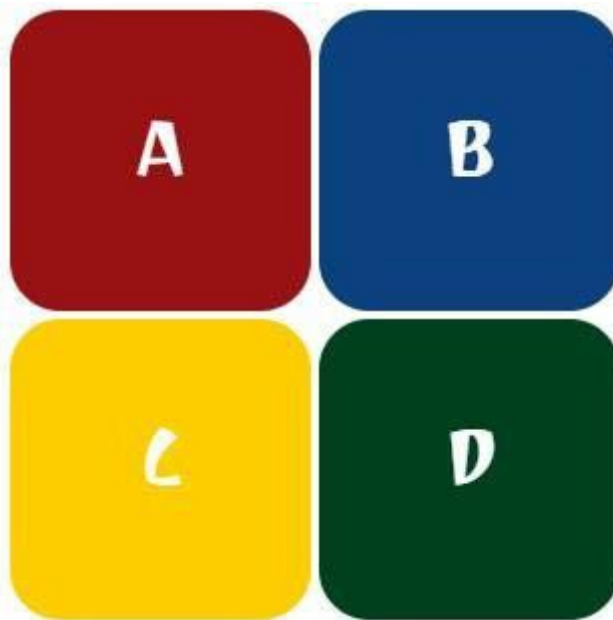


פרוייקט הגנת סייבר
התמחות 14.50
במסגרת תוכנית גבהים

שם התלמיד:	גיא מרקמן
מספר ת.ז.:	209310705
בית הספר:	ליאו בק
מקצוע:	הגנת סייבר
שמות המנחים:	שרית לולב ו- אלון בר-לב
שם הפרוייקט:	PyHoot



תוכן העניינים

1	תוכן העניינים
3	מבוא
4	ארכיטקטורה
6	רקע תיאורטי
6	טכנולוגיה
9	מימוש
9	שרת אסינכרוני
9	הצורך
9	polling
9	Select API
10	Poll API
11	בקשת HTTP
13	Block Diagram
14	Database Diagram
15	הצטרפות למשחק
16	State Machine - Master
18	State Machine - Player
20	Get Quetsion: השירות - בקשת שאלה
21	שליחת תשובה
21	בניית שאלון
22	PyHoot-Robot
23	רשימת שירותים
24	register_quiz - Service
24	homepage - Service
24	answer- Service
25	getnames - XMLService
25	disconnect_user - Service
26	check_name - XMLService
26	join - Service

26	check_test_exist - XML Service
27	new - Service
27	get_join_number - XMLService
27	get_information - XMLService
28	set_timer_change - Service
28	check_timer_change - XML Service
28	order_move_all_player - Service
28	order_move_all_not_answered - Service
29	check_move_next_page - XMLService
29	moved_to_next_page - Service
29	move_to_next_question - XMLService
29	get_xml_leadeboard - XMLService
30	get_question - XMLService
30	check_move_question - XMLService
30	get_score - XMLService
31	start_question - Service
31	get_answers - XMLService
31	get_title - XMLService

32 בעיות ידועות

33 התקנה ותפעול

33 PyHoot

36 PyHoot-Robot

37 תוכניות עתיד

37 תוכניות ל: PyHoot:

37 תוכניות ל: PyHoot-Robot:

38 פרק אישי

39 קוד פרויקט

40 נספחים

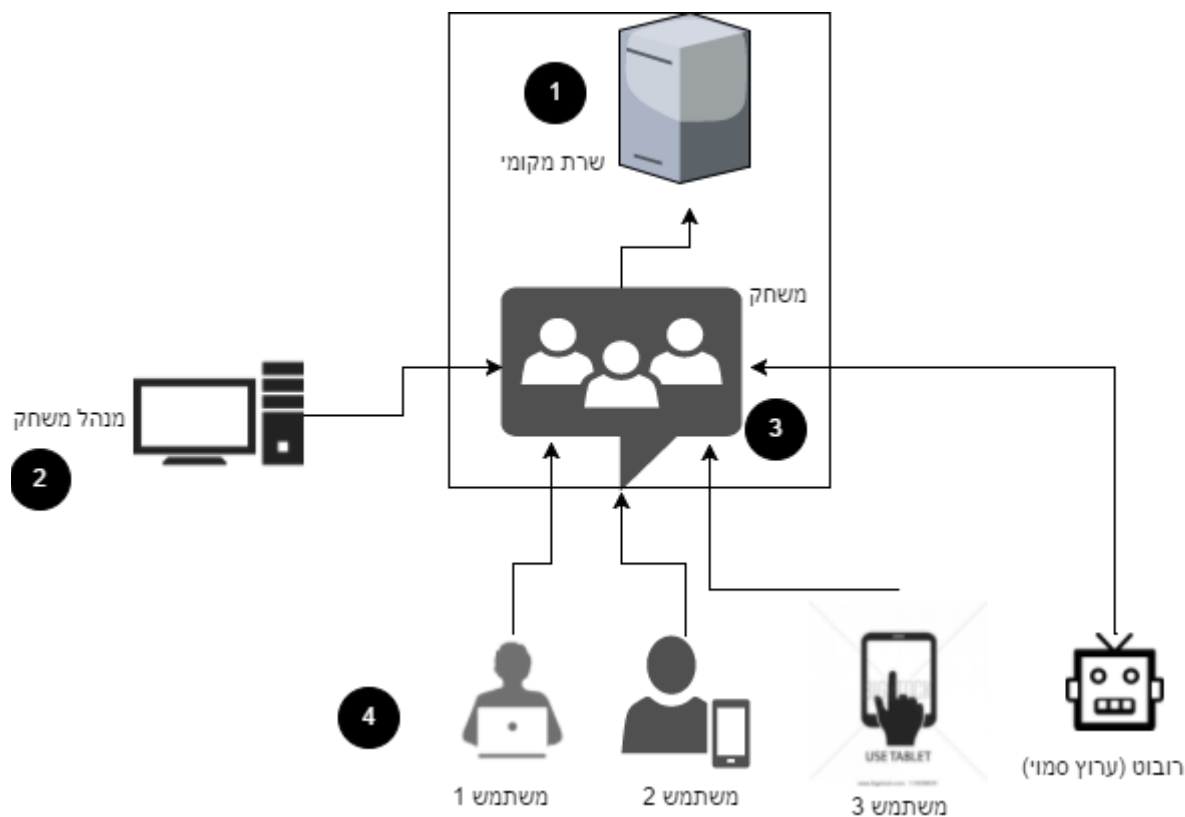
מבוא

מסמך זה הוא מסמך האיפיון של פרויקט הסיום שלי, PyHoot. PyHoot הוא משחק בסגנון משחק האינטרנט Kahoot, הבנוי על שאלות בסגנון "שאלות רבות ברירה" ("שאלות אמריקאיות"). PyHoot מאפשר הצגת שאלונים לצורכי למידה ולצורכי הנאה, המאפשר לבדוק את רמת הידע של המשתתפים בנושאי השאלון. ניתן לשלוט במשחק ולהשתתף בו מכל מכשיר שהוא. בו זמנית, התכנה יכולה לרוץ גם כשרת HTTP אסינכרוני ללא קשר למשחק. מטרת המוצר היא לאפשר פלטפורמה פשוטה שתוכל לשמש את כלל הציבור, ובעיקר אנשי-חינוך ותלמידים, ככלי להנאה, לאימון, לחזרה על החומר ולהכנה לבחינות. במוצר זה יוכל להשתמש כל אדם, ללא צורך בהרשמה מראש.

ייחודו של כלי זה, לעומת משחק האינטרנט Kahoot הוא בעובדה שלעומת ה-Kahoot העובד על רשת האינטרנט, ה-Pyhoot עובד על רשת מקומית. במסגרת זו מנהל המשחק או המשתתפים יכולים להשתתף במשחק דרך כל כלי התקשורת המחוברים לרשת המקומית, כגון: מחשב, טלפון נייד, טאבלט וכדומה. העובדה כי המשחק פועל על הרשת המקומית מאפשרת זמן תגובה מהיר יותר, הגנה על הפרטיות של המשתמשים וכן הגנה של הקבצים על המערכת. יתרון נוסף של השימוש ב-Pyhoot הוא שליטה רבה יותר של מנהל המשחק בהיבטים שונים של המשחק, ובעיקר באפשרויות העיצוב.

בנוסף לתכנת השרת, יש ל-PyHoot גם "רובוט", שהוא ערוץ סמוי, המדמה שחקן פיקטיבי. ה"רובוט" מסוגל לפענח מה היא התשובה הנכונה במרבית המקרים. כאשר הוא לא מצליח לפענח את התשובה הנכונה, הוא בוחר בתשובה רנדומלית. מטרת ה"רובוט" היא לדמות שחקן אנושי, בכדי ליצור הרגשה של תחרות רבה יותר בקרב השחקנים.

ארכיטקטורה



ישויות:

מספר / שם	תפקיד
1. שרת מקומי	השרת אשר מכיל את המידע ואליו מתחברים המשתמשים
2. מנהל המשחק	משתמש המתחבר לשרת ומפעיל את המשחק על השרת.
3. המשחק	משחק אשר מנהל השרת הפעיל, אליו מתחברים (באמצעות השרת) השחקנים ובו הם משחקים. על השרת יכולים לפעול משחקים שונים בו-זמנית.
4. שחקנים	שחקנים אשר מתחברים לשרת ומשחקים במשחק.
הערה	גם השחקנים וגם מנהל המשחק יכולים להתחבר לשרת ולהשתמש בשרת HTTP אסינכרוני רגיל.

תהליכים

תהליך	תיאור
מנהל משחק - שרת	מנהל המשחק מתחבר לשרת ויוצר עליו משחק חדש
משחק-שרת	המשחק מבקש מהשרת את קובץ השאלון ומאתחל את המשחק
משתמש-שרת	המשתמש מתחבר לשרת ומבקש ממנו את הקבצים הדרושים (דף הבית)
משתמש-משחק	המשתמש מתחבר למשחק
משחק-מנהל משחק	המשחק מעדכן את המנהל שהצטרף שחקן חדש
מנהל משחק - משחק	מנהל המשחק מתחיל את המשחק

רקע תיאורטי

טכנולוגיה

[Python](#) - פייתון היא שפת סקריפטים מהנפוצות ביותר אשר תוכננה תוך שימת דגש על קריאות הקוד, בכך שהיא מבטלת סיבוכים מיותרים שקיימים בשפות אחרות וכוללת מבני נתונים המיועדים לכתיבת קוד קריא. בפייתון יש תמיכה לתכנות מונחה עצמים וכן לתכנות פונקציונלי. Python מכילה בתוכה אוסף מכובד של ספריות, כמו לדוגמא ספריה המאפשרת טיפול אינטנסיבי ב-HTML, XML, וגזרותיהם, הקמת חיבורי TCP ו-UDP וכדומה.

[HTML](#) - שפת תגיות לתצוגה ועיצוב דפי אינטרנט ותוכן לתצוגה בדפדפן. זוהי השפה המרכזית בעולם האינטרנט, המהווה שלד למרבית העמודים באינטרנט. HTML מאפשרת עיצוב תוכן בצורה מהירה וקלה הן ללימוד והן לכתיבה. HTML תוכננה לעבוד על כל מחשב, מכל סוג והיא "סלחנית" מאוד לגבי פרטים קטנים, היא מותרת לשימוש על ידי כל מפתח, ללא צורך ברכישת זכויות יוצרים, והיא ניתנת לקריאה בכל סוגי המערכות.

[JavaScript](#) - היא שפת תכנות מונחית עצמים המתואמת לשימוש באתרי אינטרנט ורצה על ידי דפדפן בצד הלקוח. השפה מרחיבה את יכולת שפת HTML ומאפשרת בכך ליצור יישומי אינטרנט מתוחכמים. רוב אתרי האינטרנט המודרניים משלבים שפה זו על מנת להציג דפים דינמיים שמשולבת בהם תוכנה.

[CSS](#) - בשמה המלא Cascading Style Sheets היא פורמט לעיצוב דפי אינטרנט. הפורמט קובע את עיצובם של תגים ב-HTML ובכל שפה דומה ל-XML לבניית אתרי אינטרנט. CSS נוצרה במטרה להפריד בין תוכן ומבנה דפי האינטרנט לבין עיצובם: עד ליצירת ה-CSS בשנת 1995, תוכן וסגנון האתרים נכתבו באותו דף HTML, אשר הפך את הקוד למסובך ובלתי קריא. בנוסף לכך שינויים בעיצוב שלם דרשו מעבר על דף אחד אחרי השני. באמצעות CSS ניתן למקם הגדרות עיצוב בקובץ יחיד, ששינוי בו ישתקף בבת אחת בכל הדפים העושים בו שימוש.

[XML](#) - ראשי התיבות של eXtensible Markup Language, הוא תקן לייצוג נתונים במחשבים. שימוש ב-XML מקל על החלפת נתונים בין מערכות שונות שפועלות על גבי תשתיות שונות. תקן ה-XML לא מגדיר איזה מידע יוצג אלא כיצד יוצג מידע באופן כללי.

[Asynchronous IO](#) - במדעי המחשב Asynchronous IO הוא צורה של עיבוד קלט/פלט אשר מאפשר לתהליכים אחרים, אשר אינם תלויים בהשלמת הקלט/פלט להמשיך לפני שהשידור הסתיים. תהליכים של קלט ופלט יכולים להיות איטיים מאוד ביחס לפעולות במחשב, וזאת Asynchronous IO בא לפתור.

[TCP](#) - ראשי תיבות ל-Transmission Control Protocol הוא פרוטוקול בתקשורת נתונים הפועל בשכבת התעבורה ומבטיח העברה אמינה של נתונים בין שתי תחנות ברשת מחשבים. TCP מעביר

נתונים שהועברו באמצעות IP, מוודא את נכונותם, ומאשר את קבלת הנתונים במלואם או מבקש שליחה מחדש של נתונים שלא הגיעו בצורה תקינה.

[HTTP](#) - ראשי תיבות ל Hypertext Transfer Protocol הוא פרוטוקול תקשורת שנועד להעברת דפי HTML ואובייקטים שהם מכילים (תמונות, קבצי קול, קבצים וכו') ברשת האינטרנט. הפרוטוקול פועל בשכבת היישום של מודל OSI. שרתי HTTP הם שרתי התוכן המרכזיים בשרת האינטרנט ודפדפנים הם תוכנות הלקוח הנפוצות ביותר לפרוטוקול HTTP. על מנת ליצור תקשורת בין הלקוח לשרת שמובססת על היסטוריית הבקשות-תשובות בין השרת ללקוח נעשה שימוש ב Cookies (עוגיות).

HTTP מחולק לשני חלקים: הבקשה (Request) והתשובה (Response). לרוב הבקשה היא מהמשתמש והתשובה היא מהשרת, אך לא חובה.

שני החלקים זהים בערך ומורכבים מ Request/Response וגוף הבקשה/תשובה. החלק הראשון מחולק גם הוא לשני חלקים, שורת הסטטוס וגוף הבקשה/תשובה, אך שני החלקים דומים בכך שכל שורה בהם נגמרת בירידת שורה (\n) וירידת שורה נוספת בסוף הבקשה/תשובה.

שורת הסטטוס שונה מהבקשה ומהתשובה, אך מטרתה זהה, להגיד מה אנחנו רוצים מהשרת ומה התשובה שלו לבקשה זו. בנוסף לכך, שורת הסטטוס מהווה אמצעי אימות לבקשה.

בבקשה - שורת הסטטוס בנויה כך
 HTTP-Version Method Request-Uri
 Method - שיטות בקשה אשר אנו רוצים להשתמש בה, מהאנחנו רוצים לעשות בבקשה זו. נכון לגירסה 1.1 ישנן 8 שיטות בקשה שונות: GET, HEAD, POST, PUT, DELETE, OPTIONS, CONNECT ו-TRACE. בפרוייקט שלי יש תמיכה רק בGET, קבלת הקובץ בRequest-URI.
 REQUEST-Uri - הקובץ שאנחנו רוצים להפעיל עליו את המטודה.
 HTTP-Version - גרסת הHTTP בה אנו משתמשים. בפרוייקט יש תמיכה רק בHTTP/1.1.

בתשובה - שורת הסטטוס בנויה כך
 HTTP-Version Status-Code Status-Text
 HTTP-Version - גרסת הHTTP בה אנו משתמשים. בפרוייקט יש תמיכה רק בHTTP/1.1
 Status-Code Status-Text קוד של שלוש ואחריהם הטקסט המתאים לקוד זה. דבר זה מהווה אוטנטיקציה נוספת, במידה והם לא מתאימים אחד לשני, כנראה שהייתה בעיה בהעברה. הסטטוס והטקסט הם תשובה של השרת לבקשה. הידועים והנפוצים ביותר הם
 200 OK
 403 Forbidden
 404 Not Found
 500 Internal Server Error

לאחר מכן, יופיע לנו הקובץ שאנו שולחים או מקבלים מהצד השני.

HTTP עובד על פרוטוקול TCP. כל HTTP מועבר דרך הContent של TCP.

[עוגיה](#) - היא מחרוזת אותיות ומספרים, המשמשת לאימות, למעקב ולאגירת מידע על הגולש באתר אינטרנט, כגון שמירת העדפות המשתמש. העוגיה נוצרת על ידי השרת שמעביר אותה לדפדפן ששומר אותה בזיכרון המחשב. המחרוזת מוחזרת חזרה לשרת בכל פעם שהדפדפן יוצר קשר עם השרת וכך למעשה יכול השרת לזהות את המשתמש ולאחזר מידע שנשמר בין שיחות שונות.

[XMLHttpRequest](#) - הוא ממשק אשר מאפשר העברת מידע בין דפדפן לבין שרת אינטרנט. XMLHttpRequest מסופק לדפדפן על ידי השרת באמצעות JavaScript. בחלקו, המידע המוחזר מ-XMLHttpRequest משמש על מנת לערוך דף אינטרנט אשר נטען כבר. למרות שמו, XMLHttpRequest יכול להשתמש להעברת מידע אחר מקבצי XML, כגון קבצי HTML וקבצי טקסט.

[Socket](#) - בעברית שקע הוא נקודת עבור זרם נתונים בתקשורת בין תהליכים על גבי רשת מחשבים. כיום, בעידן האינטרנט רוב ה-Sockets הם שקעים המבוססים על פרוטוקול האינטרנט ולכן מרבית ה-Sockets הם Internet Sockets. לכל Socket יש כתובת שהיא שילוב של כתובת ה-IP ומספר הפורט. בהתבסס על כתובת זו Internet Sockets מאפשרות העברת נתונים אל התהליכים השונים.

[File Descriptor](#) - שם כללי למספר המתאר רכיב תוכנה (קובץ, Pipe, Socket וכו'). ניתן להשתמש בו ככתובת לרכיב התוכנה במקום לשמור את הרכיב כולו. כאשר ניגשים לרכיב התוכנה, המערכת ניגשת לטבלה בשם File Table אשר מתאימה לנו את הקובץ הנכון. כמובן שזה מייעל מאוד את כל העבודה עם רכיב תוכנה, כיוון שזה מקטין לנו את השימוש בזיכרון.

מימוש

שרת אסינכרוני

הצורך

כפי שנכתב בפרק הרקע התיאורטי על AsyncronicIO, פעולות הקלט והפלט מאוד איטיות. בנוסף לכך משתמשים שונים שולחים בקשות בזמנים שונים ויכולים לקבל מידע בזמנים שונים. מכאן אפשר להבין, שלהמתין או לבדוק כל פעם לכל משתמש האם הוא יכול לקבל או לשלוח מידע אל השרת מבזבז זמן רב וכן משאבים רבים מהמערכת. בנוסף המשתמשים משתמשים במערכות מחשוב פחות אמינות (כמו פלאפונים, מחשבים ניידים וטאבלטים) והחיבורים יכולים להתנתק.

על כל אלו, בא AsyncronicIO לענות. הוא מאפשר לנו, לבדוק בבת אחת את כל המשתמשים שיכולים לקבל מידע, את כל המשתמשים שיכולים לשלוח מידע ואת כל המשתמשים שיש לנו תקלה בחיבור איתם בבת אחת. בכך הוא חוסך לנו זמן יקר מאוד ומאפשר זמן תגובה הרבה יותר מהיר לכל משתמש. פעולה זאת נקראת polling.

polling

polling הינו תהליך שבו אנו שולחים למערכות את כל החיבורים הזמינים לנו והמערכת מחזירה לנו את האפשרויות השונות של חיבור הקשורות להם. בהתאם לפרוטוקול המערכת בונה מבנה נתונים המגדיר את אפשרויות ההתקשרות עם הלקוח, בודק כל אחת מהם ומאשר או שולל את דרך ההתקשרות הזו ומחזיר לנו את האפשרויות השונות. ישנם שני ממשקים של AsyncronicIO בשיטת polling אשר בהם פרוייקט זה משתמש: Poll ו Select.

Select API

בממשק Select המערכת מקבלת מהשרת שלוש רשימות של Sockets של החיבורים השונים שונים. הראשונה רשימת כל Sockets שלהם אנו רוצים לכתוב (לשלוח מידע), השנייה רשימת כל Socket שמהם אנו רוצים לקבל מידע, ורשימת כל Sockets שבהם אנו רוצים לבדוק שאין שגיאה בחיבור. המערכת מנפה כל אחת מהרשימות מאלו שהפעולה אינה אפשרית בה ומחזירה לנו תת רשימה של כל Sockets שהפעולה אפשרית בהם. ממשק זה הינו הממשק הוותיק יותר ונמצא בכל מערכות ההפעלה המודרניות.

Poll API

Poll הוא הגרסה המשודרגת והחדשה יותר של ממשק Select, בממשק זה אנו שולחים רשימה של כל File Descriptors שאנו מעוניינים לבדוק עליהם פעולות (למשל של כתיבה, קריאה, בדיקת שגיאה, בדיקת התנתקות וכו') והמערכת מחזירה עבור כל File Descriptor את כל הפעולות שניתן לבצע עבורו.

מכיוון שאנו משתמשים בFile Descriptors ולא Socket, ניתן לבצע זאת גם על קבצים ולא רק חיבורים, אך אפשרות זו אינה משומשת בפרויקט.

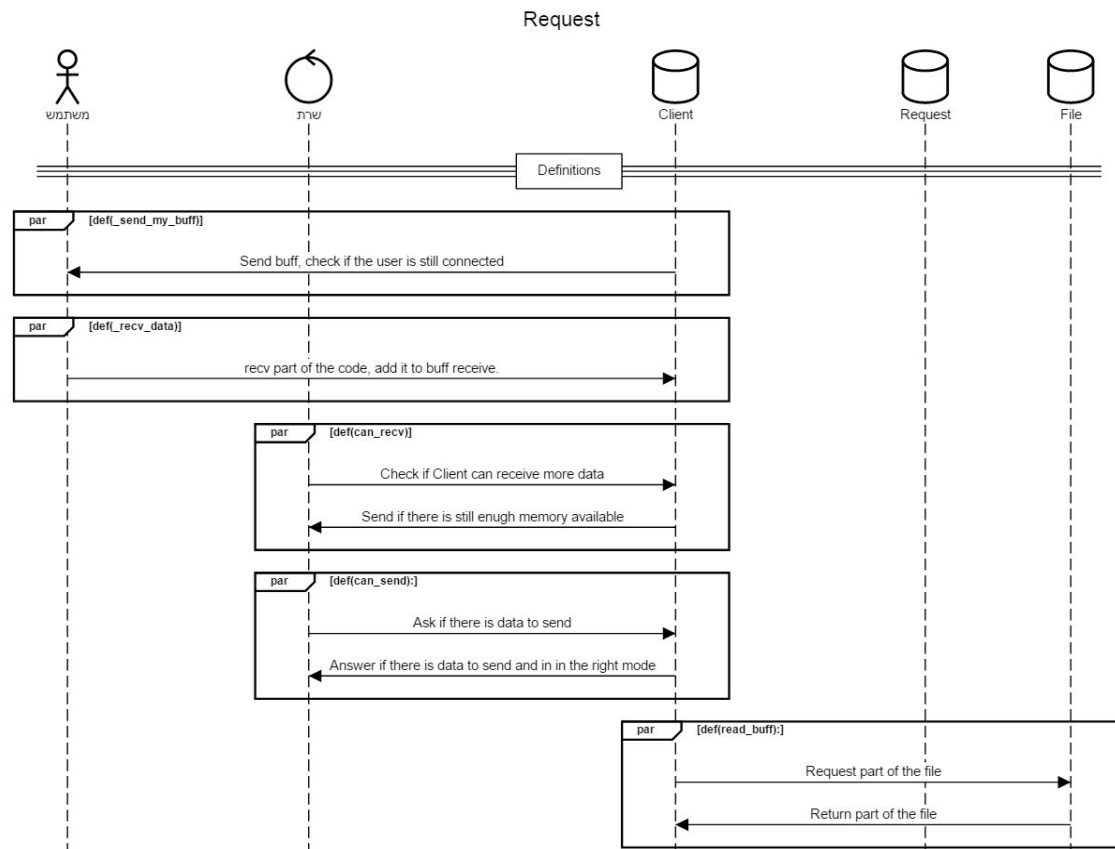
היתרון של Poll הינו בחיסכון עצום בזיכרון ובזמן תקשורת.

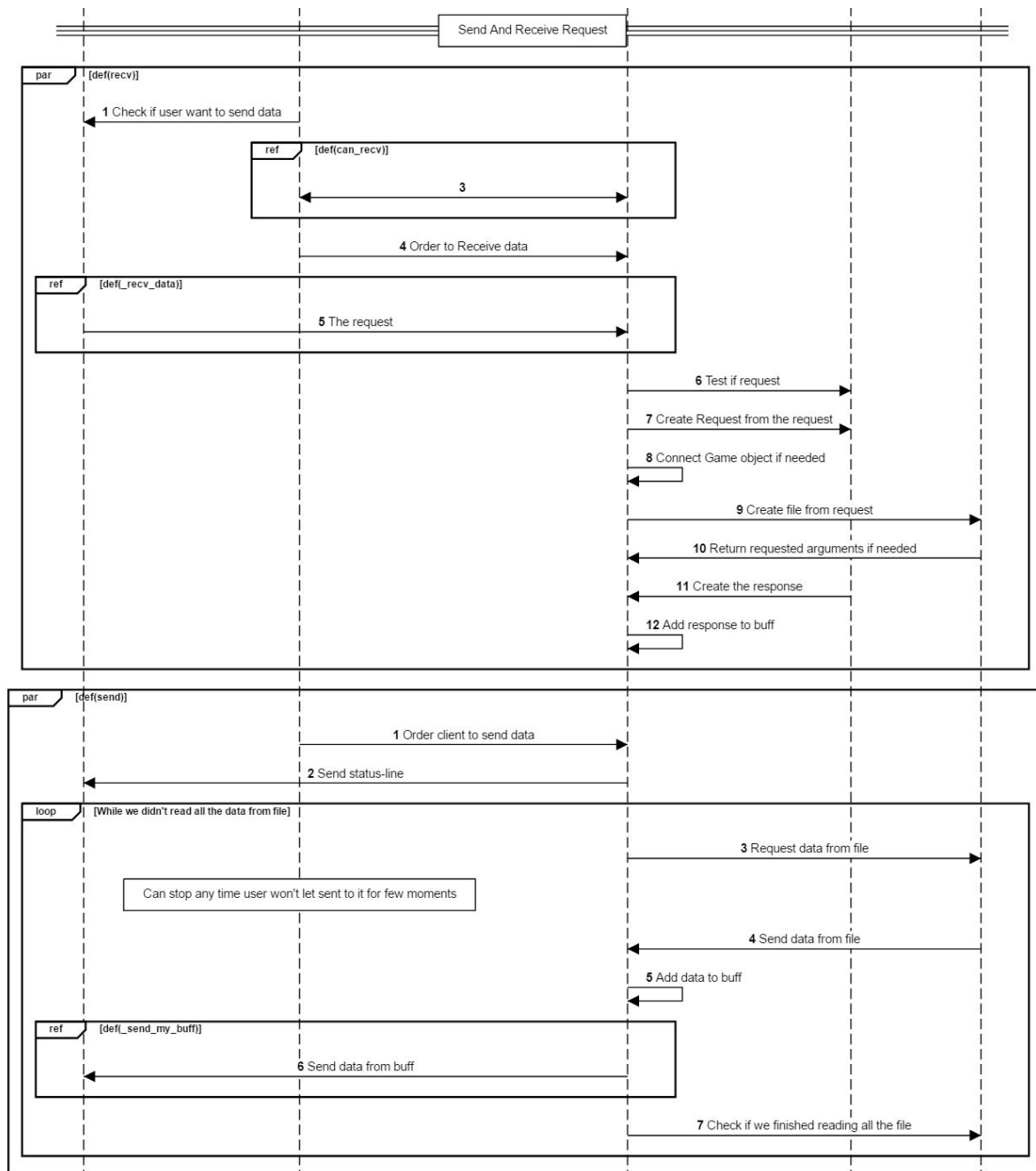
החיסרון העיקרי שלו זה שממשק זה זמן רק על מערכות BSD Unix ששתי הגרסאות העיקריות שלהן הם Linux וMacOS, אך לא על מערכות הפעלה מבית Windows.

בקשת HTTP

הקוד המלא לדיאגרמה נמצא בנספחים

[\[Link\]](#)

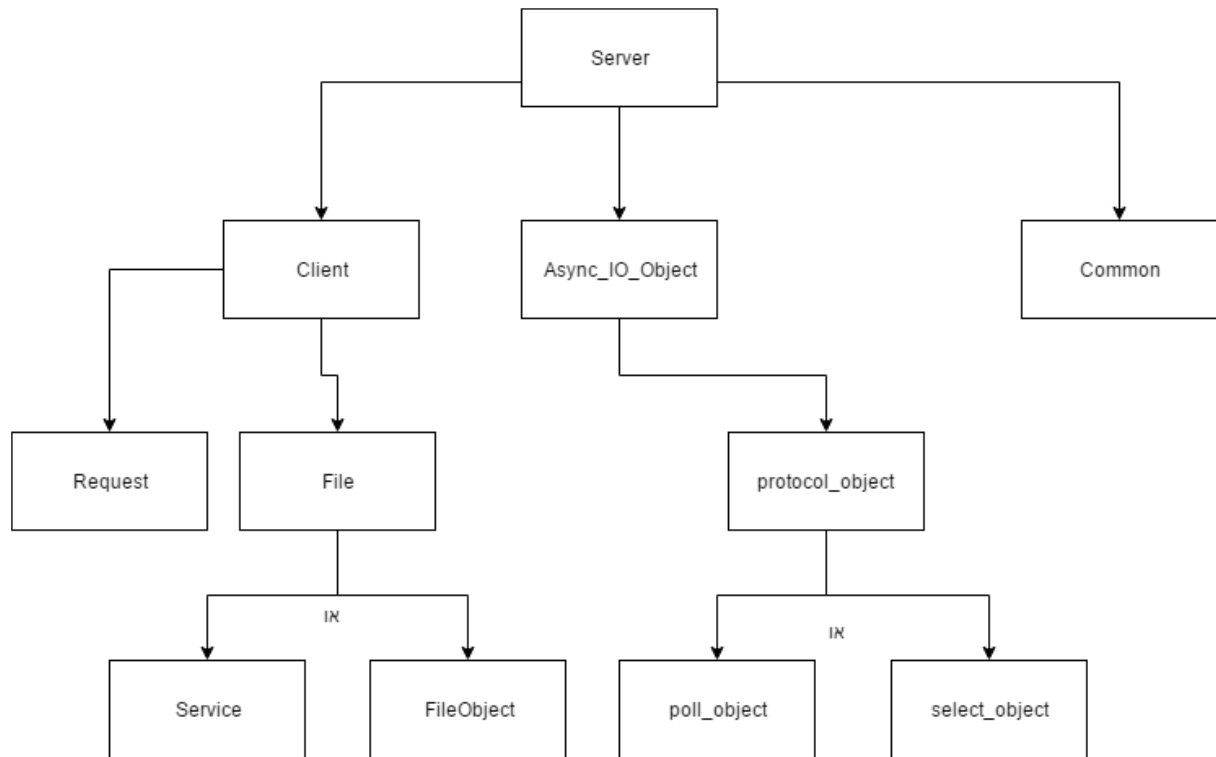




Block Diagram

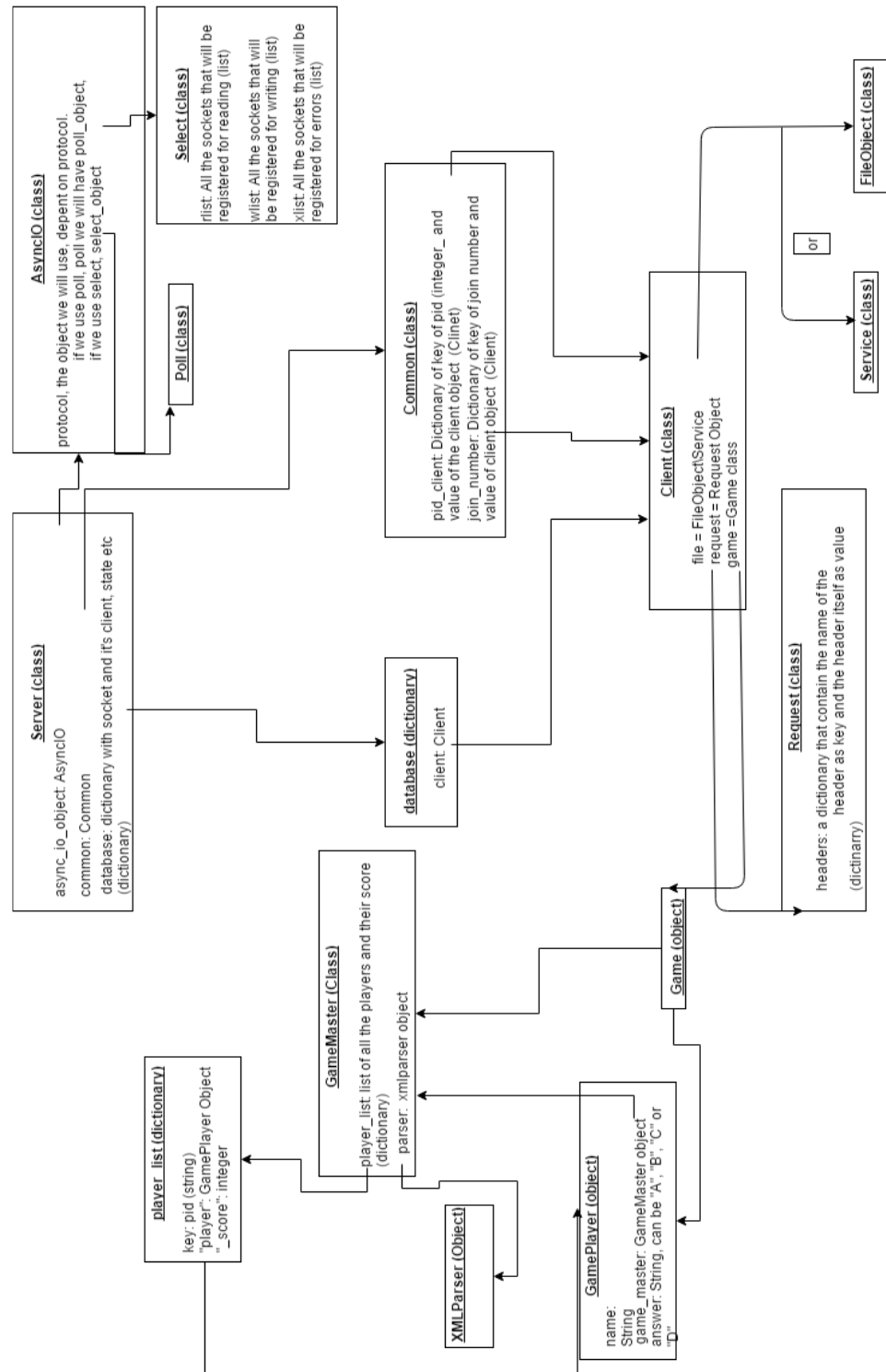
הקשר בין המחלקות והאובייקטים השונים

[\[link\]](#)



Database Diagram

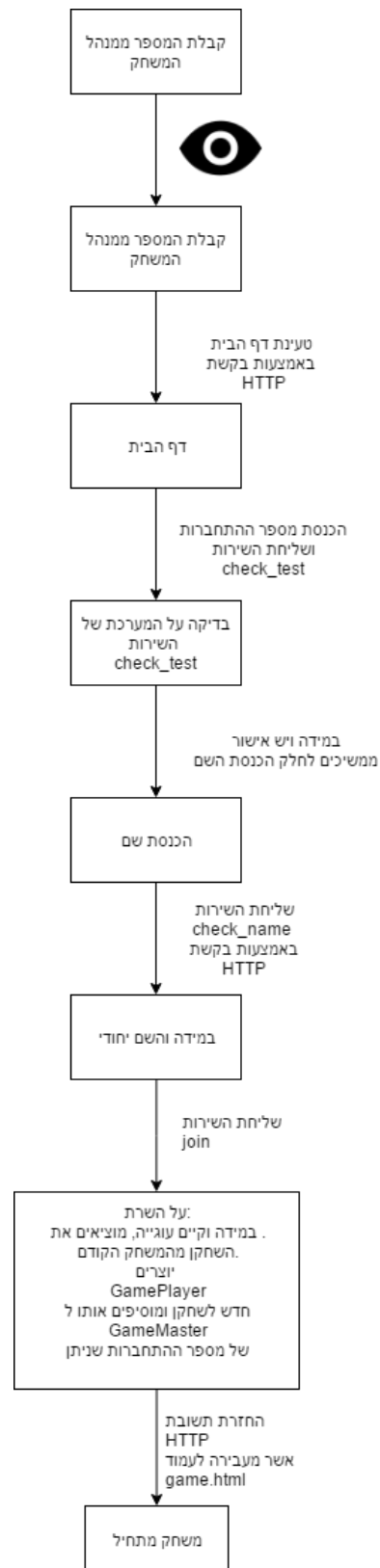
[\[Link\]](#)



הצטרפות למשחק

תהליך ההצטרפות למשחק

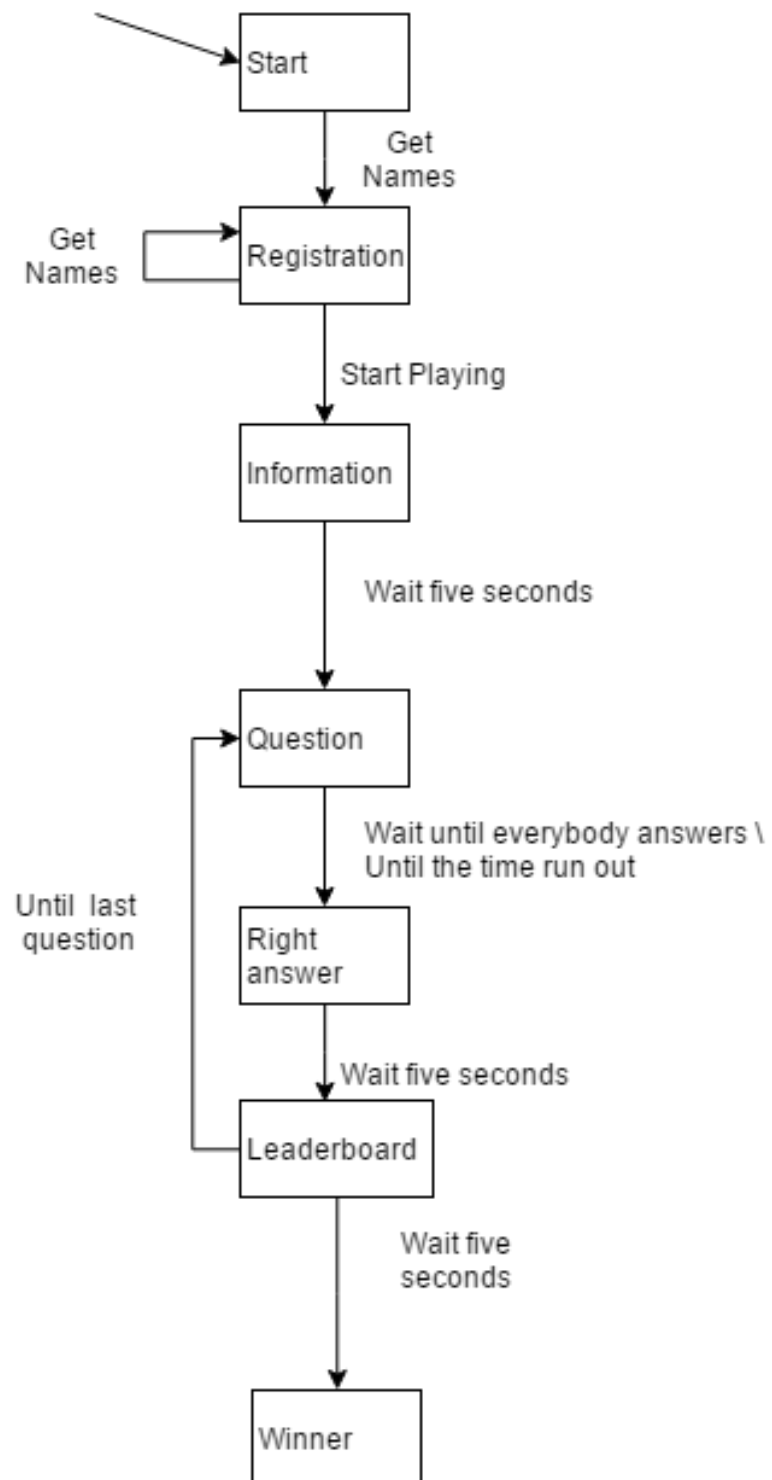
[Link]



State Machine - Master

[\[Link\]](#)

סדר המהלכים במשחק של מנהל המשחק



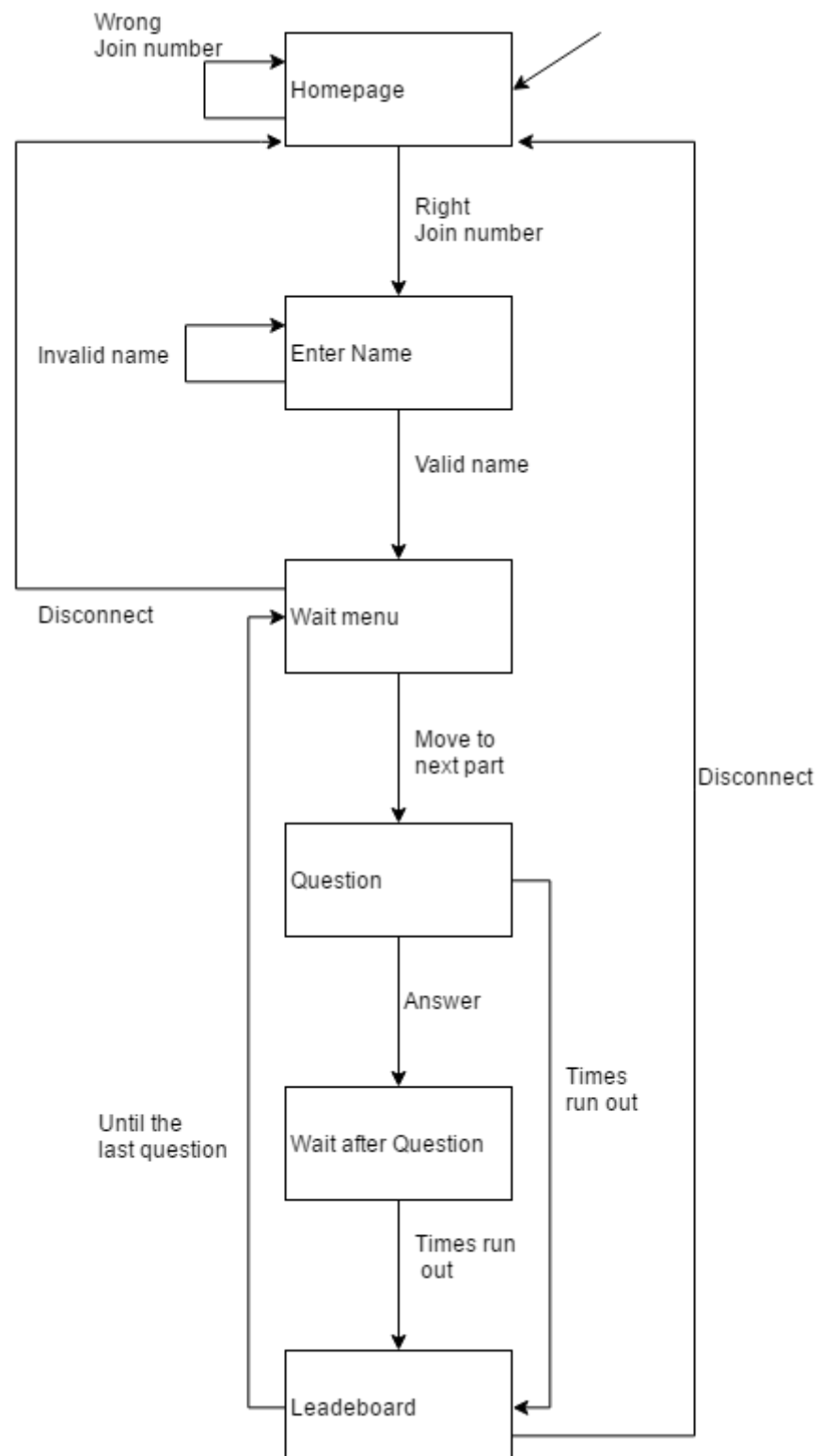
הסבר

שם	הסבר
Start	ישירות אחרי שהמנהל המשחק יוצר את המשחק, היווצרות של המשחק ואיתחולו
Get Names	מסמל את סיום האיתחול, קורה כל שנייה. קריאה לשירות אשר מחזיר את שמות כל השחקנים המחוברים למשחק הזה ומדפיס אותך על המסך
Registration	מסך ההרשמה, יופע על המסך מספר ההתחברות לשחקנים
Start Playing	מנהל המשחק לוחץ על הכפתור שמתחיל את המשחק
Question	על המסך מופיע השאלה והתשובות שלה.
Wait until everybody answer \ time runout	ממתינים עד שכל השחקנים השונים עונים על השאלה או שנגמר הזמן שהוקצב לשאלה (המוקדם מביניהם).
Leadeboard	על המסך מופיעים השחקנים הטובים ביותר ותוצאתיהם הנוכחיות
Until Last answers	השליבים Quetsion עד Leadeboard חוזרים על עצמם עד השאלה האחרונה.
Winner	על המסך מופיע השחקן המנצח והנקודות שצבר.

State Machine - Player

סדר המהלכים במשחק של שחקן במשחק

[\[Link\]](#)

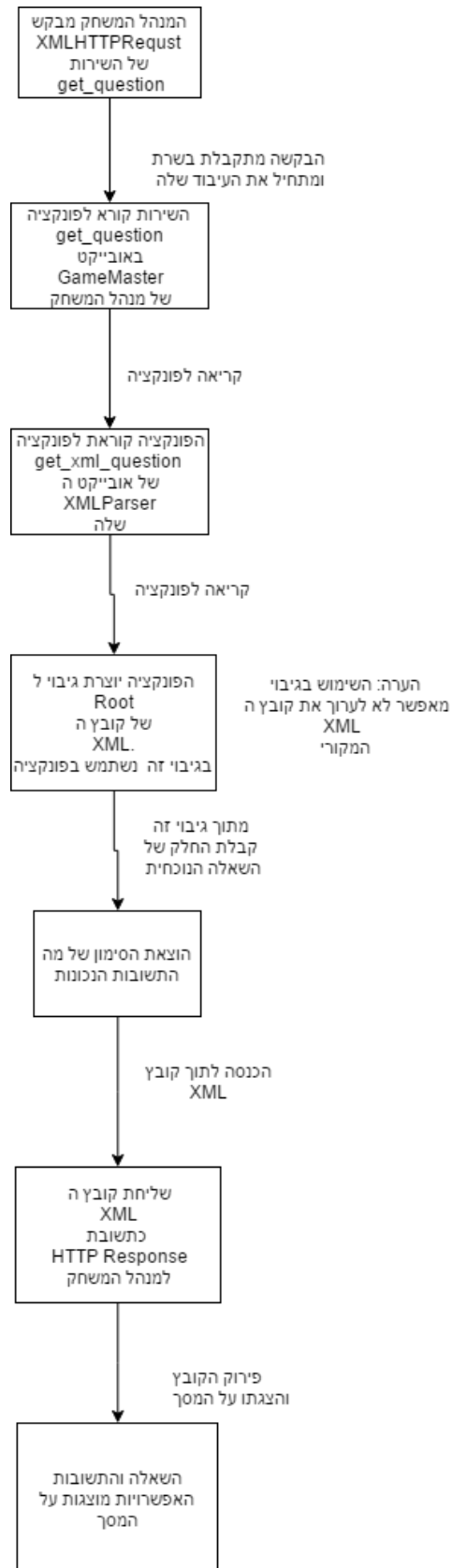


הסבר

שם	הסבר
Homepage	מסך הבית, לשם המשתמש מגיע לראשונה. מכיל שדה אשר מאפשר להכניס Join Number
Wrong Join Number	במידה והמשתמש הכניס Join Number ששום משחק הרץ נכון לרגע זה במערכת.
Right Join number	במידה והמשתמש הכניס Join number של משחק הרץ נכון לרגע זה במערכת.
Enter Name	המשתמש מועבר למסך שבוא הוא יכול להכניס את השם שלו / של השחקן.
Valid name	המשתמש הכניס שם שאורכו ארוך משלושה שווים או שווה לשלושה תווים ואף משתמש אחר במערכת לא משתמש בשם זה.
Invalid name	המשתמש הכניס שם שאורכו קצר משלושה תווים או שם שמשתמש אחר במערכת משתמש בו כבר.
Wait Menu	מסך שבוא השתמש מחכה עד שהשאלה הבאה תתחיל. יש במסך זה אפשרות להתנתק מהמערכת.
Disconnect	מנתק את המשתמש מהמערכת, מחזיר אותו לHomepage
Move to next part	מנהל המשחק מעביר את המשחק לחלק הבא שלו.
Question	המשתמש מועבר למסך שבו הוא יכול לבחור את התשובה הנכונה וכן רואה את השאלה עצמה
Answer	השחקן בוחר את אחת השאלות
Times run out	נגמר הזמן שהוקצב לשאלה או שכל השחקנים ענו
Leadeboard	המשתמש מועבר למסך שבוא יש מוצג לו מיקומו במשחק וכן את הניקוד שצבר עד. בנוסף לכך יש לו אפשרות להתנתק במסך זה
Until last question	התהליך מWait Menu ועד Leadeboard ממשיך עד שעוברים על כל השאלות

בקשת שאלה - השירות Get Quetsion:

[[Link](#)]



שליחת תשובה

- המשתמש לוחץ על אחד מהכפתורי תשובה
- הלחיצה על הכפתור שולחת בקשת של XMLHttpRequest של השירות answer עם פרמטר בשם answer שערכו הוא "A", "B", "C" או "D".
- השירות מתקבל על השרת ומגדיר את answer של של יישום אובייקט GamePlayer שלו כפרמטר.
- בהגדרת הפרמטר נשמר גם אצל השחקן הזמן שבו התקבלה התשובה.
- מוחזר לשרת טקסט ריק שמסמל כי הבקשה התקיימה.

בניית שאלון

לצורך בניית השאלון כתבתי תוכנה פשוטה (Files\editor.py או Files\editor.pyc) אשר לוקחת מידע מהמשתמש והופכת אותו לקובץ XML פשוט. ניתן גם ליצור קובץ באופן ידני, אך יש לשים לב שאם הוא לא יעמוד בפורמט המדויק, לא יהיה ניתן לשחק בו.

הפורמט בנוי בצורה הבאה (ניתן לראות פירוט זה גם בקובץ Quiz\template.xml):

XML פורמט ה <?xml version="1.0" encoding="UTF-8"?>

<Root> הבסיס של הקובץ

<Quiz name="שם השאלון" number_of_questions="מספר השאלות בשאלון">

עבור כל שאלה

<Question duration="זמן ההצגה המקסימלי בשניות">

<Text>HTML השאלה עצמה בשפה חופשית או מבנה של</Text>

עבור כל שאלה חייבות להיות ארבע תשובות

<Answer correct="1"עבור נכון, בלי הערך או כל ערך אחר עבור טעות">

<Text>התשובה עצמה בשפה חופשית או מבנה של</Text>

HTML</Text>

</Answer>

</Question>

</Quiz>

</Root>

PyHoot-Robot

הרובוט מתנהג מתנהג כשחקן לכל דבר, מאותחל על ידי אחד מהשחקנים (אשר מכניס לו את המידע של Join Number ואת השם של השחקן), אחרי הכנסת המידע הרובוט פועל באופן עצמאי לחלוטין. הרובוט רק עובר בין השלבים השונים (בלי לקרוא לקבצים הנדרשים, דבר שאינו מפריע לכל שאר המשחק) עד שמגיעים לחלק של השאלה עצמה. בחלק זה הרובוט קורא לשירות `get_title` אשר ממנה הוא מקבל קובץ XML של השאלה עצמה, בלי התשובות. מתוך קובץ זה הוא מפרק את הנתונים ובודק האם קיימת תמונה לשאלון זה. במידה וקיימת תמונה הוא עושה HTTP Request לתמונה זו.

על תמונה זו הוא מבצע פענוח בניסיון לזהות מהיא התשובה הנכונה המוצפנת בתוך התמונה. והיה והצליח, הוא יחזיר את תשובה זו. במידה והרובוט לא מוצא תמונה בשאלה, או שלא מצליח להוציא מהתמונה את התשובה הנכונה, הוא ישלח תשובה רנדומלית.

רשימת שירותים

הגדרות:

בפרויקט ישנם שני סוגי שירותים שונים:

:Service

מחלקת בסיס אשר מהווה את הבסיס לכל השירותים, מהווה פולימורפיזם למחלקה `FileObject`. מכיל בתוכו את המשתנים הבאים:
`finished_reading`, בוליאני, בדיקה האם סיימנו לקרוא את כל המסמך שאנו צריכים לעבוד איתו. ערך התחלתי: `False`
`read_pointer`, מספר שלם, מהווה סמן שבאמצעותו ניתן לחזור למקום בתוך הקובץ שאנו מחזירים. ערך התחלתי: `0`
`_content_page`, הפניה לפונקציה אשר תהווה לנו את תוכן הפונקציה ערך התחלתי: `None`,
 לרוב יוגדר ב `read_buff` או מספר שורות לאחר מכן בפונקציית האיתחול.
 בנוסף לכך, לכך שירות יש קבוע בשם `NAME` אשר מהווה את ה `URL` לקבלת השירות ממחלקה זו. לרוב יהיה "שם_השירות/"

בנוסף לכך, ל `Service` יש כמה פונקציות מובנות:
`content`: מחזיר `string` ריק, התוכן של הדף שמוחזר. דף ריק מוחזר עבור שירותים ששום דבר לא אמור לחזור עבורם, אבל בגלל המבנה של `HTTP` אנו חייבים לאשר את הקבלה ולכן אנו מחזירים `string` ריק.
`close`: ב `FileObject` אמור לסגור את המסמך שאנו עובדים עליו, כאן אנו פשוט ממשיכים הלאה (`pass`).
`headers`: מקבל מילון של כל ה `Headers` שצריך להוסיף לתשובה ומחזיר תשובת `HTTP` גרסה 1.1 עם קוד `200 OK`, למסמך `HTML` עם ה `Headers` שקיבלנו.
`read_buff`: מקבל את המקסימלי שאפשר לקרוא, מחזיר מחרוזת בגודל הזה או קצרה יותר של תוכן התשובה של השירות. בנוסף לכך מגדיר את `_content_page` בתור הפונקציה `content` אם עדיין לא הוגדרה פונקציה.
`get_status`: מחזיר את `finished_reading`

:XMLService

מחלקה היורשת מ `Service` והשינוי העיקרי שלה הוא ש `headers` מחזיר תשובה למסמך `XML` במקום `HTML`.

:boolean xml

`xml` המייצג ערך בוליאני.

מבנה:

<Root answer:"True or False"/>

רשימת שירותים:

הערה: במידה ורשום בקלט "אין" הכוונה היא שהקלט הוא רק הבקשה שהתקבלה. בקלט הכוונה היא בנוסף לתשובת הHTTP המתאימה.

register_quiz - Service

NAME: /register_quiz

שירות שיפעל רק עם אובייקטים Game מסוג GameMaster.

קלט:

מהמשתמש: שם שאלון וכן pid נוכחי, ערך עוגיית ההזדהות מול המערכת.
מהשרת: המחלקה common וכן base_directory של השרת, המיקום של הקבצים בעלי הגישה, התיקיה Files.

פלט:

רושם את מנהל המשחק לשרת, יוצר משחק על שם זה ובמידה ומנהל המשחק רשום למשחק אחר (הן בתור מנהל המשחק והן בתור שחקן) השרת מנתק את המנהל ממשחק זה.
מחזיר הפנייה (קוד 302, Found) לדף quiz.html וכן מגדיר עוגייה בשם pid אשר זהה הpid של אובייקט המשחק שנוצר.

homepage - Service

/ = NAME

דף הבית

קלט:

אין

פלט:

מחזיר הפנייה (קוד 302, Found) לדף home.html

answer- Service

שירות שיפעל רק עם אובייקטים Game מסוג GamePlayer.

NAME = /answer

קלט:

פרמטר letter אשר הוא רשימה שהערך היחיד הוא אות מתוך "A", "B", "C", או "D".

פלט:

String ריק

רושם את התשובה ואת זמן קבלת התשובה למשתמש.

getnames - XMLService

NAME = /getnames

שירות שיפעל רק עם אובייקטים Game מסוג GameMaster.

קלט:

מהשרת:

game - אובייקט Game מסוג GameMaster

המחלקה common

פלט:

מסמך XML המכיל בתוכו את שמות כל השחקנים במבנה הבא

<Root>

עבור כל שחקן

<playe name:"שם השחקן"/>

</Root>

disconnect_user - Service

NAME = /disconnect_user

קלט:

מהמשתמש:

pid - מספר עוגיית הזיהוי

מהשרת:

game - אובייקט Game

המחלקה common

פלט:

String ריק

מנתק את המשתמש מהמערכת

check_test - Service

NAME = /check_test

קלט:

מהמשתמש: join_number - מספר התחברות

מהשרת: המחלקה common

פלט:

xml boolean של האם קיים משחק עם מספר הצטרפות זה במערכת.

check_name - XMLService

NAME = /check_name

קלט:

מהשתמש:

join_number - מספר הצטרפות

name - שם מבוקש לבדיקה

מהשרת: המחלקה common

פלט:

xml boolean של האם אין שחקן במשחק של מספר הצטרפות עם השם הזה.
האם אפשר להתחבר למשחק עם שם זה.

join - Service

NAME = /join

קלט:

מהשתמש:

join_number - מספר הצטרפות

pid - מספר עוגיית הזיהוי, עם קיימת

שם - name

מהשרת: המחלקה common

פלט:

רושם את השחקן למשחק, יוצר לו אובייקט GamePlayer וכן עוגייה מתאימה ל'pid החדש. כמו כן,
אם השתמש מחובר למשחק אחר, מנתק אותו ממנו.
מחזיר הפנייה (קוד 302, Found) לדף game.html.

check_test_exist - XML Service

NAME = /check_test_exist

קלט:

מהשתמש:

quiz_name - שם השאלון המבוקש לבדיקה

מהשרת:

base_directory - המיקום של הקבצים בעלי הגישה, התיקייה Files.

פלט:

xml boolean של האם קיים במערכת שאלון בשם זה.

new - Service

NAME = /new

דף יצירת משחק חדש

קלט:

אין

פלט:

מחזיר הפנייה (קוד 302, Found) הפניה לדף home.html

get_join_number - XMLService

NAME = /get_join_number

קלט:

pid - ערך עוגיית ההזדהות מול המערכת

המחלקה common

פלט:

קובץ XML אשר מכיל בתוכו את מספר ההתחברות.

מבנה הקובץ

<Root>

<join_number>

מספר ההתחברות

</join_number>

</Root>

get_information - XMLService

שירות שיפעל רק עם אובייקטים Game מסוג GameMaster.

NAME = get_information

קלט:

game - אובייקט Game מסוג GameMaster

פלט:

קובץ XML הכולל מידע על המשחק: שם השאלון ומספר השאלות בשאלון

מבנה הקובץ

<?xml version="1.0" encoding ="UTF-8"?>

<Root>

<Quiz name="שם הקובץ" number_of_question="מספר השאלות">

</Quiz>

</Root>

set_timer_change - Service

NAME = /set_timer_change

קלט:

game - אובייקט Game

new_time - כמה שניות אנו רוצים להגדיר את הtimer

פלט:

String ריק

מגדיר את הtimer באובייקט game לכמה שניות שאנו מעוניינים לחכות.

check_timer_change - XML Service

NAME = /check_timer_change

קלט:

game - אובייקט Game

פלט:

בודק האם הטיימר "מצלצל", עבר הזמן שהגדרנו לו, מחזיר את התשובה כboolean.

order_move_all_player - Service

שירות שיפעל רק עם אובייקטים Game מסוג GameMaster.

NAME = /order_move_all_players

קלט:

game - אובייקט Game מסוג GameMaster

פלט:

String ריק

מורה לכל השחקנים של מנהל המשחק לעבור לחלק הבא.

order_move_all_not_answered - Service

שירות שיפעל רק עם אובייקטים Game מסוג GameMaster.

NAME = /order_move_all_not_answered

קלט:

game - אובייקט Game מסוג GameMaster

פלט:

String ריק

מורה לכל השחקנים של מנהל המשחק, שעדיין לא ענו, לעבור לחלק הבא.

check_move_next_page - XMLService

NAME = /check_move_next_page

קלט:

Game - אובייקט game

פלט:

boolean xml של האם על המשתמש לעבור לחלק הבא.

moved_to_next_page - Service

NAME = /moved_to_next_page

קלט:

Game - אובייקט game

פלט:

String ריק

המשתמש מאשר כי הוא עבר לחלק הבא.

move_to_next_question - XMLService

NAME = /move_to_next_question

קלט:

Game - אובייקט Game מסוג GameManager

פלט:

מעביר את המשחק לשאלה הבאה.

מחזיר קובץ XML במבנה הבא

<Root>

<question number_of_question="מספר השאלה הנוכחי"/>

</Root>

get_xml_leadeboard - XMLService

שירות שיפעל רק עם אובייקטים Game מסוג GameManager.

NAME = /get_xml_question

קלט:

Game - אובייקט Game מסוג GameManager

פלט:

לוח התוצאות כקובץ XML, מבנה הקובץ:

<Root>

עבור כל שחקן

<Player name="שם השחקן" score="שלו"/>

</Root>

get_question - XMLService

שירות שיפעל רק עם אובייקטים Game מסוג GameManager.

NAME = /get_question

קלט:

game - אובייקט Game מסוג GameManager

פלט:

קובץ XML של השאלה הנוכחית, מבנה הקובץ

```
<Question duration="30">
  <Text>השאלה עצמה</Text>
  עבור כל תשובה, ללא החלק של האם נכון או לא נכון
  <Answer>
    <Text>התשובה עצמה</Text>
  </Answer>
</Question>
```

check_move_question - XMLService

שירות שיפעל רק עם אובייקטים Game מסוג GameManager.

NAME = /get_move_question

קלט:

game - אובייקט Game מסוג GameManager

פלט:

xml boolean של האם צריך לעבור לשאלה הבאה

get_score - XMLService

שירות שיפעל רק עם אובייקטים Game מסוג GameManager.

NAME = /get_score

קלט:

game - אובייקט Game מסוג GameManager

פלט:

קובץ XML המכיל את מיקום השחקן ומספר הנקודות שצבר. מבנה הקובץ.

```
<Root>
  <score_place score="הניקוד" place="המיקום"/>
</Root>
```

start_question - Service

שירות שיפעל רק עם אובייקטים Game מסוג GameManager.

NAME = /start_question

קלט:

game - אובייקט Game מסוג GameManager

פלט:

String ריק.

מתחיל את השאלה על אובייקט GameManager, מגדיר את זמן התחלת השאלה

get_answers - XMLService

שירות שיפעל רק עם אובייקטים Game מסוג GameManager.

NAME = /get_answers

קלט:

game - אובייקט Game מסוג GameManager

פלט:

קובץ XML המכיל בתוכו את התשובות הנכונות. מבנה קובץ:

```
<Root>
```

```
  עבור כל תשובה נכונה
```

```
  <answer answer="התשובה"/>
```

```
</Root>
```

get_title - XMLService

שירות שיפעל רק עם אובייקטים Game מסוג GameManager.

NAME = /get_title

קלט:

game - אובייקט Game מסוג GameManager

פלט:

מחזיר קובץ XML של השאלה עצמה. מבנה הקובץ:

```
<Root>
```

```
  <title>
```

```
    השאלה עצמה
```

```
  </title>
```

```
</Root>
```

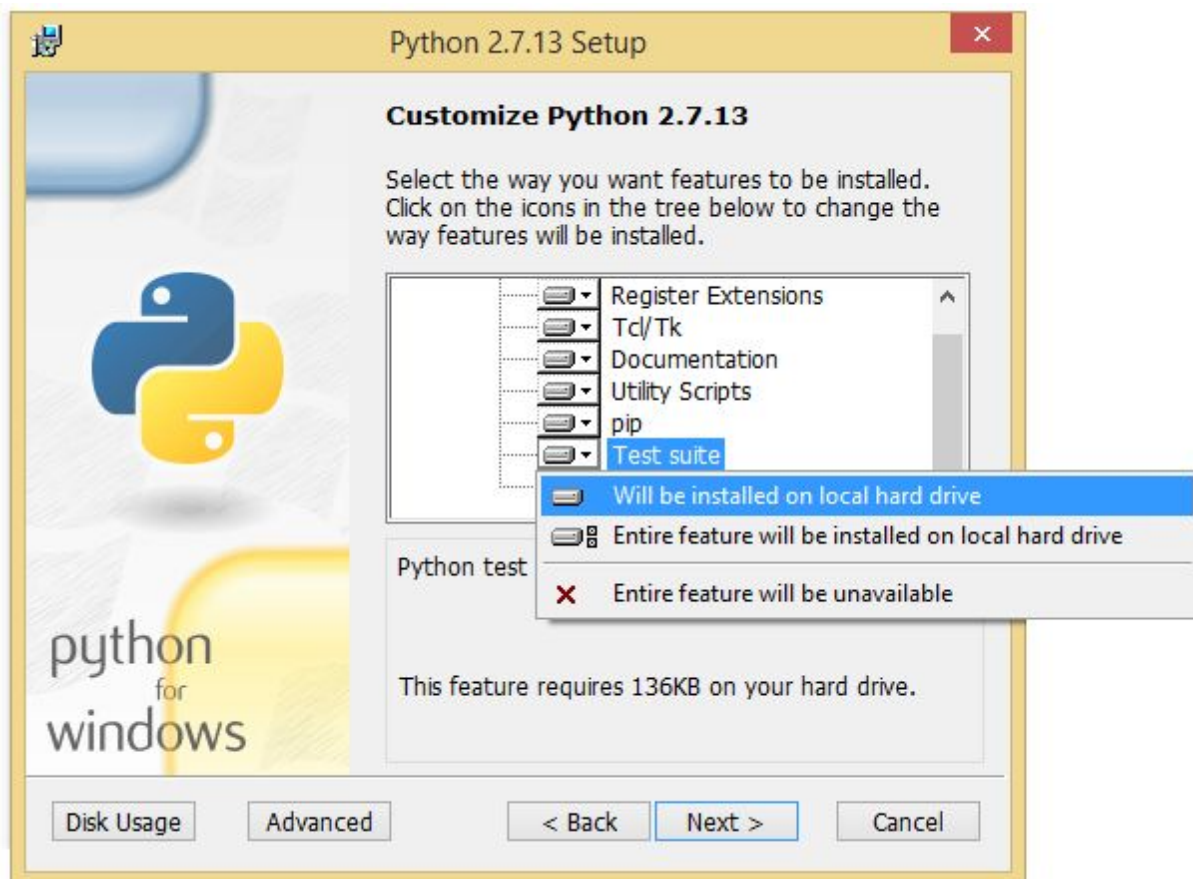

בעיות ידועות

ישנה בעיה ידועה אחת, והיא שכאשר כמות המשתמשים גדולה של משתמשים, המשתמש יכול לצאת מסינכרון.

התקנה ותפעול

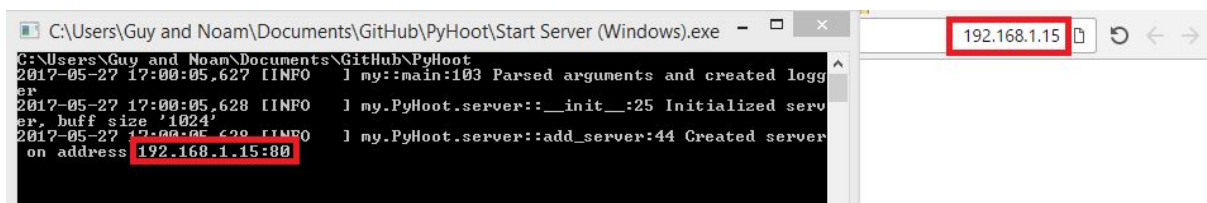
PyHoot

המערכת דורשת Python 2.7 על מנת לעבוד, ניתן להוריד בקישור [כאן](#).
בנוסף לכך, כאשר מתקינים, ב'א. Python 2.7 Customize' וודאו כי אתם לוחצים על 'Add
python.exe to Path' ואז 'Will be installed on local hard drive'.



הורידו את התיקייה מתוך GitHub Releases [\[קישור\]](https://github.com/Guy-Markman/PyHoot/releases/tag/v0.90) והעבירו אותה למיקום הרצוי במחשב.
<https://github.com/Guy-Markman/PyHoot/releases/tag/v0.90>

כיצד להתחיל את השרת?
 למשתמשי Windows (מומלץ):
 עברו לתיקיית PyHoot ולחצו על 'Start Server (Windows).exe' מתוך התיקייה.
 פעולה זו תתחיל את השרת על כתובת ה-IP שלכם עם פורט 80. כלומר, עכשיו תוכלו להצטרף למשחק עם כל מכשיר המחובר לרשת המקומית אליה מחובר השרת, על ידי כתיבת כתובת ה-IP של השרת בשורת הכתובת.
 אם אינכם יודעים את הכתובת של השרת, תוכלו למצוא אותה באחת השורות הראשונות של החלון שייפתח, לאחר המילה "address".



אם אתם משתמשים ב-shell \ command line (מומלץ למומחים ולמשתמשי Linux):
 התחילו את ה- command line או shell ועברו לתיקייה שבה שמתם את תיקיית PyHoot.
 רשמו את הפקודה הבאה ב-shell \ command line:

`python -m PyHoot`

פקודה זו תאפשר לכם להשתמש בפרמטרים שונים מברירות המחדל. תוכלו לעשות זאת על ידי שימוש בפורמט הבא:

`python -m PyHoot --[name of argument]=[the value of the arguement]`

בדרך זו תוכלו להוסיף פרמטרים ככל שתמצאו.

Arguments list:

-h, --help show this help message and exit
 --address ADDRESS [ADDRESS ...]

The address(es) we will connect to, default
['Your IP address:8080']

--buff-size BUFF_SIZE
Buff size for each time, default 1024

--base BASE Base directory

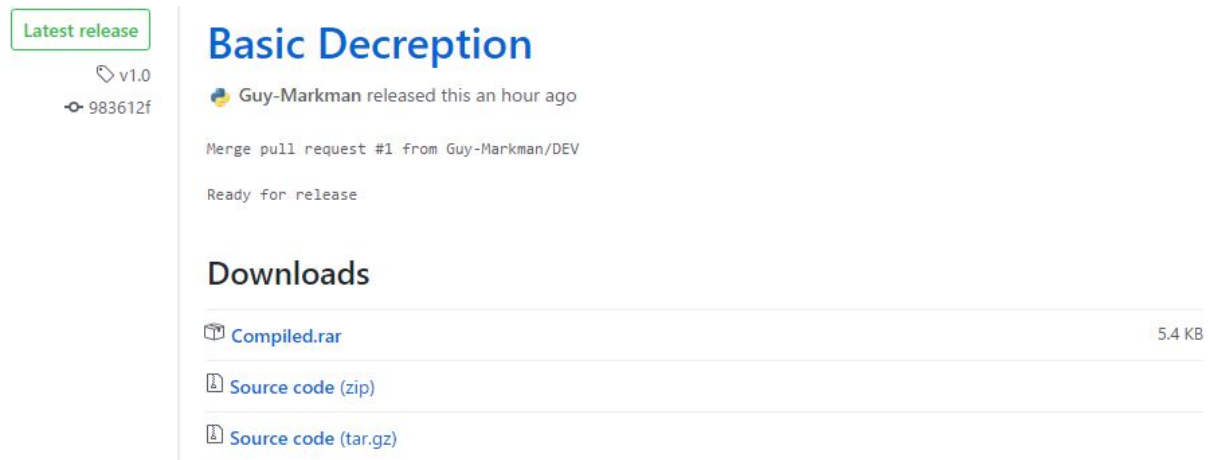
--io-mode {select, poll}
IO that will be used, default select.
In windows only select available.

--log-level {DEBUG,INFO,WARNING,CRITICAL,ERROR}
Log level

--log-file FILE Logfile to write to, otherwise will log to console.

PyHoot-Robot

הורידו את התיקיה מתוך Releases ב GitHub [קישור] והעבירו אותה למיקום הרצוי במחשב.
<https://github.com/Guy-Markman/PyHoot-Robot/releases>



Latest release

v1.0

983612f

Guy-Markman released this an hour ago

Merge pull request #1 from Guy-Markman/DEV

Ready for release

Downloads

Compiled.rar	5.4 KB
Source code (zip)	
Source code (tar.gz)	

על מנת להפעיל את הרובוט, השתמשו באותה דרך של ההפעלה המתקדמת של PyHoot

Argument List:

- h, --help show this help message and exit
- bind-address BIND_ADDRESS
Bind address, default: 0.0.0.0:0
- connect-address CONNECT_ADDRESS
The address we will connect to, default localhost:8080
- buff-size BUFF_SIZE
Buff size for each time, default 1024
- log-level {DEBUG,INFO,WARNING,CRITICAL,ERROR}
Log level
- log-file FILE Logfile to write to, otherwise will log to console.

תוכניות עתיד

ישנם שני סוגים של תוכניות נוספות לעתיד. הראשון לPyHoot והשני לPyHoot-Robot.

תוכניות לPyHoot:

- הוספת אפשרות לבניית שאלונים מתוך הדפדפן
- אפשרות להורדת מידע על פעילות כל אחד מהשחקנים בזמן השאלון
- שחקנים יוכלו לתת משוב על השאלון
- שחקנים יוכלו להמליץ על שאלות לעתיד
- אפשרות לבחור רק חלק מהשאלות בשאלון במשחק הספציפי

תוכניות לPyHoot-Robot:

- הרובוט לא יענה ישר אלה ימתין זמן מסויים של שניות ואז יגיב
- הרובוט לא יצדק בכל שאלה גם אם נמצאה התשובה הנכונה, אלה רק באחוז מסויים של המקרים.
- הצפנות יותר טובות אשר יותר מסובך לפתור אותן.

פרק אישי

במהלך הפרויקט למדתי הרבה, גם על נושאים של פיתוח תוכנה ורשתות וגם על היכולות, החוזקות והקשיים שלי כאדם וכמפתח.

התחלתי עם רעיון לשפר משחק שהכרתי בעבר, וחיפשתי את הדרכים המתאימות להכניס את השיפורים עליהם חלמתי. כדי להגיע למטרה זו השתמשתי בידע שהוקנה לנו במסגרת השעורים בבית-הספר וכן למדתי להשתמש בספריות חדשות ובטכנולוגיות חדשות. הלמידה דרך הרשת העמידה אותי מול אתגרים רבים. את חלקם צלחתי בקלות, לאחרים נזקקתי לעזרה. למדתי לבקש עזרה ממורי ומחבריי ואף מאנשים טובים ש"פגשתי" באינטרנט.

בתהליך למידה זה הגעתי לפיתוח של שרת חדש, מהבסיס ממש, ועד לשרותים השונים שהוא מאפשר, "הנדסתי" מבני-זיכרון מסובכים ואובייקטים שאפשרו לי להשתמש בידע שלי מתוך תיכנות מבוסס עצמים.

בתהליך זה למדתי את שפת פייתון לעומק רב, הרבה יותר משהכרתי קודם לכן, וכן למדתי שתי שפות חדשות: HTML ו-JAVA SCRIPT. כמו כן למדתי עיצוב באמצעות CSS. למדתי את מבנה-הנתונים של XML ואיך להשתמש בו כדי ליצור קבצים טובים יותר. בסך-הכל, עברתי בשנה האחרונה תהליך למידה מעמיק ואינטנסיבי, שמילא חלק מרכזי וחשוב בחיי, ואני גאה שהצלחתי להגיע למצב בו אני מסוגל לפתח מוצר כה מרשים.

אני מבקש להודות לשרית לולב ולאלון בר-לב על ההוראה ארוכת השנים, על עומק הלמידה שהעניקו לנו, לא רק של "חומר" אלא גם למידה "איך ללמוד", על השעות האין-סופיות של הדרכה אישית, כולל בשבתות ובחגים, בלילות ובשעות הבוקר המוקדמות. בלעדיהם פרוייקט זה לא היה יוצא לפועל, ועל כך אני מודה להם מקרב לב. אין מורים כאלה!

קוד פרויקט

GitHub - PyHoot - Master [\[link\]](#)

<https://github.com/Guy-Markman/PyHoot>

PyHoot-Robot - Master [\[link\]](#)

<https://github.com/Guy-Markman/PyHoot-Robot>

Documentation [\[link\]](#)

<https://drive.google.com/open?id=0B751jMgmuFrbX3pBNzhLRFd1YmM>

נספחים

קוד ל sequencediagram.org להצגת דיאגרמת Request באיכות טובה יותר

title Request

actor "משתמש" as User
 control "שרת" as Server
 database Client
 database Request
 database File

==Definitions==

par def(_send_my_buff)
 Client->User: Send buff, check if the user is still connected
 end

par def(_recv_data)
 User->Client: recv part of the code, add it to buff receive.
 end

par def(can_recv)
 Server->Client: Check if Client can receive more data
 Client->Server: Send if there is still enough memory available
 end

par def(can_send):
 Server->Client: Ask if there is data to send
 Client->Server: Answer if there is data to send and in in the right mode
 end

par def(read_buff):
 Client->File: Request part of the file
 File->Client: Return part of the file
 end

==Send And Receive Request==

autonumber 1
 par def(recv)

Server->User: Check if user want to send data

ref def(can_rcv)

parallel

Server->Client:

Client->Server:

parallel off

end

Server->Client: Order to Receive data

ref def(_rcv_data)

User->Client: The request

end

Client->Request: Test if request

Client-> Request: Create Request from the request

Client-> Client: Connect Game object if needed

Client->File: Create file from request

File->Client: Return requested arguments if needed

Request->Client: Create the response

Client->Client: Add response to buff

end

autonumber 1

par def(send)

Server->Client: Order client to send data

Client->User: Send status-line

loop While we didn't read all the data from file

Client->File: Request data from file

box over Server:Can stop any time user won't let sent to it for few moments

File->Client: Send data from file

Client->Client: Add data to buff

ref def(_send_my_buff)

Client->User: Send data from buff

end

Client->File: Check if we finished reading all the file

end

end