Erlang Project

8/8/2019

The Nightmare Garden

נאור דהן
ניר קוברובסקי

מטרת הפרויקט

התנסות מעשית בתכנות פונקציונלי בשפת erlang וישום העקרונות הנלמדו במהלך הסמסטר המהווים חלק אינטגרלי בשפה.

תקשורת במערכות מבוזרות על ידי שימוש בעקרונות השפה הפונקציונלית ובארכיטקטורה של gen server, אשר ממדל מערכת יחסים בין שרת ולקוח.

כמו כן התנסנו בעבודה עם qlc אשר מעניק ממשק משתמש נוח להפנית שאילתות ל-MNESIA (תקשורת מבוזרת) אשר בעבודתינו לקחה חלק בשמירת ועידכון המידע עבור התצוגה הגרפית ונתוני השרתים השונים.

תאור הפרוייקט -

בעבודתינו הקמנו סימולציה לגינה בה גדלים פרחים הסובלים מבעיות שונות אשר מוגרלות אקראית בכל מרווח זמן אקראי, בהתאם לקושי שהוגדר מראש בהגדרות ריצת הסימולציה.

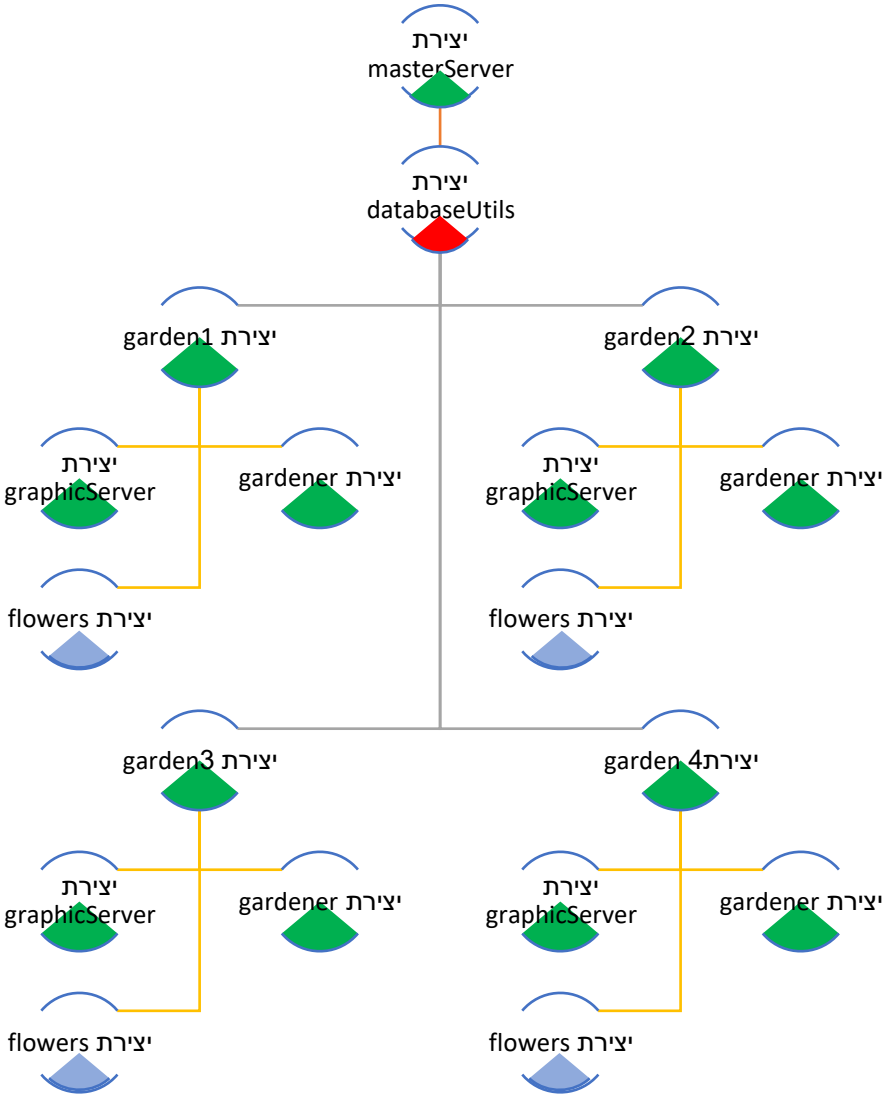
עבור טיפול בבעיות אלה מטיילים בגינה גננים אשר דואגים להשקות את הצמח במקרה והתיבש, או לרסס אותו מפני מזיקים התוקפים אותו.

עבור כל מזיק ועבור מצב של התיבשות, קיים מרווח זמן נסבל ומוגדר מראש בו הפרח יכול לשרוד. המערכת תשאף בכל עת לשלוח גנן פנוי לעבודה אל הפרח הקרוב ביותר למיתה ובמידה והגן לא יגיע אליו בזמן הקצוב, הפרח ינבל ויעלם מהמפה ואילו המערכת תעצור את הגנן ותתן לו משימה אחרת.

המערכת בנויה כגינה אחת גדולה, אשר מחולקת בין מחשבים שונים וכל מחשב אחראי על המרחב שלו. כאשר גנן נדרש להגיע לפרח הנמצא בחלק אחר (מחשב אחר), המערכת יודעת לנטב את הגנן ל-נ.ב המצאים של הפרח, באופן יחסי, ולהעביר את הגן גרפית בין המחשבים השונים.

חשוב להדגיש הגנן איננו משוייך לחלק מסוים של גינה, או למחשב מסוים, ויכול לנדוד גרפית בין המחשבים השונים, לטפל בבעיות ולהסתובב בגינה הגדולה כאוות נפשו.

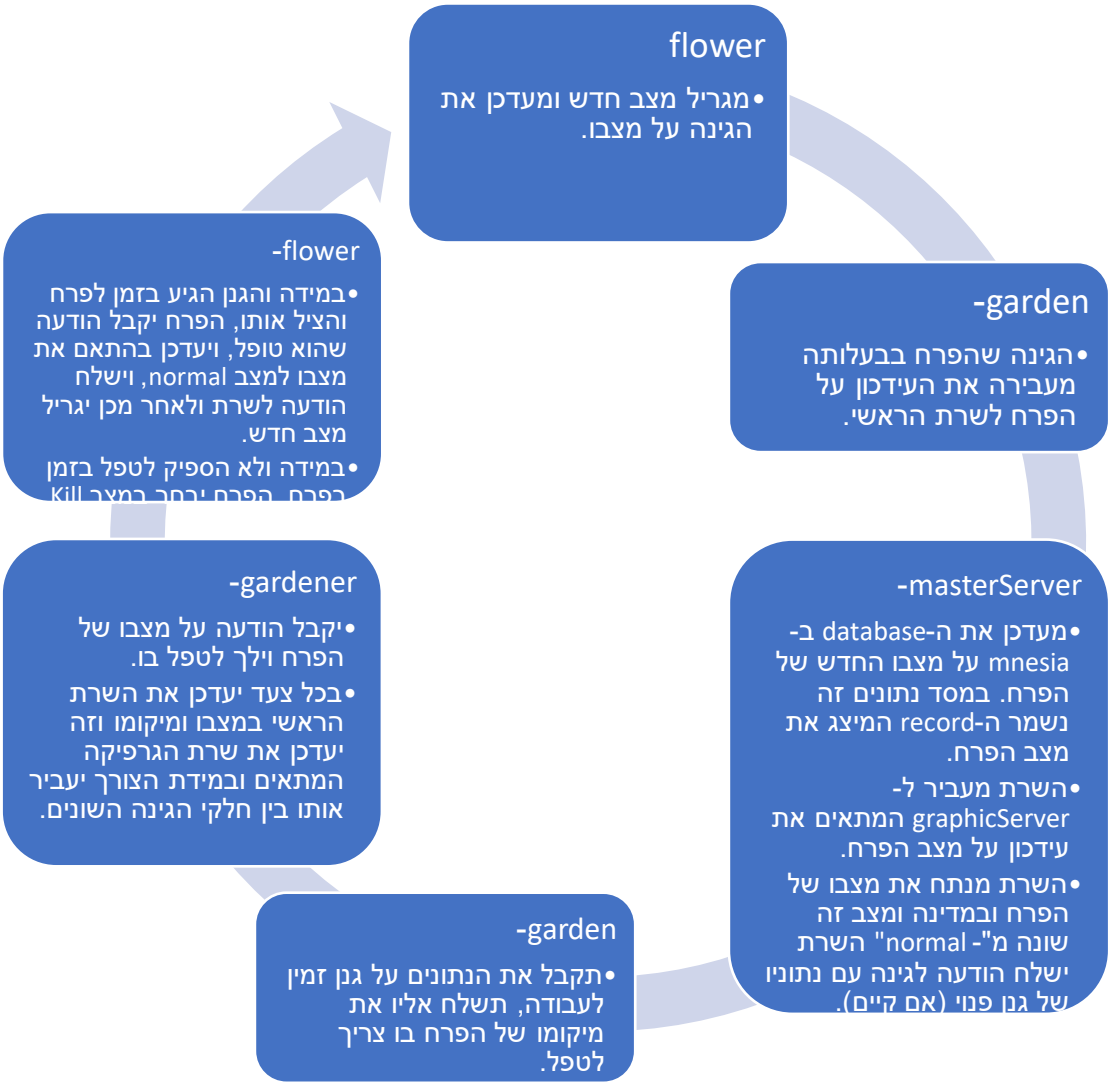
תאור המערכת -



כאשר:

- נוצר ידנית תרם הופעלה הסימולציה.
- נוצר בזמן ריצת האפליקציה על ידי קודקודי השורש.
- Gen server
- תהליך המדמה מכונת מצבים.
- Database

הטיפול בפרח: שרשרת החלפת הודעות



מרכיבי המערכות: מודולים

masterServer-

אחראי על הקישוריות בין הגינות, אחראי על הקישוריות והעידכונים בין האובייקטים השונים ושרתי הגרפיקה וכן אחראי בלעדי על עידכון מסד הנתונים לגבי המצב הנתון של כלל האובייקטים – גינות, גננים ופרחים.

פונקציות:

Handle cast

1. newFlower-
   - מעדכן את שרת הגרפיקה ליצור פרח חדש.
   - מעדכן את מסד הנתונים לגבי ישות חדשה של פרח ומעדכן בה את הנתונים.
2. newGardener-
   - שולח לשרת הגרפיקה המתאים בקשה להציג גנן חדש בגינה המתאימה.
   - מעדכן במסד הנתונים.
3. changeFlowerStatus-
   - שולח לשרת הגרפיקה המתאים לעדכן את מצבו של הפרח מבחינה גרפית.
   - מעדכן את מסד הנתונים.
   - בודק האם קיים גנן פנוי לטפל בפרח במידה והפרח בסקנה ובמידה וכן שולח הודעה מתאימה לגינה עם נתוניו של הגנן והפרח שצריך טיפול.
4. updateFlower-
   - מעדכן את מצבו של הפרח במסד הנתונים.
5. deleteFlower-
   - שולח בקשה משרת הגרפיקה המתאים למחוק פרח במפה לאחר שפרח זה מת. עם הבקשה נשלחים נתוני הפרח הרלוונטים.
6. gardenerWalkToFlower-
   - מעדכן במסד הנתונים את מצבו של הגנן הספציפי.

7. changeGardenerLocation-
- שולח בקשה עם נתוני הגנן והמיקום הקודם שלו, משרת הגרפיקה לעדכן גרפית את מיקומו של הגנן וידמה הליכה.
- מעדכן את מסד הנתונים בנתוניו החדשים של הגנן.

8. changeGardenerGarden-
- שולח בקשה לשרת הגרפי המתאים למיקומו הקודם של הגנן למחוק את הגנן מהמפה.
- שולח בקשה לשרת הגרפי המתאים למיקומו החדש של הגנן להציג את הגנן בחלק הגינה העכשיוי.
- מעדכן את שרת הנתונים.

9. gardenerResting-
- שולח בקשה לשרת הגרפי המתאים להושיב את הגנן.
- מעדכן את מסד הנתונים.
- בודק האם קיים פרח הזקוק לעזרה ובמידה וכן שולח הודעה מתאימה לגינה עם נתוני הפרח והגנן.

Flower-

הפרח מדמה מערכת מצבים סופית כאשר בכל פעם שהוא מטופל על ידי גנן עבור בעיה מסוימת, הוא מגריל מצב חדש. דחיפות הסקנות בה יפגוש תלויה בדרגת הקושי שהגדרנו לתוכנית, הנעה בין 0 ל-60 כאשר ב-0 הפרח יגריל מצב "normal" בהסתברות 0.5 ובדרגה 60 יגריל מצב זה בהסתברות 0.

הודעות:

1. updateStatus-
- מגריל מצב חדש עבור עצמו. הודעה זו נשלחת לו על ידי עצמו לאחר שהמצב הקודם טופל.
- מעדכן את הגינה על מצבו החדש.

2. {setGardenerID, NewGardenerID}
- מקבל הודעה על כך שגנן מדרך לטפל בו.
- במידהו הפרח ימות לפני שהגנן יגיע, באמצעות מספר מזהה של הגנן המערכת תדע לעצור את הגנן בכדי שלא יתקדם לחינם.
- מעדכן את הגינה שתעדכן את השרת הראשי שיכתוב למסד הנתונים.

3. handleProblem
- במצב זה הגנן זה עתה מטפל בפרח והפרח ממתין לגמר הטיפול.

4. Kill
- במידה והגנן לא טיפל בפרח בזמן, הפרח שולח לעצמו הודעת המתה.
- הפרח שולח הודעה נוספת על עידכון במצבו לגינה ומודיע לה שהוא מת.

מצב after– בכל 3 שניות הפרח נכנס למצב after וכך למעשה סופר "זמן חיים". כאשר זמן החיים שספר לאחר מספר פעמים נכנס למצב after, הוא משווה את זמן חיים זה לעומת הזמן הדרוש לאותה סקנה שהוא נמצא בה להרוג אותו ובמידת הצורך שולח הודעת Kill.

-Gardener

הגנן במצב הרגיל שלו נח, עד שהוא מקבל משימה לטפל בפרח. אז מחשב את המיקום אליו צריך להגיע בכדי להיות ליד הפרח ומנסה להצילו.
הגנן ממומש כ-gen server ומתקשר באופן רציף עם הגינה ועם הפרחים אליהם הוא הולך, אך לא עם השרת הראשי או שרת הגרפיקה בצורה ישירה.

פונקציות:

Handle cast

1. -cancelWalk
- עוצר את הליכתו ומעביר את עצמו למצב resting.
- מעדכן את הגינה בהתאם.
2. -walkToFlower
- מעדכן את מצבו.
- מעדכן את הגינה שתעדכן את הפרח שגנן בדרך אליו.
- נכנס לפונקצית walking.

פונקציות עזר

1. -Rest
- שולח הודעה לגינה שהוא במצב מנוחה, והגינה מעדכנת את השרת הראשי שמעדכן את הגרפיקה ואת מסד הנתונים.

2. Walking-
   - הגנן מחשב את המיקום הגלובאלי של הפרח ואת הדרך שצריך לעשות בכדי להגיע אליו.
   - בתוך לולאה הוא מבצע צעדים עד אשר מגיע לפרח ובכל צעד הוא מעדכן את הגינה על מיקומו.
3. calcNewDest
   - פונקצית עזר אשר ממירה את המיקום של הפרח, אשר מנורמל לגודל מסך של חלק אחד מגינה, למיקום גלובאלי.
   כך הגנן יודע לאיזה מיקום להגיע, ובמקרה הצורך לעבור בין גינות.
4. moveGarden-
   - הגנן מזהה שבצעד הנוכחי לאחר שינוי המיקום הוא יופיע בחלק אחר של הגינה (מסך אחר) ומעדכן את הגינה. הגינה מעדכנת את השרת הראשי שמעדכן את מסד הנתונים ואת שרת הגרפיקה למחוק את הגנן מחלק הגינה הקודם, ולמקמם אותו מחדש בחלק הגינה הבא.
5. isCanceledWalk-
   - בודק אם הוא צריך לעצור במידהו הפרח שלח הודעה שהוא מת, ואין טעם להמשיך ללכת אליו.
6. isArrive-
   - בודק אם הגיע למיקומו של הפרח.
7. handleFlower
   - מבצע טיפול בפרח, ומעדכן בהתאם את הגינה.

graphicServer-

אחראי על עידכון המסך ויצירת אובייקטים חדשים תוך כדי תקשורת רציפה עם השרת הראשי לגבי עידכון מצב האובייקטים. השרת הראשי ממומש כ-wx_object שמתפקד כ-gen server.

Handle cast

1. Update-
   - מקבל flower record ומעדכן בהתאם את תמונת הפרח במיקום המתאים לפי המצב המופיע באותו record. קורא לפונקציה updateFlowerStatus שתבצע את ההחלפה.
2. deleteGardenerFromGarden-
   - במקרה שגנן עובר בין חלק גינה, פונקציה זו נקראת ושרת הגרפיקה מוחק את תמונת הגנן ושם במקומה את האובייקט שהיה קודם לכן.

3. makeSteps-
   - שולח לפונקצית makeSteps את המיקום הישן של הגנן ואת ה-gardener record שהוא מקבל לפונקציה.
4. newFlower-
   - מקבל flower record עם נתוני עבור מיקום הפרח החדש וסוגו, ושולח את הנתונים הללו לפונקצית עזר drawNewFLower.
5. Rest
   - מקבל gardener record ומושיב את הגנן שינוח.
6. addGardener
   - פונקציה זו נקראת לאחר שהגנן עבר גינה, ושרת מקבל בפונקציה זו את מיקום הגנן המנורמל לגודל של חלק יחיד, ומצייר אותו בחלק הגינה החדש (כלומר מחליף בין המסכים).

פונקציות עזר

1. initializeGraphicWindow-
   - טוען את כלל התמונות בסימולציה.
   - פותח את החלון הראשי ומצייר עליו את המדשאה.
   - מאתחל מטריצת מיקומים ריקה עבור אובייקטים שיוספו בהמשך.
2. updateFlowerStatus-
   - מקבל flower record המכיל את מיקום הפרח ואת מצבו החדש, ובהתאם לכך דורס את התמונה במיקום הרלוונטי ומצייר במקומה תמונה חדשה עבור הפרח.
   - מעדכן את מטריצת האובייקטים בהתאם.
3. drawNewFLower-
   - מקבל flower record ומצייר על המצב פרח חדש.
   - מעדכן את מטריצת האובייקטים שתשמור את המצב החדש.
4. makeSteps-
   - מקבל את סוג הגנן, מיקומו הקודם והמיקום אליו רוצה להגיע, מצייר את הגנן במיקום החדש.
   - מצייר מחדש את האוביקטים שהיו ממוקמים במיקומו הקודם של הגנן.
5. sitDownTheGardener-
   - מקבל gardener record ומעדכן במיקום הנדרש את תמונת הגנן לישיבה.
6. initObjectMatrixAsMap
   - מאתחל את מטריצת האובייקטים שתעזור לזכור מה היה בכלל המיקומים בתוכנית לשם תזוזת הגננים.

7. fillAvailableCoordinateList-
   - עוזרת לפונקצית initObjectMatrixAsMap
8. updateObjectStatusInObjectsMatrix-
מקבלת אובייקט חדש ומיקום, ומחליפה את האובייקט החדש באובייקט ישן במידה והם באותו מיקום.
9. drawFromRecovery-
   - מקבל רשימה של flower record ורשימה של gardener record ומצייר אותם על המסך לאחר נפילת התוכנית.
   - הנתונים נשאבים ממסד הנתונים על ידי השרת הראשי/ אחת הגינות האחרות ומועברים לפונקציה זו שמאתחלת מחדש את חלק הגינה ומציירת עליו את כלל האובייקטים שהיו קיימים בזמן הנפילה.

Flowr-

הפרח הינו מכונת מצבים סופית אשר מגרילה לעצמו בעיות באופן הסתברותי.
לאחר שהגריל בעיה, הפרח מעדכן את הגינה והיא מעדכנת את השרת.
מצב 1: הפרח במצב Normal ולכן לא צריך לטפל בו.
מצב 2: הפרח במצב של סקנה. תמונתו משתנה על ידי שרת הגרפיקה והשרת הראשי שולח אליו גנן פנוי.
מצב 3: הפרח ניצל בזמן על ידי הגנן – כלומר, זמן החיים שלו המתאים לבעיה הנתונה עדיין לא נגמר כשהגנן הגיע אליו.
מצב 4: הפרח מת, התהליך נסגר והתמונה שלו נעלמת מהסימולציה.

מסקנות-

1. תכנון מוקפד יותר עבור פרויקטים גדולים מסוג זה יכול לחסוך זמן רב.
2. התעסקות עם גרפיקה, שברקע עובר מידע רב, הינה דורשת קוד יעיל ואף הרבה עבודה בכדי לתזמן את התהליכים והגרפיקה יחדיו.
3. בפרויקט שלנו ישנם שלושה שרתים גרפים, שלושה שרתים המדמים גינה, שרת ראשי המקשר בין כולם ובעל גישה למסד הנתנים וכן שרתים נוספים עבור הגננים.

מאחר ומדובר בתהליכים רבים הרצים ברקע, וצריכים לתקשר האחד עם השני, התקשנו לדאוג לכך שהסינכרון יהיה מושלם, אך ניתן לראות במהלך הסימולציה שעל אף הקושי הצלחנו להגיע למצב בו הגננים ניגשים לפרחים בסקנה, הגרפיקה תואמת לצעדייהם המחושבים בקוד, הצלחנו לעצור את הגננים מלהתקדם לחינם במקרה ופרח מת ועוד.

4. העבודה עם ממשק gen server מפשטת מאוד את תהליך כתיבת הקוד ומספקת צורה נוחה ויעילה לתקשורת בין תהליכים, ועל אף כמות גדולה של הודעות ליחידת זמן נראה כי הסימולציה פועלת כשורה.

הפעלת הפרויקט-

יש לקמפל את הפרויקט עם גירסאת ארלנג 20 ומעלה בכדי לתמוך ב-mnesia.

- מרימים את השרת הראשי בטרמינל מספר 1, מתוך תקיית הפרויקט וכותבים

➢ Erl -name master -setcookie xxx
➢ masterServer:start().

- בטרמינל של מחשב מספר 2, מתוך תקית הפרויקט

➢ Erl -name garden1 -setcookie xxx
➢ garden:start_link(1, 'master@IP').

- בטרמינל של מחשב מספר 3, מתוך תקית הפרויקט

➢ Erl -name garden2 -setcookie xxx
➢ garden:start_link(2, 'master@IP').

- בטרמינל של מחשב מספר 4, מתוך תקית הפרויקט

➢ Erl -name garden3 -setcookie xxx
➢ garden:start_link(3, 'master@IP').

דגשים:

1. **יש לקמפל את קבצי הפרויקט בכל מחשב עם גירסאת ארלנג גבוהה מ20.**
2. **IP הינו ה-IP של המחשב הראשון.**
3. **יש לבצע את ההוראות לפי הסדר הנ"ל בכדי שהקוד ירוץ.**