

תיק עבודה

street fighter 2017

מגיש:דוד בן יעקב ת״ז:320759921

עתיד-תיכון למדעים-לוד

**תוכן עניינים**

**נושא עמוד**

הקדמה 3

הגדרת העבודה 4

מדריך למשתמש 5

מדריך למפתח- תיאור מחלקות -76

מדריך למפתח- תיאור מבנה נתונים 8

מדריך למפתח- אלגוריתמים עיקריים 9-11

בעיות פתוחות והצעות לשיפורים 12

ביבליוגרפיה 13

**הקדמה**

בחרתי בנושא של משחק פעולה, מסגנון משחקי הפעולה של שני שחקנים משנות ה80 וה90, משום שאני זוכר שבילדותי אהבתי את סוגי המשחקי רטרו האלה, וכאשר הבנתי לאחר כמה שנים איך יצרו משחקים כאלה, הבנתי שאני רוצה ליצור אחד כזה בעצמי. כאשר הגעתי לתיכון ובחרתי במגמת המחשבים לא ידעתי אם אני אגיע לרמה של לבנות משחק כזה בעצמי, אבל התברר לי שזה אפשרי, אם אני אלמד הרבה ואבחר במסלול של ה10 יחל, ונוכחתי לגלות כי לקראת סוף לימודי בכיתה י״א היה לי משחק כמעט מוכן דומה למה שרציתי תמיד לעשות. המשחקים שהיוו לי השראה למשחק הזה היו המשחקים הידועים: street fighter וlittle fighter, וזאת היא גם הסיבה שבחרתי כמוזיקת רקע את מוזיקת הרקע האייקונית של little fighter 2. אני לא חושב כי המשחק שלי טוב יותר מהמשחקים האחרים שיצאו באותו תחום, אבל אני כן מרגיש שבניתי אותו בצורה של ניסיון לחבר בין הדור ההוא לבין הדור העכשווי, בעזרת שימוש בקולות אשר הכנתי בעזרת קולות של מכות ממשחקים עדכניים ובעזרת בניית פיקסל ארטים של דמויות שנראות כמו הדמויות הישנות ממשחקי הרטרו.

מטרת התוכנה/משחק היא ליצור סביבת משחק ידידותית לשני שחקנים, אשר יכולה להתבצע דרך שני מחשבים אשר מחוברים דרך האינטרנט, אך ניתן אף לפתוח בקלות שרת בתוך רשת מחשבים מקומית.

קהל היעד הוא כל אדם, אשר מרגיש כי הוא רוצה לחוות את חווית המשחק של שנות ה80 וה90 שוב פעם, או שאף בפעם הראשונה.

הטכנולוגיה המרכזית שהשתמשתי בה היא טכנולוגית הswing, אשר מאפשרת למתכנת ליצור פלט גרפי על מסך המחשב, אשר ניתן לשלוט בו על ידי קטעי קוד אשר גורמים לעצמים לזוז, וכך נוצרת התמונה הנעה של המשחק.

**הגדרת העבודה**

מטרתה המרכזית של התוכנה היא לשמש כמשחק פעולה רטרו משנות ה80 וה90 אשר ידידותי למשתמש ובעל זמן טעינה קצר ביותר, אך ניתן לשחק אותו רק במצב של שני שחקנים אחד נגד השני דרך שני מחשבים המחוברים דרך האינטרנט או דרך רשת מקומית הממוקמת על אחד מהמחשבים, כאשר הקמת רשת זו גם אינה כרוכה בהרבה מאמץ וזמן.

כיום כבר לא מייצרים הרבה משחקים בסגנון כזה וזה ממש מצער אותי משום שממש קשה למצוא משחקים כאלה, אשר אינם ממוקמים באינטרנט, ולכן רציתי ליצור אחד כזה בעצמי.

**מדריך למשתמש**

**הפעלה ראשונה של התוכנית**

1. לבלב
2. דלג
3. מריצים את הקבצים Server.bat ן – Client.bat, בסדר הזה

**הפעלה שנייה ואילך**

מריצים את הקבצים Server.bat ן – Client.bat, בסדר הזה

**הוראות למשחק**

בתחילת המשחק ישנה אפשרות לבחור את צבע השחקן שלך ואת השם שלו (אשר מיוצג על-ידי המילה הכחולה שנמצאת בשני הקצוות העליונים של המשחק).

ברגע שנפתח חלון המשחק יש קול שאומר ״Fight״ ולאחר מכן מוזיקת הרקע של המשחק מתחילה.

על-מנת לקבל שליטה על השחקנים יש ללחוץ על חלון המשחק, זאת על-מנת ליצור פוקוס עליו, לאחר מכן ניתן להתחיל במשחק עם המקשים הבאים:

מקשי התזוזה הם:

1. ← - על-מנת להזיז את השחקן לצד שמאל.
2. → - על-מנת להזיז את השחקן לצד ימין.
3. ↑ - על-מנת לקפוץ במקום, לא אפשרי בזמן תנועה.
4. ← + ↑ / → + ↑ - על מנת לבצע קפיצה גבוהה אשר זזה בכיוון שאליו היא מופנית, מפסיקה לזוז לצד בנתון אם משחררים מכפתור התנועה, לא ניתן לשנות את כיוון הקפיצה לאחר שהיא מתחילה.

מקשי הפעולה הם:

1. A - על-מנת להשתמש במכת אגרוף רגילה, מעלה 9 נקודות כוח לשחקן שנפגע.
2. S - על-מנת להשתמש בבעיטה רגילה, מעלה 9 נקודות כוח לשחקן שנפגע.
3. X - על-מנת להשתמש בבעיטה מסובבת, מורידה 200 נקודות כוח לשחקן שמשתמש בה (מיוצגות על ידי המספר הצהוב שליד המילה ״Power:״), בנוסף, היא מעלה לשחקן שנפגע 27 נקודות כוח.
4. Z - על-מנת להשתמש בשילוב של שתי מכות, מכה מלמטה ולאחר מכן מכה ישרה, מורידים 400 נקודות כוח לשחקן שמשתמש בו, בנוסף, הוא מעלה לשחקן שנפגע 40 נקודות כוח.

מקשים נוספים:

1. R - על-מנת לטעון מחדש את חלון המשחק אצל שני השחקנים, ניתן לשימוש בכל שלבי המשחק.

שינוי ברמת החיים של השחקן כתוצאה מפגיעה:

1. בעקבות מכת אגרוף רגילה, יורדים לשחקן 15 נקודות חיים (מיוצגות על ידי המספר האדום שליד המילה ״Hitpoints:״).
2. בעקבות בעיטה רגילה, יורדים לשחקן 15 נקודות חיים.
3. בעקבות בעיטה מסובבת, יורדים לשחקן 40 נקודות חיים.
4. בעקבות שילוב של שני מכות, יורדים לשחקן 60 נקודות חיים.

בסוף המשחק נשמע קול שאומר את המילים ״Game Over״, זאת בעקבות הגעת נקודות החיים של אחד השחקנים לערך 0, מה שמראה כי נגמרו לו החיים ולכן הוא הובס, מופעלת אנימציה של השחקן המובס נופל ונשאר על הרצפה, מופיעה על המסך ההודעה ״Game Over״ ומתחתיה המשפט ״\_\_\_\_ is the winner!״ כאשר במקום המקום הריק מופיע השם שניתן לשחקן שניצח עוד לפני תחילת המשחק.

**מדריך למפתח - תיאור מחלקות**

* **המחלקה UDPServer :**

תפקיד מחלקה זו היא ליצור את השרת המקומי שהשחקנים ירוצו עליו, מקבל נתונים מכל שחקן (UDPClient) ומחזיר חזרה לכל השחקנים, מהווה מקום קישור ותקשורת בין שני השחקנים, זאת בעזרת DatagramSocket אשר הוא משתמש בו כדי לקלוט נתונים מהשחקן, ולהעבירם לשחקן השני, מאפשר את הפעלת התוכנית על שני מכשירים שונים, על ידי חיבור שנעשה בעזרת פורט מספר 5000 על המכשירים. ללא מחלקה זאת התקשורת בין שני השחקנים תהיה בלתי אפשרית, משום שהנתונים ששחקן אחד יוצר לא יעברו לשחקן השני.

* **המחלקה UDPClient :**

תפקיד מחלקה זאת היא להוות את שלד השחקן, בעזרתו השחקן יכול לתקשר עם הסרבר, ועם שאר השחקנים המחוברים, הוא מבצע את התקשורת בינו לבין הסרבר בעזרת DatagramSocket, אשר ניתן למלא בנתונים ולאחר מכן לשלוח לסרבר, אשר מחזיר את נתונים לכל השחקנים, וככה הם יכולים לנתח את מה שמתרחש במשחק בזמן אמת, כאשר כל התוצאות של הפעולות מוצגות על שני המכשירים המשתתפים.

* **למחלקה PlayerData**, יש את כל המידע על כל שחקן, ובעזרת מחלקה זו, אשר יש ArrayList שלה בכל client, המידע על כל שחקן מתעדכן בכל מסך, מכיל גם פונקציות החזרה של כל סוג מידע בנפרד, פונקציית עדכון מיקום של שחקן- updateLocation, ופונקציית החזרה של כל המידע של השחקן בצורה של String, לשימוש של הclient- getStringRepresentation.
* **המחלקה GameBoard :**

המחלקה אשר מטפלת בכל מה שקורה על המסך בצורה גרפית, כוללת בתוכה UDPClient אשר בעזרתו הוא מעביר מידע לסרבר, אשר יכול לשלוח מידע לכל השחקנים, וכך ליצור את אותו מיצג גרפי על שני מסכי המכשירים המשתתפים.

בנוסף, הוא מכיל בתוכו SpriteFighter אשר אחראי לכל הפירוק והטיפול בפעולות של השחקן הנוכחי, מחזיר את הנתונים אל הPlayerData, אשר בעזרת הUDPClient שלו מעביר נתונים אלה לכל השחקנים.

הוא מכיל גם SpriteFighter נוסף, אשר מייצג את השחקן השני שמשחק, ובעזרתו מוצגות כל הפעולות של השחקן השני על המסך בצורה גרפית.

מכיל פעולות לתגובה אחרי שהשחקן לוחץ על כפתורי הפעולה (במקלדת) ואת פעולת ההרצה אשר גורמת לבדיקה בלתי פוסקת של משתנים בוליאניים אשר גורמים לכל הפעולות במשחק, לשינוי בערכים של הכוח והחיים של השחקן, ולציור מחדש של חלון המשחק בכל רגע של המשחק.

בנוסף, אחראית על בחירת השם של השחקן בתחילת המשחק.

* **המחלקה SpriteFighter :**

אחראית על בחירת צבע השחקן בתחילת המשחק, פירוק כל התמונות המכילות את תנועות השחקנים לתמונות נפרדות עבור כל חלק בכל תנועה, אשר אותן הוא מכניס למערכים של תמונות שמהם ניתן לזמן את התמונות, בGameBoard בזמן המצגת התמונה הגרפית על המסך, בנוסף מכיל את כל הפעולות אשר מטפלות בשחקן הנוכחי: תנועות, קולות שנוצרים בעקבות פעולות, ובדיקה של גבולות השחקן, על-מנת לקבוע האם הוא פגע בשחקן השני או לא, וכך לגרום לפלט דומה אשר מגיע חזרה לGameBoard.

מכילה את כל הנתונים הקשורים למיקום הנוכחי של התמונה הנוכחית בGameBoard, ושל השורה והטור של התמונה הנוכחית, כאשר בעזרת נתונים אלה השחקן יכול להית מוצג בצורה גרפית בGameBoard. הוא משנה את כל הנתונים האלו בזמן הרצת פונקציות הפעולות אשר לפעמים אף משנות את המיקום של השחקן (בקפיצה לצד ובהליכה).

**מדריך למפתח – תיאור מבנה נתונים**

* **קבצי תמונות:**

ממוקמים בפרויקט בתיקיות מחוץ לsrc כאשר השם שלהן הוא במבנה: ״fighter \_\_\_\_\_ movement sprites״ כאשר את הרווח ממלא השם של הצבע שבו כל השחקנים התמונות שבתוך התיקייה צבועים. כולם חוץ מהאחד של השחקן הכחול מכילים 9 תמונות כאלו, כאשר הכחול מכיל בנוסף את תמונת הרקע של המשחק כולו.

* **קבצי שמע (audio):**

ממוקמים בתיקייה בשם ״NewSoundsForGame״ בתוך הפרויקט מחוץ לתיקיית הsrc.

* **ArrayList <Integer> keysDown:**

מכיל את כל ערכי הKeyCode של כל הלחצנים אשר לחוצים בזמן הנוכחי, בעזרתם בודקת המחלקה Gui איזה פעולה השחקן הנוכחי מבקש לעשות והאם זה אפשרי לפי מהלך המשחק.

**מדריך למפתח – אלגוריתמים עיקריים**

* **התחברות לשרת-** במחלקה UDPClient- בבנאי אנחנו יוצרים קשר לDatagramSocket שבUDPServer דרך פורט מספר 5000.
* העברת מידע לסרבר- במחלקה UDPClient- ממומשת בפונקציה send מאפשרת עדכון של כל הנתונים שמשתנים בשני השחקנים בזמן אמת על שני המכשירים של שני השחקנים.
* **קבלת מידע מהסרבר-** במחלקה UDPClient- ממומשת בפונקציה receive, מעבירה את המידע לDatagramSocket של הclient הנוכחי ומשם היא מעבירה לפורמט של String ושולחת לעיבוד של המידע בפונקציה processMessageFromServer.
* **עיבוד מידע שמגיע מהסרבר-** במחלקה UDPClient- ממומשת בפונקציה processMessageFromServer, אשר מפרקת את הString שהגיע לשני חלקים, החלק של סוג ההודעה שהserver שלח, מיוצג על-ידי מספר שבינו לבין המידע יש סימן של נקודתיים ״:״, והמידע הוא כל שאר הString שישנו בהודעה, לאחר מכן הפונקציה בודקת איזה סוג הודעה נשלחה, ולפי סוג ההודעה היא יודעת איך היא צריכה לעבד את המידע שנשלח ולשלוח אותו הלאה, לGui.
* **סגירת הclient-** במחלקה UDPClient- ממומשת בפונקציה close, סוגרת את הDatagramSocket של הclient שנסגר וכך מנתקת את הקשר שלו לסרבר.
* **הרצת הclient-** המחלקה וUDPClientThread- מצורפת למחלקה UDPClient, מכילה את הפונקציה run, אשר מתחילה את הרצת הבדיקה התמידית של פונקציית הreceive.
* לשרת יש פונקציות דומות, חוץ מהחיבור לשרת שאין לו ובמקום זה יש לו חיבור לport מספר 5000 בבנאי של UDPServerThread אשר מצורך למחלקה של הUDPServer, בנוסף, בפעולת הrun שלו הוא לא רק מפעיל את פעולת הreceive שלו אלא גם את פעולת הgetActionFromPlayer, אשר מקבלת מידע מכל שחקן ולאחר מכן מעבדת אותו לפי סוג ההודעה בפונקציה processActionFromPlayer ואת התוצאות היא שולחת לכל השחקנים המחוברים בעזרת הclient שלהם, וזה בעזרת הפונקציה sendToAllLocations, אשר שולחת את כל המידע על כל שחקן לכל client, אשר משתמש במידע שהוא מקבל כדי לעדכן את המשחק במסך של השחקן שלו בצורה מידית.
* למחלקה PlayerData, יש את כל המידע על כל שחקן, ובעזרת מחלקה זו, אשר יש ArrayList שלה בכל client, המידע על כל שחקן מתעדכן בכל מסך, מכיל גם פונקציות החזרה של כל סוג מידע בנפרד, פונקציית עדכון מיקום של שחקן- updateLocation, ופונקציית החזרה של כל המידע של השחקן בצורה של String, לשימוש של הclient- getStringRepresentation.
* **בנאי של הGameBoard-** בונה 2 שחקנים אשר יופיעו לשחקן הנוכחי על המסך, אחד מהם הוא של השחקן הנוכחי ואחד של השני, לפי הid של השחקן, אשר נקבע בעת התחברות השחקן, בעזרת הclient שיש בGameBoard, לשרת, התחברות אשר מתבצעת לאחר יצירת הGameBoard, בmain של המחלקה, בעזרת פונקציית connectToClient, בנוסף הוא מתחיל את המרצת מוזיקת הרקע בלופים, מאתחל את הThread של המחלקה ובונה את הArrayList שמיועד לבדוק איזה מקשים נלחצו על ידי השחקן והאם לפי חוקי המשחק השחקן שלו אמור להגיב לכך.
* **פונקציה להצגה גרפית של כל העצמים של המשחק**, הpaintComponent היא מוזמנת על ידי הפקודה repaint, אשר ממוקמת בפונקציית הrun של המחלקה ומתבצעת כל הזמן, וזאת על מנת ליצור תמונה נעה של השחקנים.
* **פונקציית refresh**, מזומנת על-ידי לחיצה על כפתור R במקלדת של השחקן, כורמת לטעינה מחדש של כל המשחק לכל השחקנים המחוברים לרשת.
* **פונקציית shrinkIcon**, פונקציה המשמשת לשינוי גודלה של תמונה מהתמונות שנמצאות בפרויקט, ממומשת עוד על עצם של המחלקה, ImageIcon בשם background, אשר מייצג את תמונת הרקע של המשחק ונקבעת להיות בגודל שווה עם החלון, מוצגת בתחילת פונקציית הpaintComponent.
* **פונקציית main**, בונה GameBoard חדש, מחברת אותו לclient משלו, ומכינה JFrame אשר בו הוא ממוקם, מהווה את החלון של המשחק, ללא JFrame, אין למשחק חלון גרפי.
* **פונקציית hit**, מטפלת בשינוי נקודות החיים של כל שחקן ומעבירה את המשתנים הבוליאניים הקשורים למצב false בסוף.
* **פונקציית powerUp**, מטפלת בהעלאת הכוח של שחקן אשר נפגע ומעבירה את המשתנים הבוליאניים הקשורים למצב false בסוף.
* **פונקציית powerDown**, מטפלת בהורדת הכוח של שחקן אשר השתמש במכה אשר דורשת הורדה בכוח, בעיטה מסובבת או קומבו ומעבירה את המשתנים הבוליאניים הקשורים למצב false בסוף.
* **פונקציית keyPressed**, פועלת רק בתנאי שהמשתנה הבוליאני שקובע כי המשחק נגמר לא נמצאת המצב true, בודקת איזה כפתורים לחוצים באותו רגע, מכניסה אותם לArrayList של keysDown ומשם בודקת כל הזמן איזה כפתורים נמצאים בתוך הArrayList בכל רגע נתון, משמשת לתזוזה של השחקנים כדי שתתבצע בצורה רציפה כל עוד כפתור התנועה לחוץ, וזאת רק אם זה בהתאם לחוקי המשחק (לפי בדיקה של משתנים בוליאניים שלא יכולים להיות במצב true בזמן שמנסים לזוז).
* **פונקציית keyReleased**, פועלת רק בתנאי שהמשתנה הבוליאני שקובע כי המשחק נגמר לא נמצאת המצב true, בודקת איזה כפתורים שוחררו, ומוציאה את הערך שלהם מהArrayList של keysDown ובאותו זמן מבצעת את הפעולות של המכות לפי הכפתורים המתאימים שנלחצו, וזאת על ידי שינוי המשתנים הבוליאניים המתאימים של המחלקה למצב true, אשר בעזרתם הפעולות מתבצעות בפונקציית הrun ובמקרה של כפתורי התזוזה היא עוצרת את פעולת התזוזה על ידי שינוי הערכים הבוליאניים המתאימים לfalse ומעדכנת את השחקנים בהתאם, וזאת רק אם זה בהתאם לחוקי המשחק (לפי בדיקה של משתנים בוליאניים שלא יכולים להיות במצב true בזמן שמנסים לעשות פעולות ספציפיות).
* **פונקציית run**, בודקת בכל רגע נתון איזה ערכים בוליאניים של איזה שחקן נמצאים במצב true, וכך מריצה את הפונקציות של התנועה ושינוי הערכים של השחקנים בזמן אמת כל פעם כשלחצנים מלחצים, וזאת לפי כללי המשחק.
* פונקציית בנאי של SpriteFighter, מסדר את כל המיקומים של התמונות הראשונות של כל התנועות, לשני השחקנים שנוצרים בGameBoard, ככה שהם ימוקמו במקומות הנכונים, בנוסף הוא מציב תמונה התחלתית ככה שהמשחק לא יתחיל בלי שחקנים.
* **לאחר מכן יש פעולה static** ארוכה בSpriteFighter אשר המטרה שלה הוא לסדר את כל התנועות מכל התמונות של התנועות (sprites) במערכים מתאימים, אשר מהם הם יזומנו במהלך פונקציות התנועה והפעולה בסדר מסוים וזאת על מנת לשנות את התמונות שמוצגות בGameBoard וכך ליצור תמונה נעה.
* **כל פונקציות inbounds\_\_\_\_\_**, כאשר הרווח מוחלף בשם של תנועה מסוימת, בודקים האם בזמן תנועה של שחקן בצורה הספציפית שמתאימה לשם הפעולה, השחקן נוגע בגבולות של השחקן השני, ובמקרה של קפיצה לצדדים והליכה זה בודק גם האם השחקן נוגע בגבול של החלון, וזאת על מנת לא לאפשר לשחקנים ללכת אחד דרך השני, לצאת מהמסך בזמן קפיצה הצידה או הליכה, ועל מנת לבדוק האם המכות של השחקנים פגעו בתחום של השחקן השני וזאת על מנת להפעיל פעולות מתאימות להורדת כוח, חיים והעלאת כוח (במקרה וצריך) בGameBoard (זאת על ידי שינוי ערכים בוליאניים מתאימים בSpriteFighter לtrue, אשר בסוף תנועת המכה (הרצת הפונקציה) בודקים האם היא פגעה או לא ולפי זה משנה את הערכים המתאימים של המשתנים הבוליאניים בGameBoard).
* **כל הפונקציות מpunchHitting עד לstepTwo** הן פונקציות לזימון והפעלה של קטעי קול (audio), אשר ממוקמים בתיקייה NewSoundsForGame, משתמשים בכולם חוץ משלושה רק בSpriteFighter, בפונקציות שמריצות את הספרייטים של התנועות, וזאת לפי תזמון של צעדים, קפיצות, מכות שמפספסות (נבדקות על ידי הפונקציות שדיברנו עליהן בנקודה הקודמת), מכות שפוגעות, וקול של פגיעה בשחקן ונפילה אשר מורצות במקרים מיוחדים (פגיעה של המכות החזקות, בעיטה מסובבת וקומבו, או שרמת החיים של אחד השחקנים מגיעה ל0, מה שמעיד על כך שהוא הפסיד), הפונקציות gameOver וfight מזומנות רק בGameBoard, fight מזומנת בתחילת פתיחת חלון המשחק, ומריצה קול אשר אומר ״Fight״, וזאת על מנת לגרום לשחקן להבין שהוא אמור להילחם, ובסוף המשחק כאשר שחקן אחד מפסיד ומופיע ההודעה של הניצחון של השחקן השני על המסך, מופעלת הפונקציה gameOver, אשר מריצה קול אשר אומר ״Game Over״, הפונקציות falling וpainFalling, גם כן מורצות במקביל לgameOver, וזה משום שהשחקן שמפסיד נופל ו״מת״, בעת צעקה וקול של נפילה.
* **כל הפונקציות מwalk עד לalways**, הן פונקציות אשר מריצות את החלפת התמונות אשר מייצגות את תנועות השחקן, וחלקן גם משנות את המיקום של כל התמונות (walk וjumpSideways) וזאת על מנת לאפשר תזוזה של השחקן המהלך המשחק, התמונות שמוחלפות נכנסות למשתנה currentSprite, Image, אשר מטרתו לשמש כספרייט הנוכחי שGameBoard צריך להציג בpaintComponent, כל התנועות מתבצעות לפי חוקי המשחק, וזה על מנת לשמור על סדר ולמנוע באגים ואי-סדר ותאימות בין סיכויי השחקנים.
* **פונקציית הsetCol**, משמשת לעדכון הcolumn הנוכחי של השחקן, לאחר שסיים ללכת, מתממשת בGameBoard, בתוך פונקציית הkeyReleased.

**בעיות פתוחות והצעות לשיפורים**

* להוסיף דף menu לפני הכניסה למשחק, ששם כל הבחירות יעשו, ויוצגו כללי המשחק והמקשים שאיתם ניתן לשחק.
* לשנות את ההצגה של החיים והכוח מהצגה מספרית פשוטה להצגה גרפית של בר נקי עבור החיים, אשר יהיה מתחת לשם השחקן, ואשר יהיה מלא בהתאם לכמות החיים שנשארה לשחקן בצבע אדום, הבר של הכוח יהיה למטה, קרוב לתחתית החלון, והוא יתמלא ויתרוקן בהתאם לכמות הכוח שיש לשחקן בצבע צהוב, בנוסך ניתן להוסיף מספור של כמות המכות החזקות מכל סוד שניתן לעשות ברגע נתון עם כמות הכוח שיש לשחקן, יופיע מעל לבר הכוח. (בצורה של X2 או X3 וכן הלאה).

**ביבליוגרפיה**

* <https://stackoverflow.com/> - אתר שעזר לי הרבה לפתור בעיות שפגשתי במהלך התכנות.