ידי כתיבת כל מיני פרויקטים, ממשחקים פשוטים ב #C עם מחלקות רבות ופרויקטים של תקשורת ב Python.

את הקליינט והמשחק שנוצר ב Unity כתבתי ב #C כיוון ש Unity עובד רק עם JavaScriptו #C ולכן היה לי מאוד נוח לכתוב ב #C כי אני כבר עם ניסיון בשפה הזו. את השרת כתבתי ב Python מפני שזאת שפה שאני שולט בה דיי טוב ולכתוב בה שרת מאוד נוח ופשוט.

ניסוח וניתוח הבעיה האלגוריתמית

לפרויקט יש מספר בעיות אלגוריתמיות:

1. סנכרון המשחק בין המשתתפים וחווית משחק רציפה:

יש צורך בשליחה של מידע בין הקליינטים דרך השרת בצורה יעילה ואמינה לשם סנכרון השולחנות של השחקנים וכדי שהמשחק יהיה רציף ולא יהיו תקיעות.

2. ייצוג גרפי של המשחק:

המסך עצמו משחק דו ממדי עם ייצוג גרפי וכל האובייקטים הגרפיים אמורים לפעול בהתאם למידע שמתקבל מהשחקן עצמו (תזוזת העכבר ולחיצה על כפתורים) והמידע שמתקבל מהשרת.

3. הכרת כל החוקים והתמודדות עימם:

המשחק אמור ליישם את כל חוקי המשחק, להכיר מצבים שונים שיכולים להתרחש במהלך המשחק ולדעת איך להתמודד איתם בצורה הנכונה בכל מצב.

4. הערכת ה"ידיים" של השחקנים:

על מנת להכריז על המנצח ועל מנת להציג לצופה במשחק נתונים על אחוזי הניצחון של כל שחקן, המערכת צריכה שתהיה לה את היכולת להעריך את חוזקה של ה"יד" (קומבינציית הקלפים) של כל שחקן.

תיאור אלגוריתמים קיימים

האלגוריתמים הקיימים הפותרים את הבעיות האלגוריתמיות שנתקלתי בהן הם:

1. סנכרון המשחק בין המשתתפים וחווית משחק רציפה:
   - העברת מידע בעזרת פרוטוקול TCP
   - העברת מידע בעזרת פרוטוקול UDP
   - שימוש בכל פרוטוקול אחר להעברת מידע.
2. ייצוג גרפי של המשחק:
   - בפייתון קיימת ספרייה בשם PyGame שניתן ליצור איתה משחקים שהם דו ממדיים, אך יחסית קשה להגיע עם ספרייה זו למשחקים שמעוצבים ממש טוב.
   - קיימת תוכנה בשם Unity שפותחה לצורך פיתוח משחקי וידאו למחשב, קונסולות, טלפונים חכמים ואתרי אינטרנט. בעזרת Unity ניתן להגיע יחסית בקלות למשחקים שמעוצבים באופן מרשים ויפה.
3. הערכת ה"ידיים" של השחקנים:

לאחר חיפוש באינטרנט ומחקר מעמיק, מצאתי מספר ספריות שאנשים יצרו ופרסמו באתרים כמו GitHub ו CodeProject שיכולות להעריך "יד" של שחקן וגם להציג נתונים של אחוזי ניצחון.

הפתרון הנבחר

1. סנכרון המשחק בין המשתתפים וחווית משחק רציפה:

ישנן כל מיני דרכים להעביר מידע דרך כל מיני פרוטוקולי תקשורת אך במשחק הזה אין שום העברת מידע אישי או דברים שכדאי להצפין וצריכים הגנה, לכן לא נשתמש בפרוטוקולים מסובכים, אלא ניקח לדוגמה את הפרוטוקולים TCP או UDP.

UDP יותר מהיר מפני שלא צריך לשלוח אישור לקבלת המידע. אולם, הבעיה בו היא שזהו פרוטוקול פחות אמין מ TCP. TCP אמין אך יותר איטי מ UDP.

אחת הדרישות שלי היא שהמשחק יעבוד חלק ללא תקיעות בין השחקנים וכמובן סנכרון ותיאום בכל המסכים ולכן, צריך גם אמינות בשליחת המידע. כלומר, אני רוצה גם אמינות וגם מהירות. בעקבות זאת בחרתי להשתמש ב TCP מפני שהוא אמין כמו שאמרנו, ומפני שאם אעביר כמה שפחות מידע בזמן המשחק, הוא יהיה גם מהיר. לדוגמא: בתחילת המשחק כל משתמש מקבל מהשרת את כמות הכסף של כל השחקנים האחרים, וכך אין צורך להעביר את המידע הזה כל פעם מחדש שמתבצע הימור.

2. ייצוג גרפי של המשחק:

בחרתי להשתמש במנוע הגרפי Unity שהוא מאוד מוכר ועם יכולות רחבות שמקלות על העבודה. ב Unity אפשר ליצור משחקים בדו ממד וזה מה שאני הייתי צריך כדי ליצור את המשחק שלי. ב Unity ישנה גם חנות של דברים חינמיים וכן דברים בתשלום, דברים מובנים שמאוד יכולים להקל על הפרויקט ולחסוך המון זמן. לדוגמא: במקום לעצב בעצמי את 52 קלפי המשחק, לקחתי מהחנות של Unity את העיצוב של הקלפים וזה הקל עליי מאוד וחסך המון זמן בביצוע הפרויקט.

3. הערכת ה"ידיים" של השחקנים:

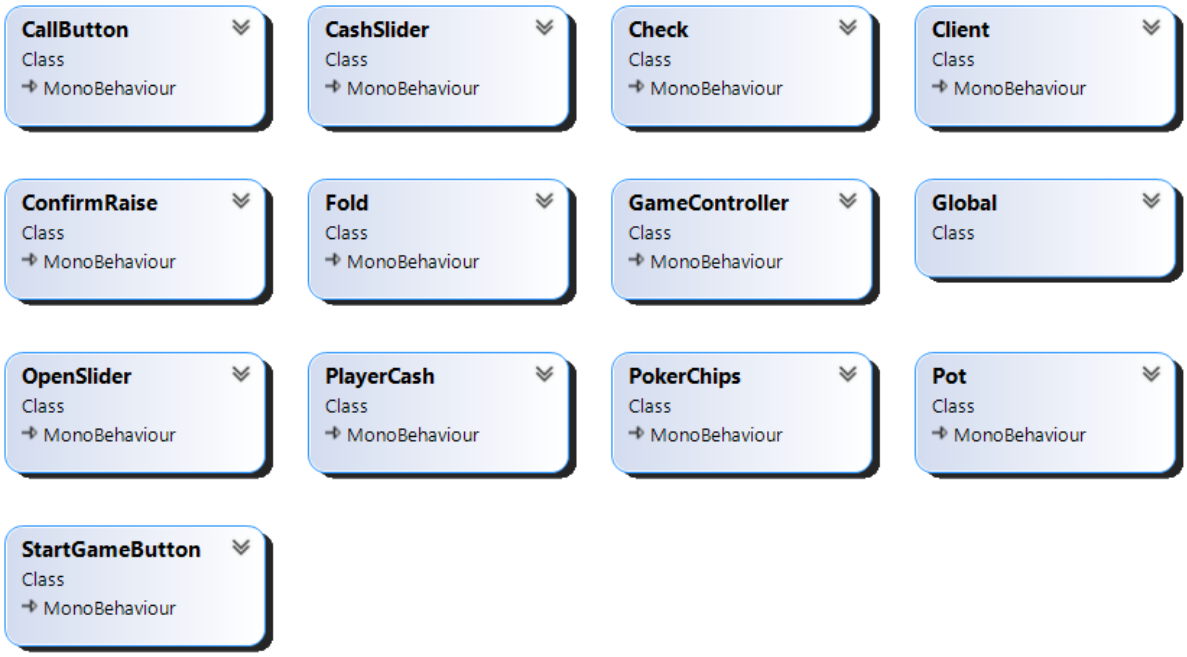
החלטתי להשתמש בספריית eval7 בפייתון, משום שהיא נוחה מאוד לשימוש: בהינתן ה"יד" היא מחשבת ומחזירה את דירוג ה"יד" באמצעות מספר – ככל שהמספר גדול יותר ה"יד" חזקה יותר. לכן, על מנת להכריז על המנצח היה עליי רק להפעיל את הפעולה על כל ה"ידיים" של כל השחקנים ואחרי זה זו פעולה פשוטה שמוצאת את הערך המקסימלי.

בחרתי גם להשתמש בספרייה holdem_calc בפייתון כי גם היא מאוד נוחה לצרכים שלי: יש לה את היכולת להציג אחוזי ניצחון של כל שחקן,

בהינתן הקלפים שעל השולחן והקלפים של השחקנים.

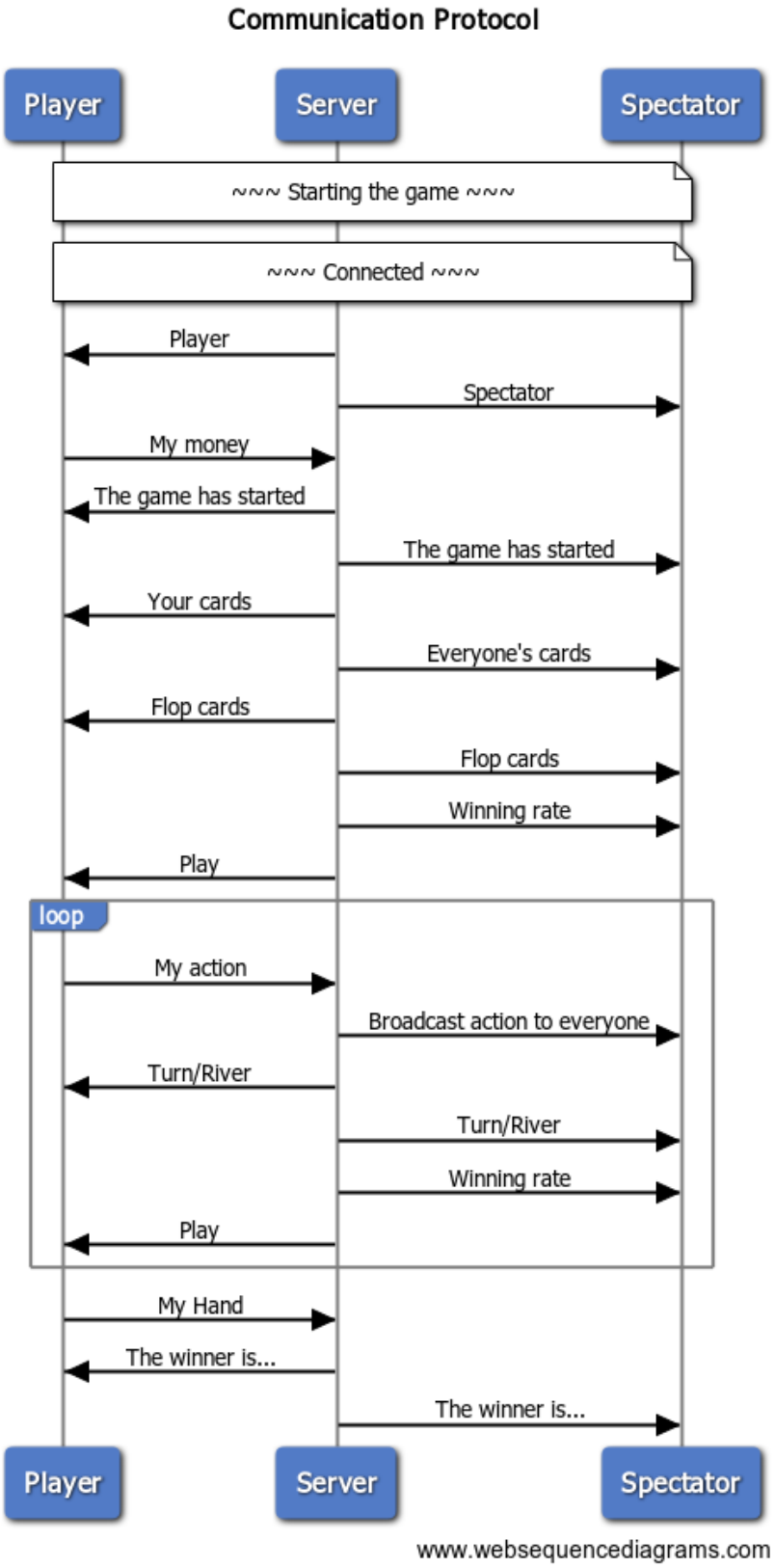
מבנה המערכת

מבנה המחלקות:



פרוטוקול תקשורת:

זהו פרוטוקול התקשורת לסיבוב אחד של משחק. לשם הנוחות יש רק שחקן אחד, אך זה מתאים גם למספר שחקנים:



פיתוח הפתרון בשכלול הקוד עם שפת התכנות

השרת:

```
#accept client. send him player/spectator and recv from him his money
def accept_client(self, client):
    connection, self.client_address = client.accept()
    print('new connection from ' , self.client_address)
    connection.setblocking(1)
    #only the last client is the spectator
    if self.players_counter < self.max_num_clients - 1:
        print('player')
        connection.send('player'.encode())
        #recv the money of the player
        self.players_cash.append(json.loads(connection.recv(1024)))
    else:
        print('spectator')
        connection.send('spectator'.encode())
    connection.setblocking(0)
    self.inputs.append(connection)
    self.message_queues[connection] = queue.Queue()
    self.players[connection] = self.players_counter
    self.players_counter += 1
    self.matched_start()
```

זוהי פעולה המאשרת חיבור של לקוח. הפעולה שולחת ללקוח אם הוא שחקן או צופה. אם הלקוח הוא שחקן, אז הפעולה גם מקבלת את סכום הכסף שלו. הפעולה גם מעדכנת את המשתנים הרלוונטיים.

```python
#after all the clients have connected, start the game
def matched_start(self):
    #start the game only if all the clients have connected
    if self.players_counter == self.max_num_clients:
        print(self.players_cash)
        cnt = 2
        players_list = list(self.players.values())
        #send all the clients that the game started and a list of neccessary things for the game: player numbers, players money
        for player in self.players:
            if self.players[player] == self.max_num_clients - 1:
                player.send(json.dumps(['game started','Spectator', players_list, self.players_cash]).encode())
            elif self.players[player] == 0:
                player.send(json.dumps(['game started','sb', players_list[:-1], self.players_cash[1:]]).encode())
            elif self.players[player] == 1:
                player.send(json.dumps(['game started','bb', players_list[1:-1] + players_list[0::-1],
                                        (self.players_cash[1:] + self.players_cash[0::-1])[1:]]).encode())
            else:
                player.send(json.dumps(['game started','None',players_list[cnt:-1] + players_list[cnt-1::-1],
                                        (self.players_cash[cnt:] + self.players_cash[cnt-1::-1])[1:]]).encode())
                cnt += 1
        #the last player of a gambling round
        self.last_player = list(self.players)[-2]
        self.generate_deck()
        #send the private cards to the clients
        self.share_cards()
        self.cards_without_fold_cards = self.cards.copy()
        #send the first 3 community cards to the clients
        self.flop_turn_river(3)
        #send play to the first player
        self.message_queues[list(self.players)[0]].put('play'.encode())
        #send the winning rates to the spectator
        stats = holdem_calc.calculate(self.generate_hand_holdem_calc(self.cards[(self.players_counter - 1) * 2:]), True, 1, None,
                                      self.generate_hand_holdem_calc(self.cards[:(self.players_counter - 1) * 2]), False)
        print(stats)
        self.message_queues[list(self.players)[-1]].put(json.dumps(stats).encode())
        print('The game has started')
```

הפעולה מתחילה את המשחק: שולחת לכל לקוח רשימה המכילה בעיקרון את רשימת השחקנים ואת הכסף שלהם. היא שולחת לשחקנים ולצופה את הקלפים האישיים, לאחר מכן את קלפי הקהילה, אחר כך שולחת לשחקן הראשון שהוא יכול לשחק ולבסוף שולחת לצופה את אחוזי הניצחון של כל שחקן.

```python
#in fact, this is the main method of the game. it receives the action of a client and respond respectively
def receive_data(self, client):
    data = client.recv(1024)
    if data:
        #the data has to be a list
        data = json.loads(data)
        if not self.over:
            #the action is fold
            if data[0] == 'fold':
                self.fold_players.append(self.players[client])
                for i in range(2):
                    self.cards_without_fold_cards.pop(self.players[client] * 2 - (len(self.fold_players) - 1) * 2)
                message = data.copy()
                message.append(self.players[client])
                #don't send the spectator the data again
                self.dont_send_to_spec = True
                list(self.players)[-1].send(json.dumps(message).encode())
                time.sleep(0.5)
                stats = holdem_calc.calculate(self.generate_hand_holdem_calc(self.cards[
                    (self.players_counter - 1) * 2:]), True, 1, None, self.generate_hand_holdem_calc(
                    self.cards_without_fold_cards[:(self.players_counter - 1 - len(self.fold_players)) * 2]), False)
                print(stats)
                list(self.players)[-1].send(json.dumps(stats).encode())
                #everyone except one player is fold - game over
                if len(self.fold_players) == len(self.players) - 2:
                    self.over = True
                    self.broadcast_without_spectator('game over'.encode())
```

זהו חלק מהפעולה שהיא למעשה הפעולה העיקרית בשרת. הפעולה מקבלת מידע מאחד הלקוחות (המידע חייב להיות רשימה. הוא מכיל את מה שביצע השחקן) ופועלת לפי מה שקיבלה. בחלק זה, הפעולה מטפלת במצב בו השחקן פרש מהסיבוב הנוכחי.

```
#action is raise, the last player of the gambling round need to change
elif data[0] == 'raise':
    self.last_player = list(self.players)[:-1][list(self.players).index(client) - 1]
#the gambling round is over
if self.last_player == client:
    print(self.cards_counter)
    #if there are 4 or less cards, there is need to send another card
    if self.cards_counter <= 4:
        self.flop_turn_river(1)
        stats = holdem_calc.calculate(self.generate_hand_holdem_calc(self.cards[
        (self.players_counter - 1) * 2:]), True, 1, None, self.generate_hand_holdem_calc(
        self.cards_without_fold_cards[:(self.players_counter - 1 - len(self.fold_players)) * 2]), False)
        print(stats)
        self.message_queues[list(self.players)[-1]].put(json.dumps(stats).encode())
    #game over
    else:
        self.over = True
        self.broadcast_without_spectator('game over'.encode())
#append the one who send the data to the received list and send it to all the clients except him
data.append(self.players[client])
print('received ' , data , ' from ' , client.getpeername())
for player in self.players:
    if player is not client:
        #if the spectator has already gotten the data
        if player is list(self.players)[-1] and self.dont_send_to_spec:
            self.dont_send_to_spec = False
        else:
            self.message_queues[player].put(json.dumps(data).encode())
            if player not in self.outputs:
                self.outputs.append(player)
```

זהו המשך הפעולה. בחלק זה, הפעולה מטפלת במצב בו השחקן העלה את ההימור. בנוסף, היא מטפלת גם במצב שבו נגמר סבב הימורים מסוים וצריך לפתוח קלף קהילה נוסף או לסיים את המשחק. בנוסף לכך, לאחר שהפעולה טיפלה במצבים השונים, היא שולחת לכל הלקוחות את מה שביצע השחקן כדי שכולם יראו את זה במסך שלהם.

```
        #if the game is'nt over, send play to the next player
        if not self.over:
            next_player = list(self.players)[(list(self.players).index(client) + 1) % (self.max_num_clients - 1)]
            count = 2
            while self.players[next_player] in self.fold_players:
                next_player = list(self.players)[(list(self.players).index(client) + count) % (self.max_num_clients - 1)]
                count += 1
            self.message_queues[next_player].put('play'.encode())
    #the game is over, receive everyone hands to evaluate them and braodcast the winner
    else:
        self.hands.append(data)
        if len(self.hands) == (self.max_num_clients - 1) - len(self.fold_players):
            self.winner = self.evaluate_all_hands(self.hands)
            self.broadcast(str(self.winner).encode())
            threading.Thread(target = self.reset_properties, args = (4,)).start()
#the client is close
else:
    print('closing ' , self.client_address , ' after reading no data')
    if client in self.outputs:
        self.outputs.remove(client)
    self.inputs.remove(client)
    #client.close()
    #del self.message_queues[client]
    self.players_counter -= 1
    self.max_num_clients -= 1
```

בהמשך, הפעולה שולחת לשחקן הבא בתור שזה תורו. בנוסף לכך, אם המשחק הסתיים, הפעולה מקבלת את ה"ידיים" של כל השחקנים ולאחר חישוב שולחת לכולם מי המנצח. יתר על כן, הפעולה סוגרת את החיבור אם הלקוח התנתק.

השחקן:

המחלקה שמנהלת את המשחק:

```
using System;
using System.Collections.Generic;
using System.IO;
using UnityEngine;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using Assets;
using UnityEngine.UI;
using Newtonsoft.Json;
using System.Collections;
using System.Linq;
using UnityEngine.SceneManagement;

Unity Script | 9 references
public class GameController : MonoBehaviour
{
    //the cards in the game
    private GameObject[] gameCards;
    private string[] shapes;
    private int[] values;
    private int cardsCounter;
    //the red Xs that show that someone is fold
    public GameObject[] redXs;
    //the crown of the winner and its positions
    public GameObject crown;
    private readonly Vector2[] crownPos = { new Vector2(0, -4f), new Vector2(-9.65f, 0),
        new Vector2(-9.65f, 5.5f), new Vector2(9.45f, -5.5f), new Vector2(9.45f ,0) };
    public static GameController _instance;
    private Pot myPot;
    private PokerChips myPokerChips;
    private Client myClient;
    private PlayerCash myPlayerCash;
    private List<int> playersCash;
    public Text[] playersCashText;
    public Button[] myButtons;
    //the numbers of the players
    private List<int> playersNum;
    // the next cards of the game
    private List<string> nextCards;
```

זוהי בעצם המחלקה העיקרית בקוד של הלקוח. היא אחראית לניהול השוטף של המשחק. כאן מוצגת הגדרת המשתנים.

```
//an action of other player
private List<object> playersAction;
private string canPlay;
private Vector2 posTableCards;
private readonly Vector2[] posAllPlayersCards = { new Vector2(-2f, -6f), new Vector2(-11.65f, -2f),
    new Vector2(-11.65f, 3.5f), new Vector2(7.45f, -7.5f), new Vector2(7.45f ,-2f) };
private int lastRaise;
//does the game over
private string isOver;
//the player number of the winner
private int winner;
private bool isFold;
//True if player False if spectator
private bool isPlayer;
//the winning rates
private List<float> stats;
public Text[] statsTexts;
private List<int> foldPlayers;
```

המשך הגדרת משתנים.

```
/// <summary>
/// gets data from the server and assign it to the right variable. running all the time in a different thread
/// </summary>
1 reference
private void GetMessageFromServer()
{
    while (true)
    {
        object message = myClient.ReturnData();
        if (message is List<string>)
            nextCards = (List<string>)message;
        else if (message is List<object>)
            playersAction = (List<object>)message;
        else if (message is List<float>)
            stats = ((List<float>)message).Skip(1).ToList();
        else
        {
            if ((string)message == "play")
                canPlay = (string)message;
            else if ((string)message == "game over")
                isOver = (string)message;
            else
                winner = int.Parse((string)message);
        }
    }
}
```

בחלק זה מוצגת הפעולה שמטפלת בתקשורת עם השרת. הפעולה משתמשת בפעולות של המחלקה Client. הפעולה רצה כל הזמן ב Thread שונה משאר הקוד. הפעולה מחכה שהשרת ישלח לה מידע, ולפי הטיפוס שלו היא יודעת מה השרת שלח לה והיא פועלת בהתאם.

```
191     /// <summary>
192     /// connects to the server and define the variables
193     /// </summary>
        1 reference
194     private void ConnectToServer()
195     {
196         myClient.StartClient();
197         isPlayer = myClient.Connect() == "player";
198         if (isPlayer)
199         {
200             //if player set its money and send it to the server
201             Debug.Log("player");
202             myPlayerCash.SetPlayerAmount();
203             myClient.Send(myPlayerCash.GetAmount().ToString());
204             playersCashText = playersCashText.Skip(1).ToArray();
205         }
206         //if spectator disable the buttons
207         else
208             foreach (Button button in myButtons)
209                 button.gameObject.SetActive(false);
210         List<object> data = myClient.StartGame();
211         playersNum = (List<int>)data[2];
212         playersCash = (List<int>)data[3];
213     }
```

בחלק זה מתבצע ההתחברות לשרת והתחלת המשחק. השרת שולח אם הלקוח הוא שחקן או צופה. אם הלקוח הוא שחקן, הוא שולח לשרת כמה כסף יש לו.

```
240         //connecting to the server
241         ConnectToServer();
242         //show the amount of money of the all players
243         for (int i = 0; i < playersCash.Count; i++)
244             playersCashText[i].text = Global.CashToString(playersCash[i]) + " $";
245         if (isPlayer)
246             //puts the player's cards
247             PutFirstCardsOnScreen((List<string>)myClient.ReturnData(), 2);
248         else
249             //puts all the players cards
250             for (int i = 0; i < playersNum.Count - 1; i++)
251                 PutOnePlayerCardsOnScreen((List<string>)myClient.ReturnData(), i);
252         //if there are less than 5 players, put red X on them
253         if (isPlayer)
254             for (int i = 4; i >= playersNum.Count; i--)
255                 redXs[i].SetActive(true);
256         else
257             for (int i = 4; i >= playersNum.Count - 1; i--)
258                 redXs[i].SetActive(true);
259         //start the thread of the communication
260         Thread thread = new Thread(new ThreadStart(GetMessageFromServer));
261         thread.Start();
262     }
```

זהו חלק מפעולת Start של Unity שרצה פעם אחת בלבד לפני ה Frame הראשון. בחלק זה מתבצעת קריאה לפעולה הקודמת שמתחברת לשרת ומתחילה את המשחק. בהמשך, הפעולה מציגה על המסך את הכסף של כל שחקן ואת הקלפים הרלוונטיים. בנוסף, היא מריצה את ה Thread של התקשורת.

```
Unity Message | 0 references
private void Update()
{
    string stateName = null;
    //put next table cards on the table
    if (nextCards != null)
    {
        PutCardsOnScreen(nextCards, nextCards.Count);
        if (isPlayer)
            lastRaise = 0;
        nextCards = null;
    }
    //does an action of other player
    if (playersAction != null)
    {
        //the action is raise or call
        if ((string)playersAction[0] == "raise" || (string)playersAction[0] == "call")
        {
            if (isPlayer)
                if ((string)playersAction[0] == "raise")
                    lastRaise = Convert.ToInt32(playersAction[1]);
            Call(Convert.ToInt32(playersAction[1]), playersNum.IndexOf(Convert.ToInt32(playersAction[2])));
        }
        //the action is fold
        else if ((string)playersAction[0] == "fold")
        {
            redXs[playersNum.IndexOf(Convert.ToInt32(playersAction[2]))].SetActive(true);
            foldPlayers.Add(playersNum.IndexOf(Convert.ToInt32(playersAction[2])));
        }
        playersAction = null;
    }
```

זוהי הפעולה המרכזית של המחלקה, פעולת ה Update של Unity שמופעלת אוטומטית בתחילת כל Frame. הפעולה מעדכנת את המסך לפי המידע שהתקבל מהשרת. בחלק הזה, הפעולה מציגה על המסך את קלפי קהילה החדשים במידה והתקבלו והיא מציגה גם את מה שביצע שחקן מסוים.

```
293         if (isPlayer && canPlay != null)
294         {
295             //this is the player's turn. enable the buttons
296             if (lastRaise == 0)
297                 foreach (Button button in myButtons)
298                 {
299                     if (button.name != "CallButton")
300                         button.interactable = true;
301                 }
302             else
303                 foreach (Button button in myButtons)
304                 {
305                     if (button.name != "CheckButton")
306                         button.interactable = true;
307                 }
308             canPlay = null;
309         }
310         if(isPlayer && !isFold && isOver != null)
311         {
312             //the game is over. disable the buttons and send the player's hand to the server for evaluation
313             foreach (Button button in myButtons)
314                 button.interactable = false;
315             List<string> hand = new List<string>();
316             for (int i = 0; i < shapes.Length; i++)
317                 if (values[i] != 0)
318                     hand.Add(shapes[i] + '_' + values[i]);
319             myClient.Send(JsonConvert.SerializeObject(hand));
320             isOver = null;
321         }
```

בהמשך, הפעולה משחררת את נעילת הכפתורים במידה וזהו תורו של השחקן.

בנוסף, במידה והמשחק נגמר והשחקן לא פרש, הפעולה שולחת לשרת את ה"יד" של השחקן.

```
322         if(winner > -1)
323         {
324             //the server found the winner
325             int winnerIndex = playersNum.IndexOf(winner);
326             Debug.Log(winnerIndex);
327             //set the crown above the winner
328             crown.transform.position = crownPos[winnerIndex];
329             crown.SetActive(true);
330             //give the winner his money with an animation
331             int potCash = myPot.EmptyPot();
332             Debug.Log("The winner gets: " + potCash);
333             myPokerChips.CallWithoutAnim(potCash, winnerIndex);
334             stateName = "PokerChips" + winnerIndex + " R";
335             myPokerChips.GetAnimator().Play(stateName);
336             if (isPlayer)
337                 if (winnerIndex == 0) myPlayerCash.AddAmount(potCash);
338                 else
339                 {
340                     playersCash[winnerIndex - 1] += potCash;
341                     playersCashText[winnerIndex - 1].text = Global.CashToString(playersCash[winnerIndex - 1]) + " $";
342                 }
                    (field) Text[] GameController.playersCashText
343             else
344             {
345                 playersCash[winnerIndex] += potCash;
346                 playersCashText[winnerIndex].text = Global.CashToString(playersCash[winnerIndex]) + " $";
347             }
348             winner = -1;
349             Debug.Log(myPlayerCash.GetAmount());
350             //restart the game by loading the same scene. before of that changing the amount of money of the player in the file
351             myPlayerCash.ChangeAmountFile(myPlayerCash.GetAmount());
352             Thread.Sleep(5000);
353             SceneManager.LoadScene("Game");
354         }
```

בחלק זה של הפעולה, הפעולה מטפלת במצב בו השרת שולח ללקוח את המנצח. הפעולה מציגה על המסך את המנצח, ומביאה לו את הכסף שמגיע לו. לבסוף, הפעולה מאתחלת מחדש את המשחק.

```
355         if (!isPlayer && stats != null)
356         {
357             //show the winning rate of all the players
358             int statsIndex = 0;
359             for (int i = 0; i < stats.Count + foldPlayers.Count; i++)
360             {
361                 //if player is fold he has obviously no chance to win
362                 while (foldPlayers.Contains(i))
363                 {
364                     statsTexts[i].text = "Win: 0%";
365                     i++;
366                 }
367                 if (i < stats.Count + foldPlayers.Count)
368                     statsTexts[i].text = "Win: " + Math.Round(stats[statsIndex++] * 100).ToString() + "%";
369             }
370             stats = null;
371         }
372     }
```

זהו החלק האחרון של הפעולה שבו היא מציגה על המסך את אחוזי הניצחון של כל שחקן במידה והלקוח הוא צופה.

המחלקה האחראית על הכסף של השחקן:

המחלקה הזו אחראית לניהול כספו של השחקן. המחלקה שומרת את כספו של השחקן כדי שלא יעלם בכל פעם שהוא נכנס מחדש למשחק, על ידי שמירת הכסף בקובץ במחשב של השחקן.

```
39      /// <summary>
40      /// creating the file which saves the money
41      /// </summary>
42      /// <param name="filename"></param>
43      /// <param name="writtenValue"></param>
44      /// <returns></returns>
        2 references
45      private string CreateFile(string filename, int writtenValue)
46      {
47          //remove exist file
48          if (File.Exists(filename))
49              File.Delete(filename);
50          //create the file and write the money in it
51          using (FileStream fs = File.Create(filename))
52          {
53              byte[] text = new UTF8Encoding(true).GetBytes(writtenValue.ToString());
54              fs.Write(text, 0, text.Length);
55          }
56          //open the file and return the money in it
57          using (StreamReader sr = File.OpenText(filename))
58          {
59              string s = "", text = "";
60              while ((s = sr.ReadLine()) != null)
61                  text += s;
62              return text;
63          }
64      }
```

פעולה זו יוצרת את הקובץ במחשב של השחקן וכותבת שם כמה כסף יש לו.

```
70      private string OpenFile(string filename)
71      {
72          //open the file and return the money in it
73          using (StreamReader sr = File.OpenText(filename))
74          {
75              string s = "", text = "";
76              while ((s = sr.ReadLine()) != null)
77                  text += s;
78              return text;
79          }
80      }
81      /// <summary>
82      /// get the player's amount from the file. if the player is new, give him 2 million
83      /// </summary>
        1 reference
84      public void SetPlayerAmount()
85      {
86          //get amount from exist file
87          try
88          {
89              string stringAmount = OpenFile(path);
90              amount = int.Parse(stringAmount);
91          }
92          //create new file and give 2 million
93          catch
94          {
95              string stringAmount = CreateFile(path, 2000000);
96              amount = int.Parse(stringAmount);
97          }
98          if (amount <= 0)
99              amount = 2000000;
100         Debug.Log(amount);
101         UpdateText();
102     }
```

הפעולה הראשונה פותחת את הקובץ ומחזירה את הכסף שכתוב בו.

הפעולה השנייה היא הפעולה המרכזית במחלקה: היא משלבת את הפעולות הקודמות וקובעת את כספו של השחקן.

המחלקה האחראית על הפוקר צ'יפס:

מחלקה זו היא האחראית על הפוקר צ'יפס – על התמונה שלהם, על האנימציה שלהם, על הערך שלהם ועוד.

```
41      /// <summary>
42      /// does a call/raise from given amount and number of animation
43      /// </summary>
44      /// <param name="amount"></param>
45      /// <param name="animNum"></param>
        3 references
46      public void Call(int amount, int animNum)
47      {
48          this.amount = amount;
49          //define the sprite relatively the amount of money
50          SetSpriteByAmount();
51          myText.GetComponent<Transform>().position = transform.position;
52          myText.text = Global.CashToString(amount) + " $";
53          //change the animation to the right animation
54          ChangeAnimation(animNum);
55          //starts the animation
56          myAnimator.SetTrigger("Trigger");
57      }
```

פעולה זו היא המרכזית במחלקה – היא מבצעת הימור.

המחלקה האחראית על הצד של הלקוח בתקשורת:

המחלקה הזו אחראית על החלק של התקשורת עם השרת.

```
/// <summary>
/// return from the server the list which include mainly the number of the players and their money. than the game start
/// </summary>
/// <returns></returns>
1 reference
public List<object> StartGame()
{
    int bytesRec = ReceiveData(sender);
    //deserialize the object
    string message = Encoding.ASCII.GetString(bufferBytes, 0, bytesRec);
    List<object> data = JsonConvert.DeserializeObject<List<object>>(message);
    //convert to list of ints
    data[2] = ((Newtonsoft.Json.Linq.JArray)data[2]).ToObject<List<int>>();
    data[3] = ((Newtonsoft.Json.Linq.JArray)data[3]).ToObject<List<int>>();
    return data;
}
```

פעולה זו בעצם מקבלת את הרשימה ההתחלתית שהשרת שולח ללקוח, הרשימה שמעידה על כך שהמשחק התחיל. לבסוף היא מחזירה את הרשימה לאחר שהיא מבצעת בה המרות לשם הנוחות.

```
/// <summary>
/// return data from the server: can be list of objects, list of strings, list of floats or a string
/// </summary>
/// <returns></returns>
3 references
public object ReturnData()
{
    //get the data
    int bytesRec = ReceiveData(sender);
    string message = Encoding.ASCII.GetString(bufferBytes, 0, bytesRec);
    Debug.Log(message);
    try
    {
        //trying to convert it to different kinds of lists
        List<object> data = JsonConvert.DeserializeObject<List<object>>(message);
        if (data.Count == 1)
            return JsonConvert.DeserializeObject<List<string>>(JsonConvert.SerializeObject(data));
        else if (data[1] is string)
            return JsonConvert.DeserializeObject<List<string>>(JsonConvert.SerializeObject(data));
        else if(data[1] is double)
            return JsonConvert.DeserializeObject<List<float>>(JsonConvert.SerializeObject(data));
        return data;
    }
    catch
    {
        //the data is a string
        return message;
    }
}
```

פעולה זו היא המרכזית במחלקה: היא מקבלת מידע שמגיע מהשרת, ולפי הטיפוס שלו היא יודעת מה היא קיבלה ולמה להמיר אותו אם צריך. אחרי זה היא מחזירה את המידע.

תיאור המודולים של מערכת התוכנה

המערכת משתמש במודולים שונים על מנת להריץ את האלמנטים השונים בתוכנה.

להלן אפרט כמה ממודולי המערכת:

- GameController.cs

מחלקה אשר מקשרת בין כל מחלקות השחקן ואחראית לניהול השוטף של המשחק.

- Client.cs

מחלקה אשר אחראית על הצד של הלקוח בתקשורת עם השרת.

- PokerChips.cs

מחלקה אשר אחראית על הפוקר צ'יפס ועל ההימורים.

- PlayerCash.cs

מחלקה אשר אחראית על ניהול כספו של השחקן.

- Pot.cs

מחלקה אשר אחראית על הכסף שבקופה.

- Global.cs

מחלקה המכילה פעולות גלובליות אשר כמה מהמחלקות האחרות משתמשות בהן.

- ServerTexasHoldem.py

מחלקה אשר אחראית על השרת.

תיעוד הקוד

בחלק זה אפרט על כל פונקציה שנמצאת בקוד.

בטבלאות הבאות (טבלה לכל מחלקה) יצוין שם הפונקציה, מה היא מקבלת, מה היא מחזירה ומה תפקידה.

השרת

ServerTexasHoldem.py

כותרת הפונקציה	טענת כניסה	טענת יציאה	תפקיד
def __init__(self, port)	מקבלת את הפורט	-	מתחילה את ההאזנה של השרת ומגדירה משתנים
def reset_properties(self, sec_to_wait)	מקבלת כמה שניות להמתין	-	מגדירה את המשתנים שצריכים להתאפס כל סיבוב
def accept_client(self, client)	מקבלת את הלקוח שמתחבר	-	מאשרת את חיבור הלקוח
def matched_start(self)	-	-	מתחילה את המשחק
def share_cards(self)	-	-	מחלקת את הקלפים האישיים
def flop_turn_river(self, cards_number)	מקבלת את מספר הקלפים	-	מחלקת את המספר הדרוש של קלפי הקהילה
def receive_data(self, client)	מקבלת את הלקוח ששלח מידע	-	מקבלת את המידע ופועלת לפיו
def send_data(self, client)	מקבלת את הלקוח שצריך לשלוח אליו מידע	-	שולחת ללקוח מידע
def exceptional(self, client)	מקבלת את הלקוח שיש אצלו אירוע חריג	-	מטפלת באירוע חריג

def main_loop(self)	-	-	לולאה תמידית שקוראת לפעולות: קבלת המידע, שליחת מידע ואירוע חריג
def evaluate_all_hands(self, hands)	רשימה של כל הידיים	מספר המנצח	מחשבת את המנצח ומחזירה אותו
def broadcast(self, data)	מקבלת מידע לשליחה	-	שולחת לכולם מידע מסוים
def broadcast_without_spectator(self, data)	מקבלת מידע לשליחה	-	שולחת לכולם חוץ מהצופה מידע מסוים
def generate_deck(self)	-	-	יוצרת חבילה סטנדרטית של 52 קלפים
def get_random_cards(self, cards_number)	מקבלת את מספר הקלפים	רשימת קלפים	מביאה מספר של קלפים אקראיים מתוך החבילה
def convert_mycard_to_eval7(self, card)	מקבלת קלף בייצוג שלי	קלף בייצוג אחר	ממירה בין הייצוג שלי לקלף לייצוג של הספריות eval7, holdem_calc
def generate_hand_eval7(self, cards)	מקבלת רשימה של קלפים	רשימת קלפים	ממירה "יד" מהייצוג שלי לייצוג של הספרייה eval7
def generate_hand_holdem_calc(self, cards)	מקבלת רשימה של קלפים	רשימת קלפים	ממירה "יד" מהייצוג שלי לייצוג של הספרייה holdem_calc

השחקן

GameConroller.cs

כותרת הפונקציה	טענת כניסה	טענת יציאה	תפקיד
GetLastRaise()	-	ההימור האחרון שבוצע	מחזירה את ההימור האחרון שבוצע
ResetLastRaise()	-	-	מאפסת את ההימור האחרון שבוצע
GetMessageFromServer()	-	-	מקבלת מידע מהשרת ושמה אותו במשתנה המתאים
PutCardsOnScreen(List<string> myCards, int cardsNum)	רשימת קלפים ומספר קלפים	-	מציגה על המסך את קלפי הקהילה
PutFirstCardsOnScreen(List<string> myCards, int cardsNum)	רשימת קלפים ומספר קלפים	-	מציגה על המסך את הקלפים האישיים של השחקן
PutOnePlayerCardsOnScreen(List<string> myCards, int playerNum)	רשימת קלפים ומספר שחקן	-	מציגה על המסך קלפים אישיים של שחקן מסוים
AddAnimatorComponent(GameObject gameObject, string animation)	אובייקט משחק, שם האנימציה	-	מוסיפה לאובייקט משחק קומפוננט של אנימציה
Call(int amount, int animNum)	כמות הכסף ומספר אנימציה	-	מבצעת הימור של שחקן מסוים
ConnectToServer()	-	-	מתחברת לשרת ומתחילה את המשחק
Start()	-	-	אתחול משתנים והתחלת המשחק

Update()	-	-	ניהול המשחק, הפעולה המרכזית שקוראת לרוב הפעולות
SetIsFold(bool isFold)	האם השחקן פרש	-	משנה את המשתנה שקובע אם השחקן פרש

Client.cs

כותרת הפונקציה	טענת כניסה	טענת יציאה	תפקיד
Connect()	-	מחרוזת של שחקן/צופה	מתחברת לשרת ומקבלת ממנו אם הלקוח הוא שחקן או צופה
StartClient()	-	-	אתחול משתנים
StartGame()	-	רשימה המכילה בעיקרה את מספרי השחקנים ואת כספם	מקבלת את הרשימה הראשונית מהשרת ומחזירה אותה
DoAction(string action, int cash)	פעולה של שחקן והימור	-	שולחת לשרת פעולה שביצע השחקן
SendData(Socket socket, List<object> lst)	Socket ורשימה לשליחה	מספר הבתים שנשלחו	שולחת לשרת רשימת אובייקטים מסוימת
Send(string msg)	מחרוזת	-	שולחת לשרת מחרוזת
ReceiveData(Socket socket)	Socket	ההודעה שהתקבלה בבתים	מקבלת הודעה מהשרת בבתים
ReturnData()	-	אובייקט	הפעולה המרכזית במחלקה: מקבלת משרת אובייקט כלשהו ומחזירה אותו
OnApplicationQuit()	-	-	סוגרת את החיבור כאשר המשחק נסגר

PokerChips.cs

כותרת הפונקציה	טענת כניסה	טענת יציאה	תפקיד
Call(int amount, int animNum)	כסף ומספר אנימציה	-	הפעולה המרכזית במחלקה: מבצעת הימור של שחקן כלשהו
CallWithoutAnim(int amount, int animNum)	כסף ומספר אנימציה	-	מבצעת הימור של שחקן כלשהו ללא הפעלת האנימציה
SetSpriteByAmount()	-	-	משנה את התמונה של האובייקט לפי גובה ההימור
ChangeAnimation(int animNum)	מספר אנימציה	-	משנה את האנימציה לפי האנימציה הנתונה
ChangeSpeed(float speed)	מהירות	-	משנה את מהירות האנימציה
GetAnimator()	-	ה Animator של המחלקה	מחזירה את ה Animator של המחלקה
Update()	-	-	מעלימה את האובייקט אם אין אנימציה שפועלת
Start()	-	-	אתחול משתנים

PlayerCash.cs

כותרת הפונקציה	טענת כניסה	טענת יציאה	תפקיד
CreateFile(string filename, int writtenValue)	שם הקובץ וכמה כסף לכתוב בו	מחרוזת של הכסף	יוצרת קובץ, כותבת בו את הכסף ומחזירה אותו כמחרוזת
OpenFile(string filename)	שם הקובץ	מחרוזת של הכסף	פותחת את הקובץ ומחזירה את הכסף
SetPlayerAmount()	-	-	הפעולה המרכזית במחלקה: קוראת לחלק מהפעולות כדי לקבוע את הכסף של השחקן
ChangeAmountFile(int newAmount)	כמות הכסף	-	משנה בקובץ את כמות הכסף של השחקן
AddAmount(int newAmount)	כמות הכסף	-	מוסיפה כסף לשחקן
ReduceAmount(int newAmount)	כמות הכסף	-	מורידה כסף לשחקן
UpdateText()	-	-	מעדכנת את הכסף במסך
GetAmount()	-	כמות הכסף	מחזירה כמה כסף יש השחקן
OnApplicationQuit()	-	-	כשהמשחק נסגר, מעדכנת את הקובץ

Pot.cs

כותרת הפונקציה	טענת כניסה	טענת יציאה	תפקיד
AddCash(int money)	כסף	-	מוסיפה כסף לקופה
EmptyPot()	-	כסף	מרוקנת את הקופה ומחזירה את סכום הכסף שהיה בה

CallButton.cs

כותרת הפונקציה	טענת כניסה	טענת יציאה	תפקיד
Call()	-	-	כאשר לוחצים על הכפתור Call, מתבצע Call

Check.cs

כותרת הפונקציה	טענת כניסה	טענת יציאה	תפקיד
SendCheck()	-	-	כאשר לוחצים על הכפתור Check, מתבצע Check

Fold.cs

כותרת הפונקציה	טענת כניסה	טענת יציאה	תפקיד
DoFold()	-	-	כאשר לוחצים על הכפתור Fold, מתבצע Fold

OpenSlider.cs

כותרת הפונקציה	טענת כניסה	טענת יציאה	תפקיד
OpenCashSlider()	-	-	כאשר לוחצים על הכפתור Raise, נפתח סליידר הכסף

CashSlider.cs

כותרת הפונקציה	טענת כניסה	טענת יציאה	תפקיד
Start()	-	-	אתחול משתנים
UpdateText()	-	-	מעדכנת את גובה הכסף במסך

ConfirmRaise.cs

כותרת הפונקציה	טענת כניסה	טענת יציאה	תפקיד
Confirm()	-	-	כאשר לוחצים על הכפתור Confirm, מתבצע Raise

StartGameButton.cs

כותרת הפונקציה	טענת כניסה	טענת יציאה	תפקיד
StartGame()	-	-	כאשר לוחצים על הכפתור Play, הפעולה טוענת את סצנת המשחק

Global.cs

כותרת הפונקציה	טענת כניסה	טענת יציאה	תפקיד
LoadPrefabFromFile(string filename)	שם הקובץ	האובייקט שהועלה	מעלה Prefab כלשהו מהתיקייה Assets
LoadSpriteFromFile(string filename)	שם הקובץ	ה Sprite שהועלה	מעלה Sprite כלשהו מהתיקייה Assets
CashToString(float cash)	כסף	מחרוזת של הכסף בתצוגה אחרת	מציגה את הכסף בתצוגה של K, M, B
StringCardToIntCard(string card)	מחרוזת של קלף	מספר המייצג את הקלף	ממירה קלף של תמונה למספר

השוואת העבודה עם פתרונות ויישומים קיימים

הפרויקט שלי, כפי שציינתי במבוא, מזכיר את המשחק Zynga Poker שהוא משחק מאוד מפורסם ומוצלח שכמה עשרות מיליוני אנשים משחקים בו. Zynga Poker הוא המשחק הרב משתתפים הכי מוצלח בז'אנר שלו.

הפרויקט שלי בסופו של דבר עושה את אותו הדבר, משחק טקסס הולדם רב משתתפים, אך יותר פשוט בלי יותר מידי אפשרויות נוספות כמו טורנירים, אירועים מיוחדים, חנויות וכדומה.

רק 5 שחקנים יכולים לשחק בעזרת אותו שרת לכן אם מספר גדול יותר של אנשים ירצה לשחק, יהיה צורך בלפתוח עוד שרתים עם פורטים או IP שונים.

לעומת המשחק Zynga Poke , המשחק שלי מופשט ומיועד למי שלא רוצה להתעסק בכל הדברים מסביב, כמו קנייה של דברים, או לחכות שמישהו אקראי יתחבר למשחק כדי להתחיל לשחק. אלא, במשחק שלי זה פשוט להתחבר עם חבר לאותו השרת וזהו, בלי לחכות ובלי כל הטורנירים והחנות. סך הכל משחק פשוט וכיף.

הערכת הפתרון לעומת התכנון והמלצות לשיפורו

בסופו של דבר, הפרויקט יצא משחק טקסס הולדם מרובה משתתפים כפי שהיה בתכנון. יחד עם זאת, עדיין ניתן לשפר אותו.

קודם כל, הייתי רוצה לאפשר ליותר אנשים לשחק בו זמנית במשחק שלי, ודבר זה דורש ליצור עוד שרתים.

אחד הדברים הנוספים שהייתי רוצה להוסיף בעתיד לפרויקט זה פרופיל לכל משתמש שכל המשתמשים יוכלו לראות, המכיל בין השאר כמה כסף יש לו, מספר ניצחונות והפסדים, סטטיסטיקות שונות עליו ועוד.

עוד דבר שהייתי רוצה להוסיף הוא שיהיו מספר "שולחנות", כאשר בכל שולחן יכולים לשחק שחקנים רק עם סכום כסף שעומד בטווח מסוים. בצורה כזו, לא ישחקו אחד נגד השני שחקנים עם פערים כספיים משמעותיים מדי שיפריעו למהלך התקין של המשחק.

תיאור של הממשק למשתמש – הוראות הפעלה

פתיחת השרת:

כדי להתחיל את השרת צריך להריץ את התוכנה שבו הוא כתוב.

פתיחת הלקוח:

המשתמש צריך להריץ את הקובץ של המשחק, ולאחר מכן יפתח מסך הפתיחה של המשחק, שבו יש את כפתור ה Play.

כדי להתחיל את המשחק צריך ללחוץ על הכפתור. אבל, עד שלא כל ששת המשתמשים (חמישה שחקנים וצופה אחד) יתחברו המשחק ימתין ולא יתחיל.

תחילת המשחק ומהלכו:

לאחר שכל המשתמשים התחברו, מתחיל המשחק.

ניתן לדעת אם אתה שחקן או צופה בקלות – שחקן יראה רק את הקלפים שלו ויהיו לו כפתורים נגישים. לעומת זאת, הצופה יראה את הקלפים של כולם והוא לא יראה שום כפתורים. לצופה גם יהיו רשומים אחוזי הניצחון של כל שחקן.

השחקן שזהו תורו ידע זאת מפני שהוא יוכל לראות שהכפתורים שלו פעילים (לא תמיד כולם. לדוגמא, אם לא בוצע הימור קודם הכפתור Call לא יהיה פעיל), כלומר ניתן ללחוץ עליהם. לעומת זאת, שחקן שזה אינו התור שלו, לא יוכל ללחוץ על הכפתורים.

כדי לשחק, כל שחקן בתורו צריך להחליט אם הוא מהמר (Raise או Call), לא מהמר (Check) או פורש מהסיבוב הנוכחי (Fold). כמובן שלא תמיד אפשר לעשות הכל וזה תלוי בחוקי המשחק. כדי לעשות זאת, כל מה שהשחקן צריך לעשות זה ללחוץ על הכפתור המתאים והפעולה שהוא רצה לעשות תתבצע.

בסיום הסיבוב, המשחק יכריז על המנצח, יביא לו את כל כספי ההימורים שבקופה ולאחר מכן, יתחיל סיבוב חדש מההתחלה.

ממשק המשתמש:

מסך הפתיחה:

עכשיו אציג איך נראה משחק, לדוגמא, של שני שחקנים וצופה.

כך זה נראה אצל השחקן שתורו:

כך זה נראה אצל הצופה:

כך זה נראה שיש הכרזה על המנצח (ניתן לראות את האנימציה של הפוקר צ'יפס. כאן היא לא זזה אבל במשחק כן)

מבט אישי על העבודה ותהליך הפיתוח

העבודה הייתה מאתגרת מאוד.

קודם כל, הייתי צריך ללמוד Unity ברמה שאוכל ליצור משחק אמיתי, רק בעזרת האינטרנט. הרבה מהלמידה שלי בתוכנה היא מהאינטרנט, שזה דבר מאתגר, אבל זה מה שאני אוהב בתוכנה – אני יכול להתקדם לבד כמה שאני רוצה וללמוד מה שאני רוצה בעזרת האינטרנט.

היו מספר קשיים בדרך שעיכבו אותי הרבה זמן ובחלק מהם אף לא הייתה שום תשובה או פתרון באינטרנט. לכן, נאלצתי לחקור דברים בעצמי ולהבין איך דברים עובדים ולחפש דברים שמזכירים את הבעיות באינטרנט, עד שהצלחתי לפתור את חלקם או "לעקוף" אותם.

לדוגמא, אחת הבעיות העיקריות שהיו לי הייתה לשלוט על האנימציה של הפוקר צ'יפס. הסתבכתי לא מעט עם ליצור את האנימציות ב Unity וגם אחר כך, הייתי צריך שתהיה לי יכולת להחליף אנימציה תוך כדי הריצה של התוכנית וגם זה גרם לא מעט בעיות.

בעיה נוספת שהייתה לי היא סנכרון המשחק בין כל המשתמשים. למרות שיש לי ידע בתקשורת מחשבים, זאת הייתה הפעם הראשונה שהייתי צריך להתמודד עם מספר רב יחסית של לקוחות אל מול שרת אחד וזה היה מאתגר.

אפשר לראות שהמשחק הזה לא כלל רק תוכנה אלא גם חשיבה יצירתית ולחשוב קצת מחוץ לקופסה – זה דבר שבעזרתו אפשר גם להצליח בכל תחום בחיים, ולא רק בתוכנה.

לסיום, למרות שהקשיים יכולים להיות מאוד מתישים, לא הייתי מוותר עליהם או על כל דבר אחר. הרי, מה יותר כיף מלהצליח לעשות משהו אחרי יושבים עליו הרבה זמן ושוברים עליו את הראש.

ביבליוגרפיה

1. https://answers.unity.com/index.html

2. https://stackoverflow.com/

3. https://forum.unity.com/

4. https://he.wikipedia.org/wiki/%D7%98%D7%A7%D7%A1%D7%A1_%D7%94%D7%95%D7%9C%D7%93%D7%9D

5. https://he.wikipedia.org/wiki/%D7%90%D7%9C%D7%99%D7%A4%D7%95%D7%AA_%D7%94%D7%A2%D7%95%D7%9C%D7%9D_%D7%91%D7%A4%D7%95%D7%A7%D7%A8

6. https://en.wikipedia.org/wiki/Zynga_Poker

7. https://docs.unity3d.com/ScriptReference/index.html

קוד התוכנית

GameConroller

```
using System;
using System.Collections.Generic;
using System.IO;
using UnityEngine;
using System.Net.Sockets;
using System.Text;
using System.Threading;
using Assets;
using UnityEngine.UI;
using Newtonsoft.Json;
using System.Collections;
using System.Linq;
using UnityEngine.SceneManagement;

public class GameController : MonoBehaviour
{
    //the cards in the game
    private GameObject[] gameCards;
    private string[] shapes;
    private int[] values;
    private int cardsCounter;
    //the red Xs that show that someone is fold
    public GameObject[] redXs;
    //the crown of the winner and its positions
    public GameObject crown;
    private readonly Vector2[] crownPos = { new Vector2(0, -4f), new Vector2(-9.65f,
0),
        new Vector2(-9.65f, 5.5f), new Vector2(9.45f, -5.5f), new Vector2(9.45f ,0) };
    public static GameController _instance;
    private Pot myPot;
    private PokerChips myPokerChips;
    private Client myClient;
    private PlayerCash myPlayerCash;
    private List<int> playersCash;
    public Text[] playersCashText;
    public Button[] myButtons;
    //the numbers of the players
    private List<int> playersNum;
    // the next cards of the game
    private List<string> nextCards;
    //an action of other player
    private List<object> playersAction;
    private string canPlay;
    private Vector2 posTableCards;
    private readonly Vector2[] posAllPlayersCards = { new Vector2(-2f, -6f), new
Vector2(-11.65f, -2f),
        new Vector2(-11.65f, 3.5f), new Vector2(7.45f, -7.5f), new Vector2(7.45f ,-2f)
};
    private int lastRaise;
    //does the game over
    private string isOver;
    //the player number of the winner
    private int winner;
    private bool isFold;
    //True if player False if spectator
```

```
    private bool isPlayer;
    //the winning rates
    private List<float> stats;
    public Text[] statsTexts;
    private List<int> foldPlayers;
    /// <summary>
    /// enable to other classes use this class
    /// </summary>
    public static GameController Instance
    {
        get
        {
            if (_instance == null)
            {
                GameObject gameObject =
GameObject.FindGameObjectWithTag("GameController");
                if(gameObject)
                    _instance = gameObject.GetComponent<GameController>();
            }
            return _instance;
        }
        set
        {
            _instance = value;
        }
    }
    public int GetLastRaise() { return lastRaise; }
    public void ResetLastRaise() { lastRaise = 0; }
    /// <summary>
    /// gets data from the server and assign it to the right variable. running all the
time in a different thread
    /// </summary>
    private void GetMessageFromServer()
    {
        while (true)
        {
            object message = myClient.ReturnData();
            if (message is List<string>)
                nextCards = (List<string>)message;
            else if (message is List<object>)
                playersAction = (List<object>)message;
            else if (message is List<float>)
                stats = ((List<float>)message).Skip(1).ToList();
            else
            {
                if ((string)message == "play")
                    canPlay = (string)message;
                else if ((string)message == "game over")
                    isOver = (string)message;
                else
                    winner = int.Parse((string)message);
            }
        }
    }
    /// <summary>
    /// puts cards on the table with an animation
    /// </summary>
    /// <param name="myCards"></param>
    /// <param name="cardsNum"></param>
    private void PutCardsOnScreen(List<string> myCards, int cardsNum)
    {
        for (int i = 0; i < cardsNum; i++)
```

```
        {
            var loadedPrefabResource = Global.LoadPrefabFromFile(myCards[i]);
            gameCards[i] = (GameObject)Instantiate(loadedPrefabResource,
posTableCards, Quaternion.identity);
            shapes[cardsCounter] = myCards[i].Substring(0, myCards[i].IndexOf('_'));
            values[cardsCounter] =
Convert.ToInt32(Global.StringCardToIntCard(myCards[i].Substring(myCards[i].IndexOf('_'
) + 1)));
            cardsCounter++;
            AddAnimatorComponent(gameCards[i], "FlipCard");
            posTableCards.x += 2.3f;
        }
    }
    /// <summary>
    /// puts the player's cards next to him with an animation
    /// </summary>
    /// <param name="myCards"></param>
    /// <param name="cardsNum"></param>
    private void PutFirstCardsOnScreen(List<string> myCards, int cardsNum)
    {
        Vector2 pos = new Vector2(-2f, -6f);
        for (int i = 0; i < cardsNum; i++)
        {
            var loadedPrefabResource = Global.LoadPrefabFromFile(myCards[i]);
            gameCards[i] = (GameObject)Instantiate(loadedPrefabResource, pos,
Quaternion.identity);
            shapes[cardsCounter] = myCards[i].Substring(0, myCards[i].IndexOf('_'));
            values[cardsCounter] =
Convert.ToInt32(Global.StringCardToIntCard(myCards[i].Substring(myCards[i].IndexOf('_'
) + 1)));
            cardsCounter++;
            AddAnimatorComponent(gameCards[i], "FlipCard");
            pos.x += 2.3f;
        }
    }
    /// <summary>
    /// puts other player's cards next to him with an animation
    /// </summary>
    /// <param name="myCards"></param>
    /// <param name="playerNum"></param>
    private void PutOnePlayerCardsOnScreen(List<string> myCards, int playerNum)
    {
        for (int i = 0; i < 2; i++)
        {
            var loadedPrefabResource = Global.LoadPrefabFromFile(myCards[i]);
            gameCards[i] = (GameObject)Instantiate(loadedPrefabResource,
posAllPlayersCards[playerNum], Quaternion.identity);
            AddAnimatorComponent(gameCards[i], "FlipCard");
            posAllPlayersCards[playerNum].x += 2.3f;
        }
    }
    /// <summary>
    /// adds to a gameobject an animation component
    /// </summary>
    /// <param name="gameObject"></param>
    /// <param name="animation"></param>
    public void AddAnimatorComponent(GameObject gameObject, string animation)
    {
        Animator animator = gameObject.AddComponent<Animator>();
        animator.runtimeAnimatorController =
(RuntimeAnimatorController)Resources.Load(/*"Levels/" + */animation);
    }
```

```
    /// <summary>
    /// does a call/raise
    /// </summary>
    /// <param name="amount"></param>
    /// <param name="animNum"></param>
    private void Call(int amount, int animNum)
    {
        Debug.Log(animNum);
        //calls the poker chips
        myPokerChips.Call(amount, animNum);
        //add money to the pot
        myPot.AddCash(amount);
        //reduce the money from the player
        if (isPlayer)
        {
            playersCash[animNum - 1] -= amount;
            playersCashText[animNum - 1].text =
Global.CashToString(playersCash[animNum - 1]) + " $";
        }
        else
        {
            playersCash[animNum] -= amount;
            playersCashText[animNum].text = Global.CashToString(playersCash[animNum])
+ " $";
        }
    }
    /// <summary>
    /// connects to the server and define the variables
    /// </summary>
    private void ConnectToServer()
    {
        myClient.StartClient();
        isPlayer = myClient.Connect() == "player";
        if (isPlayer)
        {
            //if player set its money and send it to the server
            Debug.Log("player");
            myPlayerCash.SetPlayerAmount();
            myClient.Send(myPlayerCash.GetAmount().ToString());
            playersCashText = playersCashText.Skip(1).ToArray();
        }
        //if spectator disable the buttons
        else
            foreach (Button button in myButtons)
                button.gameObject.SetActive(false);
        List<object> data = myClient.StartGame();
        playersNum = (List<int>)data[2];
        playersCash = (List<int>)data[3];
    }
    void Awake()
    {
        Instance = this;
    }
    private void Start()
    {
        //initialize the variables
        isFold = false;
        stats = null;
        winner = -1;//start the winner with -1 (nobody won)
        cardsCounter = 0;
        shapes = new string[7];
        values = new int[7];
```

```
        isOver = null;
        nextCards = null;
        playersAction = null;
        canPlay = null;
        lastRaise = 0;
        myPot = Pot.Instance;
        myPokerChips = PokerChips.Instance;
        myClient = Client.Instance;
        myPlayerCash = PlayerCash.Instance;
        playersCash = new List<int>();
        foldPlayers = new List<int>();
        gameCards = new GameObject[5];
        posTableCards = new Vector2(-4.6f, 0f);
        //connecting to the server
        ConnectToServer();
        //show the amount of money of the all players
        for (int i = 0; i < playersCash.Count; i++)
            playersCashText[i].text = Global.CashToString(playersCash[i]) + " $";
        if (isPlayer)
            //puts the player's cards
            PutFirstCardsOnScreen((List<string>)myClient.ReturnData(), 2);
        else
            //puts all the players cards
            for (int i = 0; i < playersNum.Count - 1; i++)
                PutOnePlayerCardsOnScreen((List<string>)myClient.ReturnData(), i);
        //if there are less than 5 players, put red X on them
        if (isPlayer)
            for (int i = 4; i >= playersNum.Count; i--)
                redXs[i].SetActive(true);
        else
            for (int i = 4; i >= playersNum.Count - 1; i--)
                redXs[i].SetActive(true);
        //start the thread of the communication
        Thread thread = new Thread(new ThreadStart(GetMessageFromServer));
        thread.Start();
    }
    private void Update()
    {
        string stateName = null;
        //put next table cards on the table
        if (nextCards != null)
        {
            PutCardsOnScreen(nextCards, nextCards.Count);
            if (isPlayer)
                lastRaise = 0;
            nextCards = null;
        }
        //does an action of other player
        if (playersAction != null)
        {
            //the action is raise or call
            if ((string)playersAction[0] == "raise" || (string)playersAction[0] ==
"call")
            {
                if (isPlayer)
                    if ((string)playersAction[0] == "raise")
                        lastRaise = Convert.ToInt32(playersAction[1]);
                Call(Convert.ToInt32(playersAction[1]),
playersNum.IndexOf(Convert.ToInt32(playersAction[2])));
            }
            //the action is fold
            else if ((string)playersAction[0] == "fold")
```

```
            {
redXs[playersNum.IndexOf(Convert.ToInt32(playersAction[2]))].SetActive(true);
foldPlayers.Add(playersNum.IndexOf(Convert.ToInt32(playersAction[2])));
            }
            playersAction = null;
        }
        if (isPlayer && canPlay != null)
        {
            //this is the player's turn. enable the buttons
            if (lastRaise == 0)
                foreach (Button button in myButtons)
                {
                    if (button.name != "CallButton")
                        button.interactable = true;
                }
            else
                foreach (Button button in myButtons)
                {
                    if (button.name != "CheckButton")
                        button.interactable = true;
                }
            canPlay = null;
        }
        if(isPlayer && !isFold && isOver != null)
        {
            //the game is over. disable the buttons and send the player's hand to the
server for evaluation
            foreach (Button button in myButtons)
                button.interactable = false;
            List<string> hand = new List<string>();
            for (int i = 0; i < shapes.Length; i++)
                if (values[i] != 0)
                    hand.Add(shapes[i] + '_' + values[i]);
            myClient.Send(JsonConvert.SerializeObject(hand));
            isOver = null;
        }
        if(winner > -1)
        {
            //the server found the winner
            int winnerIndex = playersNum.IndexOf(winner);
            Debug.Log(winnerIndex);
            //set the crown above the winner
            crown.transform.position = crownPos[winnerIndex];
            crown.SetActive(true);
            //give the winner his money with an animation
            int potCash = myPot.EmptyPot();
            Debug.Log("The winner gets: " + potCash);
            myPokerChips.CallWithoutAnim(potCash, winnerIndex);
            stateName = "PokerChips" + winnerIndex + " R";
            myPokerChips.GetAnimator().Play(stateName);
            if (isPlayer)
                if (winnerIndex == 0) myPlayerCash.AddAmount(potCash);
                else
                {
                    playersCash[winnerIndex - 1] += potCash;
                    playersCashText[winnerIndex - 1].text =
Global.CashToString(playersCash[winnerIndex - 1]) + " $";
                }
            else
            {
```

```
                playersCash[winnerIndex] += potCash;
                playersCashText[winnerIndex].text =
Global.CashToString(playersCash[winnerIndex]) + " $";
            }
            winner = -1;
            Debug.Log(myPlayerCash.GetAmount());
            //restart the game by loading the same scene. before of that changing the
amount of money of the player in the file
            myPlayerCash.ChangeAmountFile(myPlayerCash.GetAmount());
            Thread.Sleep(5000);
            SceneManager.LoadScene("Game");
        }
        if (!isPlayer && stats != null)
        {
            //show the winning rate of all the players
            int statsIndex = 0;
            for (int i = 0; i < stats.Count + foldPlayers.Count; i++)
            {
                //if player is fold he has obviously no chance to win
                while (foldPlayers.Contains(i))
                {
                    statsTexts[i].text = "Win: 0%";
                    i++;
                }
                if (i < stats.Count + foldPlayers.Count)
                    statsTexts[i].text = "Win: " + Math.Round(stats[statsIndex++] *
100).ToString() + "%";
            }
            stats = null;
        }
    }
    public void SetIsFold(bool isFold) { this.isFold = isFold; }
```

Client

```
using System;
using System.Collections;
using System.Collections.Generic;
using System.Net;
using System.Net.Sockets;
using System.Text;
using UnityEngine;
using Newtonsoft.Json;
/// <summary>
/// responsible on the client in the communication
/// </summary>
public class Client : MonoBehaviour
{
    //the buffer bytes in the receive
    private byte[] bufferBytes;
    //define ip settings and socket
    private IPHostEntry ipHostInfo;
    private IPAddress ipAddress;
    private IPEndPoint remoteEP;
    private Socket sender;
    public static Client _instance;
    /// <summary>
    /// enable to other classes use this class
    /// </summary>
    public static Client Instance
    {
        get
        {
            if (_instance == null)
            {
                GameObject gameObject = GameObject.FindGameObjectWithTag("Client");
                if (gameObject)
                    _instance = gameObject.GetComponent<Client>();
            }
            return _instance;
        }
        set
        {
            _instance = value;
        }
    }
    //connects to the server and return the string that the server sent (player or
spectator)
    public string Connect()
    {
        sender.Connect(remoteEP);
        Debug.Log("Socket connected to " + sender.RemoteEndPoint.ToString());
        int bytesRec = ReceiveData(sender);
        return Encoding.ASCII.GetString(bufferBytes, 0, bytesRec);
    }
    /// <summary>
    /// initialize variables
    /// </summary>
    public void StartClient()
    {
        bufferBytes = new byte[1024];
        ipHostInfo = Dns.GetHostEntry(Dns.GetHostName());
        ipAddress = ipHostInfo.AddressList[1];
```

```
        Debug.Log(ipAddress);
        remoteEP = new IPEndPoint(ipAddress, 2222);
        sender = new Socket(ipAddress.AddressFamily, SocketType.Stream,
ProtocolType.Tcp);
    }
    /// <summary>
    /// return from the server the list which include mainly the number of the players
and their money. than the game start
    /// </summary>
    /// <returns></returns>
    public List<object> StartGame()
    {
        int bytesRec = ReceiveData(sender);
        //deserialize the object
        string message = Encoding.ASCII.GetString(bufferBytes, 0, bytesRec);
        List<object> data = JsonConvert.DeserializeObject<List<object>>(message);
        //convert to list of ints
        data[2] = ((Newtonsoft.Json.Linq.JArray)data[2]).ToObject<List<int>>();
        data[3] = ((Newtonsoft.Json.Linq.JArray)data[3]).ToObject<List<int>>();
        return data;
    }
    /// <summary>
    /// send the action that has made to the sever
    /// </summary>
    /// <param name="action"></param>
    /// <param name="cash"></param>
    public void DoAction(string action, int cash)
    {
        List<object> data = new List<object>(new object[] { action, cash });
        SendData(sender, data);
    }
    /// <summary>
    /// send a list of objects to the server
    /// </summary>
    /// <param name="socket"></param>
    /// <param name="lst"></param>
    /// <returns></returns>
    private int SendData(Socket socket, List<object> lst)
    {
        //serialize the list and sending it
        string msg = JsonConvert.SerializeObject(lst);
        byte[] data = Encoding.ASCII.GetBytes(msg);
        int bytesSent = socket.Send(data);
        return bytesSent;
    }
    /// <summary>
    /// send a string to the server
    /// </summary>
    /// <param name="msg"></param>
    public void Send(string msg)
    {
        //serialize the string and sending it
        byte[] data = Encoding.ASCII.GetBytes(msg);
        int bytesSent = sender.Send(data);
    }
    /// <summary>
    /// recieve data from the server
    /// </summary>
    /// <param name="socket"></param>
    /// <returns></returns>
    private int ReceiveData(Socket socket)
    {
```

```
        int bytesRec = socket.Receive(bufferBytes);
        return bytesRec;
    }
    /// <summary>
    /// return data from the server: can be list of objects, list of strings, list of
floats or a string
    /// </summary>
    /// <returns></returns>
    public object ReturnData()
    {
        //get the data
        int bytesRec = ReceiveData(sender);
        string message = Encoding.ASCII.GetString(bufferBytes, 0, bytesRec);
        Debug.Log(message);
        try
        {
            //trying to convert it to different kinds of lists
            List<object> data = JsonConvert.DeserializeObject<List<object>>(message);
            if (data.Count == 1)
                return
JsonConvert.DeserializeObject<List<string>>(JsonConvert.SerializeObject(data));
            else if (data[1] is string)
                return
JsonConvert.DeserializeObject<List<string>>(JsonConvert.SerializeObject(data));
            else if(data[1] is double)
                return
JsonConvert.DeserializeObject<List<float>>(JsonConvert.SerializeObject(data));
            return data;
        }
        catch
        {
            //the data is a string
            return message;
        }
    }
    private void Awake()
    {
        Instance = this;
    }
    /// <summary>
    /// close the client when the game is ending
    /// </summary>
    private void OnApplicationQuit()
    {
        try
        {
            sender.Shutdown(SocketShutdown.Both);
            sender.Close();
        }
        catch(Exception e)
        {
            Debug.Log(e.ToString());
        }
    }
}
```

PlayerCash

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.IO;
using System.Text;
using UnityEngine.UI;
using Assets;
/// <summary>
/// this class is responsible to the player's money
/// </summary>
public class PlayerCash : MonoBehaviour
{
    //The amount of money of the player
    private static int amount;
    //The path of the file which saves the amount of money
    private const string path = "AmountOfPokerChips.txt";
    public Text myText;
    public static PlayerCash _instance;
    /// <summary>
    /// enable to other classes use this class
    /// </summary>
    public static PlayerCash Instance
    {
        get
        {
            if (_instance == null)
            {
                GameObject gameObject =
GameObject.FindGameObjectWithTag("PlayerCash");
                if (gameObject)
                    _instance = gameObject.GetComponent<PlayerCash>();
            }
            return _instance;
        }
        set
        {
            _instance = value;
        }
    }
    /// <summary>
    /// creating the file which saves the money
    /// </summary>
    /// <param name="filename"></param>
    /// <param name="writtenValue"></param>
    /// <returns></returns>
    private string CreateFile(string filename, int writtenValue)
    {
        //remove exist file
        if (File.Exists(filename))
            File.Delete(filename);
        //create the file and write the money in it
        using (FileStream fs = File.Create(filename))
        {
            byte[] text = new UTF8Encoding(true).GetBytes(writtenValue.ToString());
            fs.Write(text, 0, text.Length);
        }
        //open the file and return the money in it
        using (StreamReader sr = File.OpenText(filename))
```

```
        {
            string s = "", text = "";
            while ((s = sr.ReadLine()) != null)
                text += s;
            return text;
        }
    }
    /// <summary>
    /// open the file that saves the money
    /// </summary>
    /// <param name="filename"></param>
    /// <returns></returns>
    private string OpenFile(string filename)
    {
        //open the file and return the money in it
        using (StreamReader sr = File.OpenText(filename))
        {
            string s = "", text = "";
            while ((s = sr.ReadLine()) != null)
                text += s;
            return text;
        }
    }
    /// <summary>
    /// get the player's amount from the file. if the player is new, give him 2
million
    /// </summary>
    public void SetPlayerAmount()
    {
        //get amount from exist file
        try
        {
            string stringAmount = OpenFile(path);
            amount = int.Parse(stringAmount);
        }
        //create new file and give 2 million
        catch
        {
            string stringAmount = CreateFile(path, 2000000);
            amount = int.Parse(stringAmount);
        }
        if (amount <= 0)
            amount = 2000000;
        Debug.Log(amount);
        UpdateText();
    }
    /// <summary>
    /// change the amount of money in the file
    /// </summary>
    /// <param name="newAmount"></param>
    public void ChangeAmountFile(int newAmount)
    {
        string stringAmount = CreateFile(path, newAmount);
        amount = int.Parse(stringAmount);
    }
    /// <summary>
    /// increase the player's money
    /// </summary>
    /// <param name="newAmount"></param>
    public void AddAmount(int newAmount)
    {
        amount += newAmount;
```

```
        UpdateText();
    }
    /// <summary>
    /// decrease the player's money
    /// </summary>
    /// <param name="newAmount"></param>
    public void ReduceAmount(int newAmount)
    {
        amount -= newAmount;
        UpdateText();
    }
    /// <summary>
    /// update the money text
    /// </summary>
    private void UpdateText()
    {
        myText.text = Global.CashToString(amount) + " $";
    }
    public int GetAmount()
    {
        return amount;
    }
    private void Awake()
    {
        Instance = this;
    }
    /// <summary>
    /// change the amount of money in the file when the game is ending.
    /// </summary>
    private void OnApplicationQuit()
    {
        ChangeAmountFile(amount);
    }
}
```

PokerChips

```
using System.IO;
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using System.Text;
using Assets;
using UnityEngine.UI;
/// <summary>
/// responsible on the poker chips
/// </summary>
public class PokerChips : MonoBehaviour
{
    //the amount of money
    private int amount;
    public Text myText;
    private SpriteRenderer mySpriteRenderer;
    private Animator myAnimator;
    //the clips of the animations
    public RuntimeAnimatorController[] aoc;
    public static PokerChips _instance;
    /// <summary>
    /// enable to other classes use this class
    /// </summary>
    public static PokerChips Instance
    {
        get
        {
            if (_instance == null)
            {
                GameObject gameObject =
GameObject.FindGameObjectWithTag("PokerChips");
                if (gameObject)
                    _instance = gameObject.GetComponent<PokerChips>();
            }
            return _instance;
        }
        set
        {
            _instance = value;
        }
    }
    /// <summary>
    /// does a call/raise from given amount and number of animation
    /// </summary>
    /// <param name="amount"></param>
    /// <param name="animNum"></param>
    public void Call(int amount, int animNum)
    {
        this.amount = amount;
        //define the sprite relatively the amount of money
        SetSpriteByAmount();
        myText.GetComponent<Transform>().position = transform.position;
        myText.text = Global.CashToString(amount) + " $";
        //change the animation to the right animation
        ChangeAnimation(animNum);
        //starts the animation
        myAnimator.SetTrigger("Trigger");
    }
```

```
    /// <summary>
    /// does a call/raise just like before, but does not play the animation
    /// </summary>
    /// <param name="amount"></param>
    /// <param name="animNum"></param>
    public void CallWithoutAnim(int amount, int animNum)
    {
        this.amount = amount;
        SetSpriteByAmount();
        myText.GetComponent<Transform>().position = transform.position;
        myText.text = Global.CashToString(amount) + " $";
        ChangeAnimation(animNum);
    }
    /// <summary>
    /// define the sprite relatively the amount of money
    /// </summary>
    private void SetSpriteByAmount()
    {
        if (amount < 500000)
            mySpriteRenderer.sprite = Global.LoadSpriteFromFile("amount_1");
        else if (amount >= 500000 && amount < 1000000)
            mySpriteRenderer.sprite = Global.LoadSpriteFromFile("amount_2");
        else if (amount >= 1000000)
            mySpriteRenderer.sprite = Global.LoadSpriteFromFile("amount_3");
    }
    public void ChangeAnimation(int animNum)
    {
        myAnimator.runtimeAnimatorController = aoc[animNum];
    }
    public void ChangeSpeed(float speed)
    {
        myAnimator.SetFloat("animSpeed", speed);
    }
    public Animator GetAnimator() { return myAnimator; }
    private void Update()
    {
        //hide the sprite if their is no animation that running
        if (myAnimator.IsInTransition(0) &&
myAnimator.GetNextAnimatorStateInfo(0).IsName("Idle"))
        {
            mySpriteRenderer.sprite = null;
            myText.text = "";
        }
    }
    private void Start()
    {
        //initialize variables
        amount = 0;
        mySpriteRenderer = GetComponent<SpriteRenderer>();
        myAnimator = GetComponent<Animator>();
    }
    private void Awake()
    {
        Instance = this;
    }
}
```

Pot

```
using UnityEngine;
using UnityEngine.UI;
using Assets;
/// <summary>
/// responsible on the pot
/// </summary>
public class Pot : MonoBehaviour
{
    //the money in the pot
    private int cash;
    public Text potText;
    public static Pot _instance;
    /// <summary>
    /// enable to other classes use this class
    /// </summary>
    public static Pot Instance
    {
        get
        {
            if (_instance == null)
            {
                GameObject gameObject = GameObject.FindGameObjectWithTag("Pot");
                if (gameObject)
                    _instance = gameObject.GetComponent<Pot>();
            }
            return _instance;
        }
        set
        {
            _instance = value;
        }
    }
    /// <summary>
    /// adds money to the pot
    /// </summary>
    /// <param name="money"></param>
    public void AddCash(int money)
    {
        cash += money;
        potText.text = Global.CashToString(cash) + " $";
    }
    /// <summary>
    /// empty the pot and return the money in it before that
    /// </summary>
    /// <returns></returns>
    public int EmptyPot()
    {
        int originalCash = cash;
        Debug.Log("cash is: " + originalCash);
        cash = 0;
        potText.text = "0 $";
        return originalCash;
    }
    private void Awake()
    {
        Instance = this;
    }
}
```

CallButton

```
using UnityEngine;
using UnityEngine.UI;
/// <summary>
/// responsible on the call button
/// </summary>
public class CallButton : MonoBehaviour
{
    public Button[] myButtons;
    public PokerChips myPokerChips;
    private Pot myPot;
    private PlayerCash myPlayerCash;
    private Client myClient;
    private GameController myGameController;
    public void Call()
    {
        myGameController = GameController.Instance;
        //get the last raise - how much money is the call
        int lastRaise = myGameController.GetLastRaise();
        myClient = Client.Instance;
        myPot = Pot.Instance;
        myPlayerCash = PlayerCash.Instance;
        myPot.AddCash(lastRaise);
        myPlayerCash.ReduceAmount(lastRaise);
        //send the action to the server
        myClient.DoAction("call", lastRaise);
        //do the animation
        myPokerChips.Call(lastRaise, 0);
        //disable the buttons
        foreach (Button button in myButtons)
            button.interactable = false;
        myGameController.ResetLastRaise();
    }
}
```

Check

```
using UnityEngine;
using UnityEngine.UI;
/// <summary>
/// responsible on the check button
/// </summary>
public class Check : MonoBehaviour
{
    public Button[] myButtons;
    private Client myClient;
    /// <summary>
    /// send the action to the server and disable the buttons
    /// </summary>
    public void SendCheck()
    {
        myClient = Client.Instance;
        myClient.DoAction("check", 0);
        foreach (Button button in myButtons)
            button.interactable = false;
    }
}
```

CashSlider

```
using UnityEngine;
using UnityEngine.UI;
using Assets;
/// <summary>
/// responsible on what happens when the player moving the cash slider
/// </summary>
public class CashSlider : MonoBehaviour
{
    private Text myText;
    private Slider mySlider;
    private PlayerCash myPlayerCash;
    /// <summary>
    /// updates the text (the money) above the slider
    /// </summary>
    public void UpdateText()
    {
        myText.text = Global.CashToString(mySlider.value) + " $";
    }
    void Start()
    {
        //reset variables
        myPlayerCash = PlayerCash.Instance;
        mySlider = GetComponent<Slider>();
        myText = GetComponentInChildren<Text>();
        mySlider.maxValue = myPlayerCash.GetAmount();
    }
}
```

ConfirmRaise

```
using UnityEngine;
using UnityEngine.UI;
/// <summary>
/// responsible on the confirm button of the cash slider
/// </summary>
public class ConfirmRaise : MonoBehaviour
{
    public Button[] myButtons;
    public Slider mySlider;
    public PokerChips myPokerChips;
    private Pot myPot;
    private PlayerCash myPlayerCash;
    private Client myClient;
    private GameController myGameController;
    public void Confirm()
    {
        if (mySlider.value > 0)
        {
            //initialize the variables
            myGameController = GameController.Instance;
            myClient = Client.Instance;
            myPot = Pot.Instance;
            myPlayerCash = PlayerCash.Instance;
            myPot.AddCash((int)mySlider.value);
            myPlayerCash.ReduceAmount((int)mySlider.value);
            //send the action to the server
            myClient.DoAction("raise", (int)mySlider.value);
            //do the animation
            myPokerChips.Call((int)mySlider.value, 0);
            //disable the slider
            mySlider.gameObject.SetActive(false);
            mySlider.value = 0;
            //disable the buttons
            foreach (Button button in myButtons)
                button.interactable = false;
            myGameController.ResetLastRaise();
        }
    }
}
```

Fold

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
/// <summary>
/// responsible on the fold button
/// </summary>
public class Fold : MonoBehaviour
{
    public GameObject redX;
    public Button[] myButtons;
    public Slider mySlider;
    private Client myClient;
    private GameController myGameController;
    public void DoFold()
    {
        //initialize the variables
        myGameController = GameController.Instance;
        myClient = Client.Instance;
        myGameController.SetIsFold(true);
        //disable the slider
        mySlider.gameObject.SetActive(false);
        //enable the red X
        redX.SetActive(true);
        //disable the buttons
        foreach(Button button in myButtons)
            button.interactable = false;
        //send the action to the server
        myClient.DoAction("fold", 0);
    }
}
```

OpenSlider

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.UI;
/// <summary>
/// responsible on the fold button
/// </summary>
public class Fold : MonoBehaviour
{
    public GameObject redX;
    public Button[] myButtons;
    public Slider mySlider;
    private Client myClient;
    private GameController myGameController;
    public void DoFold()
    {
        //initialize the variables
        myGameController = GameController.Instance;
        myClient = Client.Instance;
        myGameController.SetIsFold(true);
        //disable the slider
        mySlider.gameObject.SetActive(false);
        //enable the red X
        redX.SetActive(true);
        //disable the buttons
        foreach(Button button in myButtons)
            button.interactable = false;
        //send the action to the server
        myClient.DoAction("fold", 0);
    }
}
```

Global

```
using System.Collections.Generic;
using System.IO;
using System.Linq;
using System.Text;
using System.Threading.Tasks;
using UnityEngine;

namespace Assets
{
    /// <summary>
    /// global class with gloabal methods
    /// </summary>
    public class Global
    {
        /// <summary>
        /// loads a prefab from the Assets file
        /// </summary>
        /// <param name="filename"></param>
        /// <returns></returns>
        public static Object LoadPrefabFromFile(string filename)
        {
            Object loadedObject = Resources.Load(/*"Levels/" + */filename);
            if (loadedObject == null)
            {
                throw new FileNotFoundException("...no file found - please check the
configuration");
            }
            return loadedObject;
        }
        /// <summary>
        /// loads a Sprite from the Assets file
        /// </summary>
        /// <param name="filename"></param>
        /// <returns></returns>
        public static Sprite LoadSpriteFromFile(string filename)
        {
            Sprite loadedObject = Resources.Load<Sprite>(/*"Levels/" + */filename);
            if (loadedObject == null)
            {
                throw new FileNotFoundException("...no file found - please check the
configuration");
            }
            return loadedObject;
        }
        /// <summary>
        /// converts money to the pattern of K,M,B
        /// </summary>
        /// <param name="cash"></param>
        /// <returns></returns>
        public static string CashToString(float cash)
        {
            if (cash < 1000f)
                return ((int)cash).ToString();
            if (cash >= 1000f && cash < 1000000f)
                return (cash / 1000f).ToString("F1") + "K";
            if (cash >= 1000000f && cash < 1000000000f)
                return (cash / 1000000f).ToString("F2") + "M";
            if (cash >= 1000000000f && cash < 1000000000000f)
```

```
                return (cash / 1000000000f).ToString("F3") + "B";
            return "1000B+";
        }
        /// <summary>
        /// convert special cards to ints
        /// </summary>
        /// <param name="card"></param>
        /// <returns></returns>
        public static int StringCardToIntCard(string card)
        {
            if (card == "J")
                return 11;
            else if (card == "Q")
                return 12;
            else if (card == "K")
                return 13;
            else if (card == "A")
                return 14;
            else
                return int.Parse(card);
        }
    }
}
```

StartGameButton

```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;
using UnityEngine.SceneManagement;
/// <summary>
/// responsible on the play button
/// </summary>
public class StartGameButton : MonoBehaviour
{
    /// <summary>
    /// load the game scene
    /// </summary>
    public void StartGame()
    {
        SceneManager.LoadScene("Game");
    }
}
```

ServerTexasHoldem

```
import socket
import select
import queue
import json
import secrets
import time
import eval7
import holdem_calc
import threading

class Server(object):

    def __init__(self, port):
        #initialize communication varialbes
        self.server = socket.socket(socket.AF_INET,socket.SOCK_STREAM)
        self.server.setblocking(0)
        self.server_address = ('0.0.0.0', port)
        print('starting up on ' , self.server_address[0] , ' port ' ,
self.server_address[1])
        self.server.bind(self.server_address)
        self.server.listen(6)
        self.inputs = [self.server]
        #initialize other variables
        self.reset_properties(0)
        self.shapes = ['Club', 'Diamond', 'Heart', 'Spade']
        self.values = ['A','2','3','4','5','6','7','8','9', '10', 'J','Q','K']
        #number of clients that can connect to the server and play
        self.max_num_clients = 3

    #initialize some variables that need to reset every round of Texas Holdem
    def reset_properties(self, sec_to_wait):
        time.sleep(sec_to_wait)
        self.outputs = []
        self.message_queues = {}
        self.players = {}
```

```
        self.fold_players = []
        self.players_cash = []
        self.players_counter = 0
        #table cards counter
        self.cards_counter = 0
        self.deck = []
        #all the cards in the game
        self.cards = []
        #only the private cards and without the fold players cards
        self.cards_without_fold_cards = []
        self.over = False
        self.hands = []
        self.winner = 0
        self.dont_send_to_spec = False

    #accept client. send him player/spectator and recv from him his money
    def accept_client(self, client):
        connection, self.client_address = client.accept()
        print('new connection from ' , self.client_address)
        connection.setblocking(1)
        #only the last client is the spectator
        if self.players_counter < self.max_num_clients - 1:
            print('player')
            connection.send('player'.encode())
            #recv the money of the player
            self.players_cash.append(json.loads(connection.recv(1024)))
        else:
            print('spectator')
            connection.send('spectator'.encode())
        connection.setblocking(0)
        self.inputs.append(connection)
        self.message_queues[connection] = queue.Queue()
        self.players[connection] = self.players_counter
        self.players_counter += 1
        self.matched_start()
```

```
    #after all the clients have connected, start the game

    def matched_start(self):

        #start the game only if all the clients have connected

        if self.players_counter == self.max_num_clients:

            print(self.players_cash)

            cnt = 2

            players_list = list(self.players.values())

            #send all the clients that the game started and a list of neccessary
things for the game: player numbers, players money

            for player in self.players:

                if self.players[player] == self.max_num_clients - 1:

                    player.send(json.dumps(['game started','Spectator', players_list,
self.players_cash]).encode())

                elif self.players[player] == 0:

                    player.send(json.dumps(['game started','sb', players_list[:-1],
self.players_cash[1:]]).encode())

                elif self.players[player] == 1:

                    player.send(json.dumps(['game started','bb', players_list[1:-1] +
players_list[0::-1],

                                            (self.players_cash[1:] +
self.players_cash[0::-1])[1:]]).encode())

                else:

                    player.send(json.dumps(['game started','None',players_list[cnt:-1]
+ players_list[cnt-1::-1],

                                            (self.players_cash[cnt:] +
self.players_cash[cnt-1::-1])[1:]]).encode())

                    cnt += 1

            #the last player of a gambling round

            self.last_player = list(self.players)[-2]

            self.generate_deck()

            #send the private cards to the clients

            self.share_cards()

            self.cards_without_fold_cards = self.cards.copy()

            #send the first 3 community cards to the clients

            self.flop_turn_river(3)

            #send play to the first player

            self.message_queues[list(self.players)[0]].put('play'.encode())

            #send the winning rates to the spectator
```

```
            stats =
holdem_calc.calculate(self.generate_hand_holdem_calc(self.cards[(self.players_counter
- 1) * 2:]), True, 1, None,


self.generate_hand_holdem_calc(self.cards[:(self.players_counter - 1) * 2]), False)

            print(stats)

            self.message_queues[list(self.players)[-
1]].put(json.dumps(stats).encode())

            print('The game has started')


    #send the private cards to the clients

    def share_cards(self):

        #the loop doesn't include the spectator because he gets all the cards

        for player in list(self.players)[:-1]:

            self.cards += self.get_random_cards(2)

            data = self.cards[-2:]

            self.message_queues[player].put(json.dumps(data).encode())

            #the spectator gets all the cards

            self.message_queues[list(self.players)[-1]].put(json.dumps(data).encode())

            if player not in self.outputs:

                self.outputs.append(player)


    #send all the clients the community cards by given the number of cards

    def flop_turn_river(self, cards_number):

        flop_turn_river_cards = self.get_random_cards(cards_number)

        self.cards += flop_turn_river_cards

        #broadcast all the clients the cards

        self.broadcast(json.dumps(flop_turn_river_cards).encode())

        self.cards_counter += cards_number


    #in fact, this is the main method of the game. it receives the action of a client
and respond respectively

    def receive_data(self, client):

        data = client.recv(1024)

        if data:

            #the data has to be a list

            data = json.loads(data)

            if not self.over:

                #the action is fold
```

```
                if data[0] == 'fold':
                    self.fold_players.append(self.players[client])
                    for i in range(2):
                        self.cards_without_fold_cards.pop(self.players[client] * 2 -
(len(self.fold_players) - 1) * 2)
                    message = data.copy()
                    message.append(self.players[client])
                    #don't send the spectator the data again
                    self.dont_send_to_spec = True
                    list(self.players)[-1].send(json.dumps(message).encode())
                    time.sleep(0.5)
                    stats =
holdem_calc.calculate(self.generate_hand_holdem_calc(self.cards[
                        (self.players_counter - 1) * 2:]), True, 1, None,
self.generate_hand_holdem_calc(
                        self.cards_without_fold_cards[:(self.players_counter - 1 -
len(self.fold_players)) * 2]), False)
                    print(stats)
                    list(self.players)[-1].send(json.dumps(stats).encode())
                    #everyone except one player is fold - game over
                    if len(self.fold_players) == len(self.players) - 2:
                        self.over = True
                        self.broadcast_without_spectator('game over'.encode())
                #action is raise, the last player of the gambling round need to change
                elif data[0] == 'raise':
                    self.last_player = list(self.players)[:-
1][list(self.players).index(client) - 1]
                #the gambling round is over
                if self.last_player == client:
                    print(self.cards_counter)
                    #if there are 4 or less cards, there is need to send another card
                    if self.cards_counter <= 4:
                        self.flop_turn_river(1)
                        stats =
holdem_calc.calculate(self.generate_hand_holdem_calc(self.cards[
                        (self.players_counter - 1) * 2:]), True, 1, None,
self.generate_hand_holdem_calc(
                        self.cards_without_fold_cards[:(self.players_counter - 1 -
len(self.fold_players)) * 2]), False)
                        print(stats)
```

```
                        self.message_queues[list(self.players)[-
1]].put(json.dumps(stats).encode())

                    #game over

                    else:

                        self.over = True

                        self.broadcast_without_spectator('game over'.encode())

                #append the one who send the data to the received list and send it to
all the clients except him

                data.append(self.players[client])

                print('received ' , data , ' from ' , client.getpeername())

                for player in self.players:

                    if player is not client:

                        #if the spectator has already gotten the data

                        if player is list(self.players)[-1] and
self.dont_send_to_spec:

                            self.dont_send_to_spec = False

                        else:

                            self.message_queues[player].put(json.dumps(data).encode())

                            if player not in self.outputs:

                                self.outputs.append(player)

                #if the game is'nt over, send play to the next player

                if not self.over:

                    next_player = list(self.players)[(list(self.players).index(client)
+ 1) % (self.max_num_clients - 1)]

                    count = 2

                    while self.players[next_player] in self.fold_players:

                        next_player =
list(self.players)[(list(self.players).index(client) + count) % (self.max_num_clients
- 1)]

                        count += 1

                    self.message_queues[next_player].put('play'.encode())

            #the game is over, receive everyone hands to evaluate them and braodcast
the winner

            else:

                self.hands.append(data)

                if len(self.hands) == (self.max_num_clients - 1) -
len(self.fold_players):

                    self.winner = self.evaluate_all_hands(self.hands)

                    self.broadcast(str(self.winner).encode())
```

```
                    threading.Thread(target = self.reset_properties, args =
(4,)).start()
        #the client is close
        else:
            print('closing ' , self.client_address , ' after reading no data')
            if client in self.outputs:
                self.outputs.remove(client)
            self.inputs.remove(client)
            #client.close()
            #del self.message_queues[client]
            self.players_counter -= 1
            self.max_num_clients -= 1

    #send data from the queue to a given client
    def send_data(self, client):
        try:
            next_msg = self.message_queues[client].get_nowait()
        except queue.Empty:
            print('output queue for ' , client.getpeername() , ' is empty')
            self.outputs.remove(client)
        else:
            print('sending ' , next_msg.decode() , ' to ' , client.getpeername())
            client.send(next_msg)
            time.sleep(0.5)

    #there is an exceptional condition. close the client
    def exceptional(self, client):
        print('handling exceptional condition for ' , client.getpeername())
        self.inputs.remove(client)
        if client in self.outputs:
            self.outputs.remove(client)
        client.close()
        del self.message_queues[client]
        self.players_counter -= 1
        self.max_num_clients -= 1

    #the infinity loop that calls the receive, send and exceptional functions
```

```
    def main_loop(self):
        while self.inputs:
            print('waiting for the next event')
            readable, writable, exceptional = select.select(self.inputs, self.outputs,
self.inputs)
            for client in readable:
                #new client
                if client is self.server:
                    self.accept_client(client)
                #client send to the server data
                else:
                    self.receive_data(client)
            for client in writable:
                self.send_data(client)
            for client in exceptional:
                self.exceptional(client)

    #evaluate all hands and return the winner
    def evaluate_all_hands(self, hands):
        winner = 0
        maximum = 0
        for hand in hands:
            fixed_hand = self.generate_hand_eval7(hand)
            print(hand)
            print(fixed_hand)
            x = eval7.evaluate(fixed_hand)
            if x > maximum:
                maximum = x
                winner = hands.index(hand)
        #the winner can't be a fold player
        while winner in self.fold_players:
            winner += 1
        return winner

    #send all the clients a given message
    def broadcast(self, data):
        for player in self.players:
```

```
            self.message_queues[player].put(data)
            if player not in self.outputs:
                self.outputs.append(player)

    #send all the players a given message (without spectator)
    def broadcast_without_spectator(self, data):
        for player in list(self.players)[:-1]:
            self.message_queues[player].put(data)
            if player not in self.outputs:
                self.outputs.append(player)

    #generate a standard 52 cards deck
    def generate_deck(self):
        self.deck = [(shape + '_' + value) for value in self.values for shape in
self.shapes]

    #return number of random cards by a given number
    def get_random_cards(self, cards_number):
        new_cards = []
        for i in range(cards_number):
            new_cards.append(secrets.choice(self.deck))
            self.deck.remove(new_cards[-1])
        return new_cards

    #convert my representation of cards to eval7 and holdem_calc representation
    def convert_mycard_to_eval7(self, card):
        if card[card.index('_') + 1:] == '10':
            return 'T' + card[0].lower()
        if card[card.index('_') + 1:] == '11':
            return 'J' + card[0].lower()
        if card[card.index('_') + 1:] == '12':
            return 'Q' + card[0].lower()
        if card[card.index('_') + 1:] == '13':
            return 'K' + card[0].lower()
        if card[card.index('_') + 1:] == '14':
            return 'A' + card[0].lower()
        return card[card.index('_') + 1:] + card[0].lower()
```

```
    #generate hand from my representation to eval7 representation
    def generate_hand_eval7(self, cards):
        hand = []
        for card in cards:
            hand.append(eval7.Card(self.convert_mycard_to_eval7(card)))
        return hand

    #generate hand from my representation to holdem_calc representation
    def generate_hand_holdem_calc(self, cards):
        hand = []
        for card in cards:
            hand.append(self.convert_mycard_to_eval7(card))
        return hand

#create an object from the class.
s = Server(2222)
#start the loop
s.main_loop()
```