**VRI>**

מסמך עיצוב

<יותם לויט>

<גרסה X>

<תאריך>

**היסטוריית גרסאות המסמך**

|  |  |  |
| --- | --- | --- |
| **תאריך** | גרסה | **תקציר השינויים** |
| 15/5/2019 | 1 | מסמך ראשוני |
|  |  |  |
|  |  |  |
|  |  |  |



**1. הקדמה**

1.1 מטרה

*מטרתו של מסמך זה, היא להציג את הפרויקט בצורה מפושטת ואת חלקיו. המסמך הזה מיועד לכל מי שרוצה להבין לעומק את הפרויקט ואיך הוא עובד בצורה פשוטה ככל שניתן בלי להיכנס לקוד.*

* 1. המוצר

*חלק זה צריך להכיל את הדברים הבאים (ניתן לתמצת מתוך מסמך האפיון):*

* *סימולטור VRI – Virtual robot interactions.*
* VRI הוא סימולטור בדיקה אשר מטרתו להקנות סביבה וירטואלית, שבה ניתן לבחון רובוטים ללא חשש לפגיעה פיזית בהם או בסביבה. יתרון VRI על פני בדיקה של רובוט בעולם הפיזי הוא שעל ידי בדיקה בסימולטור בעלי הרובוט יוכלו להימנע מתאונה מסוכנת בעולם הפיזי.

1.3 קישור למסמכים קודמים

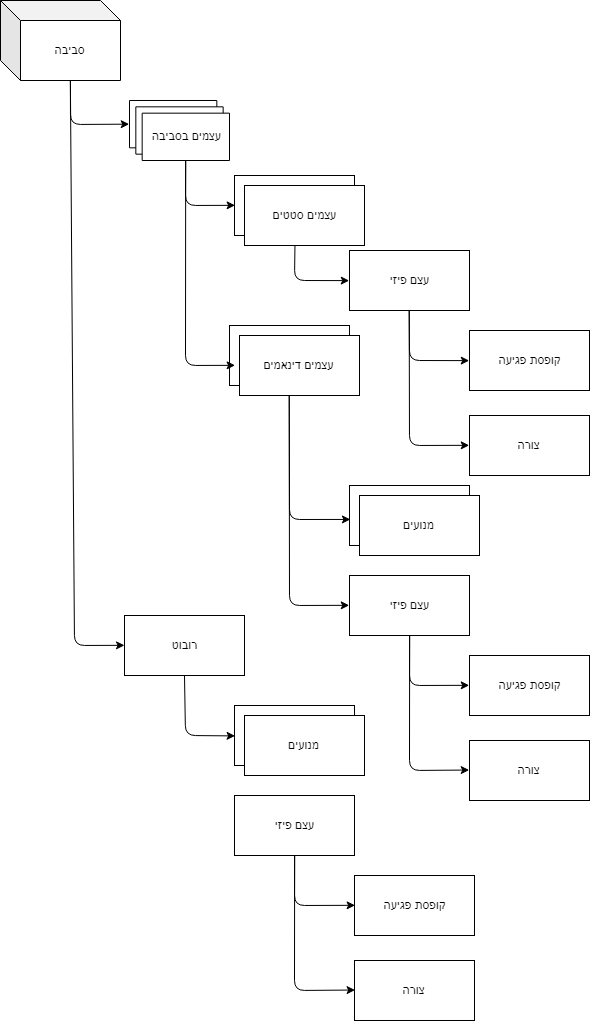
*קישור למסמכים / קבצים חיצוניים הרלוונטיים למסמך. לדוגמא: מסמך אפיון, עמוד web עם עיצוב הממשק הכללי שאתם מתכננים וכו'.*

1.4 הגדרות

*בחלק זה יש לציין את כל ההגדרות, המושגים וראשי התיבות בהם תשתמשו בהמשך המסמך.*

1. ארכיטקטורת המערכת

חלק זה כולל את תיאור מבנה המערכת ופירוט המודולים השונים בה



2.1 מבט על הארכיטקטורה:

ישנם 19 קבצי קוד המרכיבית ומפעילים את הפרוייקט

* שלושה מהם אחראים על ממשק המשתמש(מסך פתיחה, מסך קבלת מידע ומסך סימולציה). שמות הקבצים הם: open\_window\_vri, pre\_testing, Simulator.
* הקבצים Point, Vector ו-StraightLint הם בקבצים האחראיים על הגיאומטריה האנליטית בפרויקט.
* 2 קבצים האחראים על התנגשות בין אובייקטים בסביבה וצורה של האובייקטים נקראים: Hit\_Box ו-Parallelogram.
* הקובץ Object\_builder נותן סימולציה פיזית לעצמים שיהיו בסביבה וWall נותן לעצם משמעות בסביבה.
* Environment מרכז את כל העצמים והרובוט בסביבה אחת.

2.2 פירוט רכיבי המערכת

|  |
| --- |
| Environment - מחלקה |
| * Self.objects - מילון המכיל כמפתח את המספר הסידורי של האובייקט וכערך את האובייקט עצמו * Self.robot - הרובוט שיופעל בסביבה * Self.boundaries - עצם מסוג מקבילית ישרה שמדממה את הגבולות של הסביבה |
| \_\_inti\_\_ - initiation of object  Check\_robot\_in\_boundaries - **בודק אם הרובוט נמצא בגבולות הסביבה ומחזיר אמת אם הרובוט בגבולות ושקר אם הוא מחוץ**  Check\_onj\_robot\_crash**- בודק אם הרובוט התנגש באחד האובייקטים בסביבה אם יש התנגשות מחזיר אמת אם אין התנגשות מחזיר שקר**  **Check\_crash- בודק אם יש התנגשות של הרובוט בכללי מחזיר אמת אם יש ושקר אם אין**  **Add\_obj- מוסיף עצם למילון העצמים**  **- Add\_obj\_listמקבל רשימה של אובייקטים ומוסיף אותם למילון**  **Move\_robot - מקבל פעולה רצויה (ישר, אחורה, ימינה, שמאלה בשיטת הWASD) ומזיז את הרובוט לפיה**  **Get\_key - מקבל עצם שנמצא במילון ומקבל את המפתח שלו**  **Delete\_obj - מוחק אובייקט מהמילון**  **- Static - environment\_from\_file ממיר את קובץ בפורמט xml לסביבה**  **Static - data\_path - מקבל שם של קובץ ומחזיר את הכתובת שלו בספרייה של המידע**  **Static - handle\_objects\_from\_file- מטפל בעצמים בקובץ בפורמט xml**  **convert\_env\_to\_file - ממיר את הסביבה לקובץ בפורמט xml** |

|  |
| --- |
| **ObjectBuilder - יורש מObject** |
| * **Self.shape - צורה שמרכיבה את האובייקט** * **Self.hit\_box - מסגרת פגיעה ש לאובייקט** |
| **\_\_init\_\_ - initiation of object**  **Draw - מציין את האוביייקט**  **Move\_bt\_units - את האובייקט בסביבה (במנוע) לפי יחידות**  **Rotate - מסובב את האובייקט בסביבה (במנוע) במעלות**  **Static – find\_edge - מוצא את שתי הנקודות הכי קיצוניות של הצורה** |

|  |
| --- |
| **Robot - יורש מObjectBuilder-** |
| * **wheel - רדיוס הגלגל** * **Self.motor1 - מנוע אחד** * **Self.motor2 - מנוע שני** * **- Self.center\_lineווקטור נקודתי המתאר את מרכז הכובד של הרובוט ואת כיוון הניסיעה שלו** * **- Self.motor\_weightמשקל הרובוט** * **Self.shape - צורה שמרכיבה את האובייקט** * **Self.hit\_box - מסגרת פגיעה ש לאובייקט** |
| **Move - מקבל פעולה רצויה (ישר, אחורה,ימינה,שמאלה בשיטת הWASD) ומזיז את הרובוט לפיה**  **Rotate - מסובב את הרובוט בסביבה (במנוע) במעלות**  **convert\_robot\_to\_txt – ממיר את הרובוט למחזורת בפורמט xml**  **static – robot\_from\_file - ממיר את מחרוזת בפורמט xml לרובוט** |

|  |
| --- |
| **Wall – יורש מ- ObjectBuilder** |
| * **Self.shape - צורה שמרכיבה את האובייקט** * **Self.hit\_box - מסגרת פגיעה ש לאובייקט** |
| **convert\_obj\_to\_txt – ממיר את הקיר למחרוזת בפורמט xml**  **static – wall\_from\_file – ממיר פורמט xml לאובייקט קיר** |

|  |
| --- |
| **Parallelogram – יורש מ-Object** |
| * **self.relative\_angle – זווית בין הצלעות** * **self.main\_line – צלע מנחה מרכזית בצורה כיוונה ככיוון התנועה כמו כן נקודת ההתחלה של הקו היא נקודת ההתחלה של הצורה** * **self.relative\_line –צלע מנחה קו בזווית self.relative\_angle ביחס לצלע המרכזית** * **self.main\_line\_2 – קו שמקביל ל self.main\_line** * **self.relative\_line\_2 – קו שמקביל ל self.relative\_line** |
| **get\_lines – מחזיר את הקווים שמרכיבים את הצורה**  **change\_relative\_angle – מקבל זווית שתהווה כערך חדש לself.relative\_angle**  **move\_by\_units – מזיז את הצורה לפי יחידות**  **get\_middle\_point – מחזיר את נקודת המרכז הגיאומטרי של הצורה**  **change\_pos – מקבל נקודה שאליה נקודת ההתחלה של הצורה תזוז**  **change\_rotation – מקבל זווית ווקטור נדקודתי שמייצג את מרכז הכבידה ומסתובב סביבו אצ המעלות**  **change\_size – מקבל שתי אורכים ומציב אותם בקווים המנחים של הצורה**  **get\_points – מחזיר את הקודקודים של הצורה כאובייקט נקודה בלרשימה**  **get\_equation – מחזיר את המשוואות של הקווים שמרכיבים את הצורה**  **get\_point\_value – מחזיר את הערכים של הנקודות בתוך רשימה של tuples**  **draw – מצייר את הצורה**  **get\_edge\_points – מחזיר את שתי הנקודות הכי קיצוניות כtuple של עצם נקודה**  **convert\_shape\_to\_txt – ממיר את הצורה למחרוזת בפורמט xml**  **is\_Colliding – מקבל צורה ובודק אם ישנה התנגשות**  **parallelogram\_from\_file - ממיר את מחרוזת בפורמט xml לצורה** |

|  |
| --- |
| **Hit\_Box – יורש מParallelogram** |
|  |
|  |

**כאן יופיע פירוט לכל רכיב (ניתן לעשות באיטרציות – כל פעם עבור הרכיבים הרלוונטיים, לפני תהליך הפיתוח שלהם)**  
בסעיף זה יש לפרט את המבנה הפנימי של כל מודול/רכיב - כלומר:

* מבנה המחלקות השונות המרכיבות אותו, המאפיינים שלהם (public/private/protected + אילו טיפוסי משתנים כל פונקציה מקבלת ו/או מחזירה) והקשרים ביניהם (מי קורא למי, משתמש במי ולאיזה צורך) – מומלץ לצייר גרף UML (לא חייבים לפרט משתנים/פונקציות פרטיות אבל שיהיה ברור מה קשור למה ובאיזה אופן או לאיזה צורך)
* יש לתאר את ה- data flow / use case של שימושים שונים בכל תת-רכיב או פונקציה (מה קורה אם מתקבלים סוגים שונים של ערכים, איך/מדוע/באילו מקרים הרכיבים מתנהגים בצורה מסוימת או אחרת וכן הלאה)

**המלצות:**

* כדאי לתכנן רכיבים תשתיתיים עבור צרכים גלובליים במערכת - כמו למשל יצירת תקשורת רשתית, כתיבה וקריאה לבסיס נתונים (או קבצים), כתיבת תיעוד (log) הפעולות של המערכת (מקובל להגדיר "רמה" לכל שורה שנכתבת - debug, info, warning, error כדי שיהיה קל לסנן בזמן ריצת התוכנית. בנושא זה – מוזמנים להסתכל על מודול logging בפייתון).
* מומלץ לתכנן את הרכיבים באופן שיאפשר אבסטרקציה ומודולריות – בכדי לאפשר הרחבה עתידית ותחזוקה קלה. אם המימוש הפנימי של רכיב או מודול מסוים משתנה אז שאר הרכיבים/מודולים אינם מושפעים או נפגעים מכך ואין צורך לשנות גם אותם, או כאשר רוצים להוסיף פונקציונליות/רכיב חדש אז אין צורך לשכתב את כל שאר המערכת במיוחד בשבילו.

2.3 דיון בנושא העיצוב הנבחר

כאן יש לפרט ובעיקר להסביר מדוע בחרתם דווקא בחלוקת הרכיבים/תפקידים הזו ולא אחרת. מה היתרונות שלכם מבחינתה, וגם מה החסרונות שאתם מודעים אליהם. במידה וחשבתם על חלופות אחרות, יש לציין אותן ולהסביר בקצרה כיצד הן שונות מהפתרון שנבחר - ומדוע החלטתם בסופו של דבר שלא לבחור בחלופות האלו.  
כאן יש להתייחס גם לגבי שפת התכנות שבה בחרתם לכתוב כל רכיב, ומדוע דווקא בשפה זו ולא אחרת.

1. עיצוב נתונים ופרוטוקולים

כאן יופיע תיעוד של מבני נתונים שונים / פרוטוקולים המשמשים אותנו במערכת

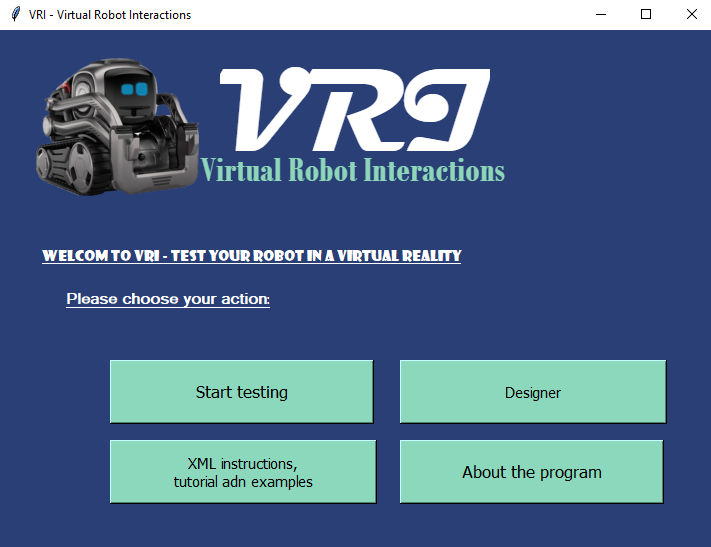
בסעיף זה יש לפרט את כל סוגי המידע אשר מועבר או נשמר במערכת - בין אם בין רכיבים, בין מודולים או בכל צורה אחרת (כלומר גם אם זה משמש רק לתקשורת פנימית ולא ע"ג הרשת, או למשל אם זה משהו ששומרים לקובץ/בסיס נתונים).

* לכל סוג מידע כזה יש לפרט על השדות אותן הוא מכיל ומאיזה סוג כל שדה, מה טווח הערכים הרלוונטי לגביו וכל הגבלה או מידע נוסף אחר שאתם מוצאים לנכון (למשל: האם מותר שהשדה יהיה ריק ומה המשמעות של זה, אורך או ערך מינימלי/מקסימלי למספר או למחרוזת)
* יש לציין לאיזו מטרה משמש כל מבנה נתונים.
* עבור פרוטוקול תקשורת-נתונים בין רכיבים שונים:
* מה המצבים השונים לכל סוג של תקשורת (למשל אותנטיקציה, הרשמה, שליחת קובץ וכו')
* באילו ייצוגי מידע נעשה שימוש + תרשים זרימה של המצבים השונים שיכולים להיות וסדר השלבים שלהם (למשל כיצד נעביר מידע על מקרים של סיסמה שגויה, משתמש חסום, התחברות מוצלחת וכו')
* מומלץ להגדיר מראש קודים של בקשות/פעולות, תוצאות/תגובות ושגיאות אשר ישותפו בין כך הרכיבים הרלוונטיים
* יש להתייחס לאופי המידע הנשמר (האם הוא טקסטואלי או בינארי) ולייצגו בצורה נוחה והולמת בהתאם
* יש לפרט אודות מבנה בסיס הנתונים (חלוקה לטבלאות, השם והטיפוס של כל עמודה/שדה, קשרים בין טבלאות ואילוצים כלשהם על עמודות)

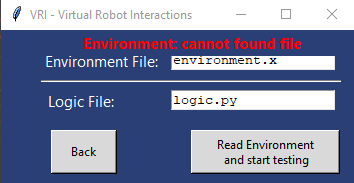
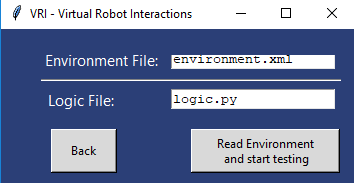
1. ממשק משתמש

**ממשק המשתמש של הפרויקט מורכב ממספר מסכים:**

1. **מסך פתיחה – החלון הראשון אשר נפתח עם פתיחת התוכנה. בחלון ניתן לראות את לוגו הפרויקט וארבעה כפתורים לבחירה. כפתור להפעלת הסביבה, כפתור לתוכנת עיצוב שעמה ניתן לבנות סביבה מותאמת אישית, כפתור להסבר על פורמט ה-**XML **שאיתו ניתן לתכנת סביבה וכפתור למידע על התוכנה.**

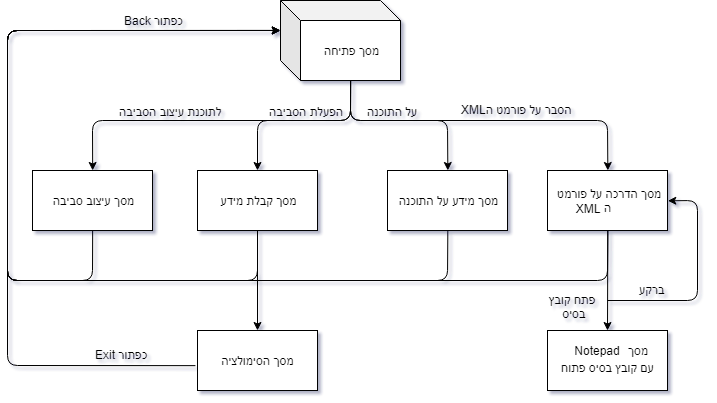
****

1. **מסך קבלת המידע – החלון שנפתח כאשר בוחרים בכפתור הפעלת הסביבה. בחלון יש אפשרות לשנות את ברירת מחדל של הקובץ של הסביבה והקובץ של לוגיקת הרובוט. כמו כן ישנם שתי כפתורים. האחד כדי לחזור למסך הקודם והאחד כדי להמשיך עם הפעלת הסביבה. אם יש בעיה של השמות של הקבצים אז על החלון תקפוץ הודעת Error.**

**מסך רגיל: מסך עם הודעת Error:**

1. **מסך הסימולציה – בחלון זה יש שורת תפריטים ובתוכה כפתור הפעלה, עצירה, שמירה של המצב הקיים ויציאה מהסימולציה כמו כן יש מחוון כדי לשלוט במהירות של ההפעלה ותווית שבה מוצגת המצב של הרובוט והסביבה (מופעל, עוצר, התנגשות).**



**דיאגרמה המתארת את מפת הGUI:**

בסעיף זה יש לפרט את הפונקציונליות של המערכת כפי שהיא מתבטאת עבור משתמש חיצוני. במידה ויש סוגים שונים של משתמשים, יש להתייחס לכולם בהתאם - אילו רכיבים או נתונים רלוונטיים עבורם, וכיצד הם מתקשרים איתם.  
  
בנוסף, יש לצרף המחשות ויזואליות של המסכים השונים, ולהסביר את התוכן שלהם (למשל מה התפקיד של כל שדה או כפתור, מה קשור/תלוי במה - למשל כפתור שמכובה בהתאם לתנאים מסוימים במערכת וכן הלאה) והקשרים ביניהם (איזה מסך מוביל לאיזה מסך ובאילו מקרים)  
  
דוגמה (לא מלאה):  


5. נספחים

**כל דבר שהייתם רוצים להוסיף בנוגע לעיצוב המערכת**