ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Σχεδιασμός-Δημιουργία εφαρμογής διάγνωσης βλαβών σε κινητά τηλέφωνα

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΜΠΑΜΠΑΚΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ

(ΑΕΜ: 2793)

ΧΑΛΚΙΑΣ ΓΕΩΡΓΙΟΣ

(ΑΕΜ: 2459)

Επιβλέπων : Δόσης Μιχαήλ

Καθηγητής

Καστοριά Μάιος - 2022

ΠΑΝΕΠΙΣΤΗΜΙΟ ΔΥΤΙΚΗΣ ΜΑΚΕΔΟΝΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ

Σχεδιασμός-Δημιουργία εφαρμογής διάγνωσης βλαβών σε κινητά τηλέφωνα

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΜΠΑΜΠΑΚΟΣ ΚΩΝΣΤΑΝΤΙΝΟΣ

(ΑΕΜ: 2793)

ΧΑΛΚΙΑΣ ΓΕΩΡΓΙΟΣ

(ΑΕΜ: 2459)

Επιβλέπων : Δόσης Μιχαήλ

Καθηγητής

Εγκρίθηκε από την τριμελή εξεταστική επιτροπή την ημερομηνία εξέτασης

.....	.....	.....
Ον/μο Μέλους	Ον/μο Μέλους	Ον/μο Μέλους
Ιδίοτητα Μέλους	Ιδίοτητα Μέλους	Ιδίοτητα Μέλους

Καστοριά Μάιος - 2022

ΕΥΧΑΡΙΣΤΙΕΣ

Θα θέλαμε να εκφράσουμε τις ευχαριστίες μας στον επιβλέποντα καθηγητή μας κ. Δόση Μιχαήλ για την συνεργασία και την πολύτιμη συμβολή του όπως επίσης και τον καθηγητή κ. Μπάτο Παναγιώτη για την καθοδήγηση που μας παρείχε καθ' όλη τη διάρκεια της παρούσας πτυχιακής εργασίας.

Τέλος ένα μεγάλο ευχαριστώ στις οικογένειάς μας για την στήριξη που μας έδωσαν.

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή εργασία πραγματεύεται στη διαδικασία διάγνωσης βλαβών κινητών τηλεφώνων. Αρχικά γίνεται η μελέτη σε υπάρχουσες έρευνες διαγνωστικών εφαρμογών με σκοπό την εξοικείωση του αναγνώστη με την εργασία. Στην συνέχεια αναφέρονται παραδείγματα από εμπορικές εφαρμογές που αφορούν κινητά τηλέφωνα με τα χαρακτηριστικά και τις λειτουργίες τους. Επιπλέον παρουσιάζεται η διαγνωστική εφαρμογή μας πρώτα σε επίπεδο διαγραμμάτων κι έπειτα σε επίπεδο ανάλυσης κώδικα. Η εφαρμογή έχει υλοποιηθεί σε περιβάλλον SWI-Prolog και λειτουργεί με ερωτήματα συμπτωμάτων στο χρήστη με σκοπό την εύρεση πιθανής βλάβης του κινητού του τηλεφώνου.

Λέξεις Κλειδιά: Διαγνωστική εφαρμογή, Υποθέσεις, Συμπτώματα, Βλάβη, SWI-Prolog, Κινητά τηλέφωνα

ABSTRACT

This thesis deals with the process of diagnosing mobile phone failures. Initially, the study is done in existing research of diagnostic applications in order to familiarize the reader with the work. The following are examples of commercial applications related to mobile phones with their features and functions. Then our diagnostic application is presented first at the level of diagrams and then at the level of code analysis. The application has been implemented in a SWI-Prolog environment and works with symptom queries to the user in order to find possible damage to the mobile phone.

Key Words: Diagnostic Application, Cases, Symptoms, Damages, SWI-Prolog, Mobile Phones

ΠΙΝΑΚΑΣ ΠΕΡΙΕΧΟΜΕΝΩΝ

ΛΙΣΤΑ ΕΙΚΟΝΩΝ

ΛΙΣΤΑ ΠΙΝΑΚΩΝ

ΕΙΣΑΓΩΓΗ

Η εργασία αυτή έχει ως στόχο την δημιουργία μιας διαγνωστικής εφαρμογής σε κινητά τηλέφωνα σε γλώσσα Prolog στην οποία ο χρήστης θα εισάγει τα συμπτώματα της κινητής συσκευής και η εφαρμογή θα αναζητεί τα συμπτώματα στην γνωσιακή βάση δεδομένων για την εμφάνιση της πιθανής βλάβης.

Το κεφάλαιο 1 παρουσιάζει τι είναι ένα διαγνωστικό πρόγραμμα και τις κατηγορίες του, καθώς και κάποια ερευνητικά άρθρα στους τομείς της υγείας και των οχημάτων. Παρέχει επίσης πληροφορίες για τα εργαλεία που χρησιμοποιούνται συνήθως για διαγνωστικές εφαρμογές βλαβών και αναφέρονται διαγνωστικές εφαρμογές εμπορικής χρήσης για κινητά τηλέφωνα.

Στο κεφάλαιο 2 παρουσιάζονται η ανάλυση και η παρουσίαση των διαγραμμάτων δέντρων αποφάσεων και δραστηριοτήτων της εφαρμογής που δημιουργήθηκαν στο πλαίσιο της πτυχιακής εργασίας.

Στο κεφάλαιο 3 πραγματοποιείται η ανάλυση και η παρουσίαση του κώδικα της εφαρμογής. Συγκεκριμένα αναφέρονται κομμάτια του κώδικα και γίνεται η επεξήγηση του βήμα προς βήμα. Τέλος παρουσιάζονται τα βήματα και παραδείγματα εκτέλεσης της εφαρμογής.

ΚΕΦΑΛΑΙΟ 1ο: Εφαρμογές Διάγνωσης Βλαβών

1.1 Διαγνωστικό πρόγραμμα

Ένα διαγνωστικό πρόγραμμα (γνωστό και ως Test Mode) είναι μια αυτόματη ακολουθία προγραμμάτων υπολογιστή που καθορίζει τη λειτουργική κατάσταση εντός του λογισμικού, του υλικού ή οποιουδήποτε συνδυασμού αυτών σε μια οντότητα, ένα σύστημα ή ένα δίκτυο συστημάτων. Τα διαγνωστικά προγράμματα παρέχουν στον χρήστη καθοδήγηση σχετικά με τυχόν ζητήματα ή προβλήματα που εντοπίζονται κατά τη λειτουργία του [1].

1.1.1 Μέθοδοι λειτουργίας διαγνωστικών

Το διαγνωστικό πρόγραμμα για μια συσκευή ή σύστημα μπορεί να βρίσκεται ή να ενσωματωθεί ανεξάρτητα. Αυτές οι μέθοδοι λειτουργίας είναι διατεταγμένες, κατά σειρά αυξανόμενης πολυπλοκότητας και αυξανόμενης αξίας διαγνωστικών πληροφοριών.

- **Παρακολούθηση παρασκηνίου δεικτών συστήματος,** για στατιστική ανάλυση τάσεων και καταγραφή μη φυσιολογικών γεγονότων.
- **Διαγνωστικά βασισμένα σε λύσεις,** που ελέγχουν για γνωστούς τρόπους αποτυχίας προσδιορίζοντας εάν ανιχνεύονται τα γνωστά συμπτώματά τους.
- **Μαύρο κουτί,** το οποίο δοκιμάζει έναν μηχανισμό χωρίς να γνωρίζει πώς λειτουργεί, και απλώς εστιάζει στην ακρίβεια των δεδομένων εξόδου που βασίζονται σε μια γνωστή είσοδο.
- **Λευκό κουτί,** το οποίο χρησιμοποιεί γνώση των εσωτερικών λειτουργιών ενός μηχανισμού για άμεσες δοκιμές.
- **Προσανατολισμένη στη λειτουργία**, ένας συνδυασμός μαύρου και λευκού κουτιού, με μία ή περισσότερες λειτουργίες μαύρου κουτιού που παρεμβάλλονται με μία ή περισσότερες λειτουργίες λευκού κουτιού. Αυτός ο τρόπος δοκιμής δεν προτιμάται, ωστόσο, ορισμένα πολύπλοκα συστήματα δεν διαθέτουν τις απαραίτητες διεπαφές για να εκτελούν τον ένα ή τον άλλο τύπο ανεξάρτητα.
- **Ενσωματωμένα διαγνωστικά παρασκηνίου,** που εκτελούν δοκιμές στοιχείων του συστήματος κατά τη διάρκεια του χρόνου αδράνειας ενός συστήματος.

- **Διαγνωστικά διαπλοκής λειτουργίας,** που ενσωματώνουν διαγνωστικά στην κανονική λειτουργία ενός στοιχείου του συστήματος, επομένως οποιοσδήποτε οριακός τρόπος λειτουργίας διαγιγνώσκεται αμέσως. Παραδείγματα στοιχείων υλικού με δυνατότητες που βοηθούν ένα διαγνωστικό πρόγραμμα είναι:
    1. Οι σύγχρονοι σκληροί δίσκοι διαθέτουν εντολές Self-Monitoring, Analysis and Reporting Technology (SMART) που παρέχουν πληροφορίες σχετικά με τις συνθήκες εσωτερικών σφαλμάτων για παράδειγμα: τις μετρήσεις ωρών λειτουργείας, τις μετρήσεις κακών τομέων και άλλα.
    2. Ορισμένα συστήματα ενδέχεται να χρησιμοποιούν μνήμη κωδικού διόρθωσης σφάλματος (ECC) που καταγράφει συμβάντα αποτυχίας μνήμης που διορθώθηκαν αυτόματα [1].

Στα πλαίσια της πτυχιακής μας εργασίας θα σχεδιάσουμε και θα υλοποιήσουμε διαγνωστική εφαρμογή ακολουθώντας την μέθοδο «Διαγνωστικό βασισμένο σε λύσεις»

1.2 Ερευνητικά δεδομένα

Οι διαγνωστικές εφαρμογές στις μέρες μας αποτελούν μεγάλο κομμάτι της τεχνολογίας σε διάφορους τομείς όπως ασθένειες, καρκίνο, υπολογιστικά συστήματα , δίκτυα βιομηχανίες αυτοκίνητων και πολλά άλλα.

1.2.1 Υγεία

Μαζί με τις θεραπευτικές αγωγές και τα εμβόλια, τα διαγνωστικά αποτελούν την κρίσιμη «τρίτη επιτακτική ανάγκη» που συχνά παραβλέπεται για τη μείωση της επιβάρυνσης από ασθένειες. Με καλύτερα διαγνωστικά έρχεται πιο ακριβής αναγνώριση ασθενειών, πιο κατάλληλες θεραπείες και χαμηλότερο κόστος. Σε πολλές περιοχές του κόσμου, ωστόσο, ορισμένες από τις πιο θεραπεύσιμες ασθένειες και συνθήκες παραμένουν μη διαγνωσμένες εξαιτίας της έλλειψης κατάλληλων ή διαθέσιμων διαγνωστικών.

Οι ρόλοι των διαγνωστικών στην υγεία είναι:

- Να κάνουν την διαφορά στην ζωή ασθενών
- Ενημέρωση κατάστασης της ασθένειας.
- Οδηγός ιατρικής παρέμβασης.
- Να οδηγεί σε βελτιωμένα αποτελέσματα υγείας [2].

Στη συνέχεια θα δούμε ένα παράδειγμα από το άρθρο του Jimmy Singla με τίτλο «Η διάγνωση ορισμένων πνευμονικών παθήσεων σε έμπειρο σύστημα σε γλώσσα Prolog».

Αυτό το εξειδικευμένο ιατρικό σύστημα χρησιμοποιείται για τη διάγνωση των κύριων πνευμονοπαθειών μεταξύ των ασθενών. Η διάγνωση γίνεται λαμβάνοντας υπόψη τα συμπτώματα που μπορούν να φανούν ή να γίνουν αισθητά. Αυτό το ιατρικό εξειδικευμένο σύστημα βοηθά τον γιατρό ή τον ειδικό να κάνει την κατάλληλη διάγνωση του ασθενούς. Οι πνευμονικές παθήσεις έχουν πολλά κοινά συμπτώματα και μερικά από αυτά μοιάζουν πολύ. Αυτό δημιουργεί πολλές δυσκολίες στον πνευμονολόγο να καταλήξει σε μια σωστή απόφαση ή διάγνωση. Αυτό το έμπειρο σύστημα μπορεί να άρει αυτές τις δυσκολίες και γνωρίζει τριάντα δύο πνευμονικές παθήσεις και υλοποιήθηκε στην SWI Prolog.

Ένα κομμάτι κώδικα του παραπάνω διαγνωστικού είναι το εξής:

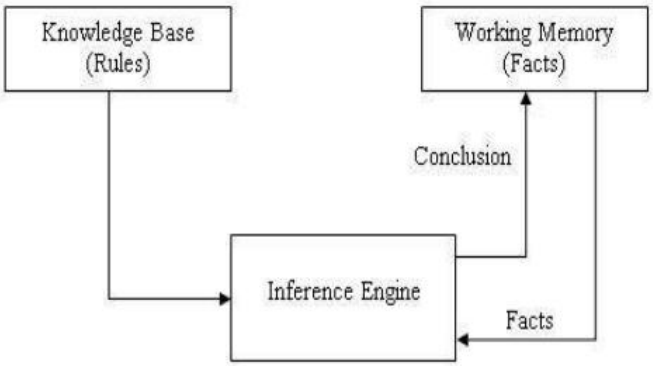
```
Disease (Patient, pneumoconiosis):- Symptom (Patient, chronic_cough),
                                    Symptom (Patient,shortness_of_breath)
```

Σύμφωνα με τα άρθρο αυτό το έμπειρο σύστημα Prolog εφαρμόζεται με επιτυχία και λαμβάνονται αποτελέσματα. Χρησιμοποιείται σε πολλούς ασθενείς και τα αποτελέσματά του είναι 70% σωστά [3].

1.2.2 Οχήματα

Ένα παράδειγμα από το άρθρο του Aggarwal Rashi και άλλων με τίτλο «Προσέγγιση για τη διάγνωση αστοχίας αυτοκινήτου - Ένα έμπειρο σύστημα». Η έρευνα έγινε για να βοηθήσει στο σχεδιασμό ενός έμπειρου συστήματος για τη διάγνωση και τις επισκευές αστοχίας αυτοκινήτου υπό περιορισμούς όπως ο χρόνος, ο τόπος και η διαθεσιμότητα της ανθρώπινης τεχνογνωσίας. Πραγματοποιήθηκε μελέτη τεχνολογιών για το σχεδιασμό έμπειρων συστημάτων για να συναχθούν τα καλύτερα

μέσα, τα οποία είναι απλά και εύκολα, να εφαρμοστούν και να διατηρηθούν κατά την ανάπτυξη ενός έμπειρου συστήματος. Στο παρακάτω σχήμα 1 βλέπουμε την λειτουργία του έμπειρου συστήματος.

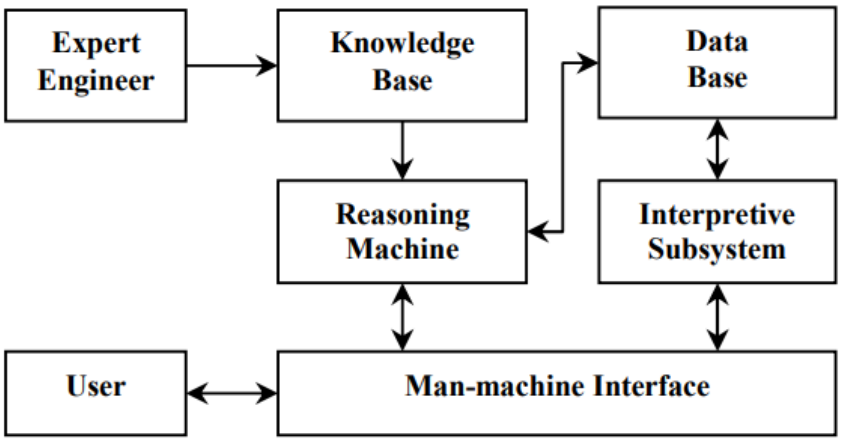


Σχήμα 1 Λειτουργία έμπειρου συστήματος [4]

Η ανάλυση του συστήματος που αναπτύχθηκε δείχνει ότι η γνώση που αποκτήθηκε στη φάση απόκτησης γνώσης οδηγεί στο σχεδιασμό του συστήματος που περιλαμβάνει τη συνολική δομή της γνώσης του συστήματος, το τμήμα προγραμματισμού και τις διεπαφές. Αφού εξετάστηκαν όλα τα στοιχεία ενός έμπειρου συστήματος, δημιουργήθηκε ένα πρωτότυπο έμπειρου συστήματος για τη διάγνωση αστοχίας αυτοκινήτου χρησιμοποιώντας τη βάση γνώσεων, βάση κανόνων που ορίζεται στη Prolog, η διαδικασία παρείχε πολύτιμες πληροφορίες για τα στάδια που οδηγούν στην ανάπτυξη ενός επιτυχημένου έμπειρου συστήματος [4].

Ένα άλλο παράδειγμα διαγνωστικού είναι του Zhuo Liu,και άλλων με τίτλο «Σχεδίαση και υλοποίηση διαγνωστικού βλαβών έμπειρου συστήματος για ελικόπτερο» Ένας συγκεκριμένος τύπος ελικοπτέρου χρησιμοποιείται ευρέως στην Κίνα. Με στόχο τα χαρακτηριστικά της διάγνωσης βλαβών, καθιερώνεται το έμπειρο σύστημα των ηλεκτρονικών και των οργάνων του συστήματος του ελικοπτέρου με βάση την Visual Prolog. Η γνώση του συστήματος αποκτάται με χειροκίνητη μέθοδο. Η βάση γνώσεων δημιουργείται σύμφωνα με τον κανόνα παραγωγής. Ο μπροστινός συλλογισμός χρησιμοποιείται για την εξαγωγή συμπερασμάτων διάγνωσης σφαλμάτων. Τέλος, η ορθότητα και η αποτελεσματικότητα του συστήματος επαληθεύεται από το παράδειγμα αστοχίας γεννήτριας.

Στο παρακάτω σχήμα 2 βλέπουμε την λειτουργία του έμπειρου συστήματος.

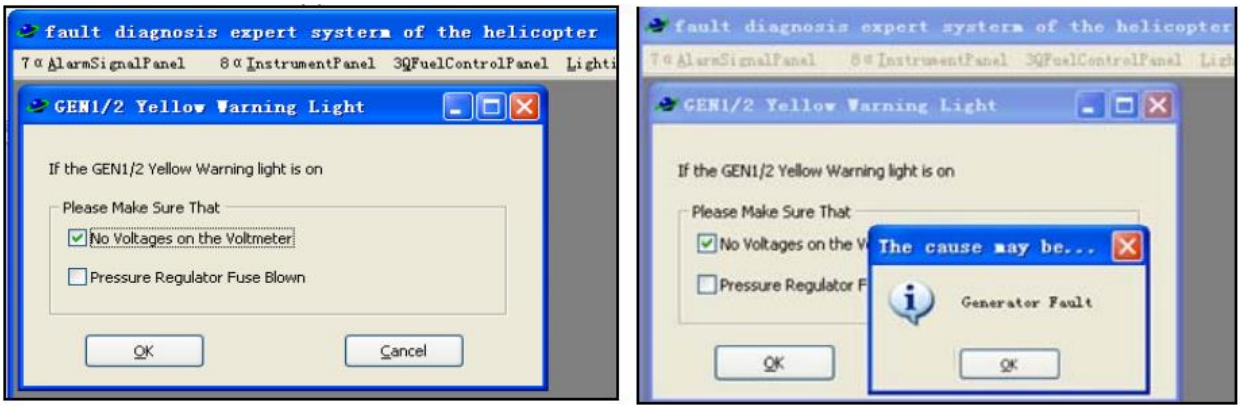


Σχήμα 2 Δομή διαγνωστικού βλαβών έμπειρου συστήματος. [5]

Ένα κομμάτι κώδικα του παραπάνω διαγνωστικού σε Visual Prolog είναι το εξής:

```
clauses
  fault (“generator fault”):-
        symptom(“Generator 1 or 2 (GEN1/2) yellow alert light on”),
        symptom(“no voltages on the voltmeter”),
        not(symptom(“pressure regulator fuse blown”)).
```

Στην συνέχεια βλέπουμε δύο στιγμιότυπα εκτέλεσης του παραπάνω κώδικα.



Σχήμα 3 Στιγμιότυπα εκτέλεσης κώδικα [5]

Το ειδικό σύστημα διάγνωσης βλαβών για το ελικόπτερο αναπτύχθηκε από τη Visual Prolog. έχει τα ακόλουθα χαρακτηριστικά μετά από ορισμένες προσαρμογές και βελτιώσεις:

1. Απλός σχεδιασμός: Ακριβώς όπως τα πλεονεκτήματα της γλώσσας Prolog, ο φόρτος εργασίας που απαιτείται για τη δημιουργία ενός έμπειρου συστήματος είναι πολύ μικρότερος από τη χρήση ορισμένων άλλων παραδοσιακών γλωσσών προγραμματισμού.
2. Υψηλή αξιοπιστία: Η γνώση του έμπειρου συστήματος προέρχεται από το "Εγχειρίδιο Ανάλυσης Βλαβών", το "Εγχειρίδιο Συντήρησης Αερομεταφερόμενου Εξοπλισμού" γραμμένο από το εργοστάσιο ελικοπτέρων και πληροφορίες από πρώτο χέρι, που συλλέγονται από το τεχνικό προσωπικό που έχει μεγάλη εμπειρία στη συντήρηση πεδίου. Οι διαδικασίες και τα αποτελέσματα του συμπεράσματος έχουν αναγνωριστεί από ειδικούς.
3. Μεγάλος όγκος πληροφοριών: Το σύστημα περιέχει περισσότερα από διακόσια διαφορετικά είδη πληροφοριών διάγνωσης βλαβών του ηλεκτρικού συστήματος και του συστήματος οργάνων. Μπορεί να αντικαταστήσει το επαγγελματικό προσωπικό συντήρησης και να παρέχει επαγγελματική καθοδήγηση για τη διάγνωση βλαβών, παρέχοντας επομένως ευκολία στους πιλότους.
4. Εύκολη λειτουργία: Ο χειρισμός του συστήματος είναι εύκολος για τους εργαζομένους, επειδή χρειάζεται μόνο να χρησιμοποιήσουν το ποντίκι. Επομένως είναι ιδιαίτερα κατάλληλο για προσωρινή συντήρηση πεδίου και ξαφνική διάγνωση βλαβών κατά την πτήση [5].

1.3 Εργαλεία

1.3.1 Prolog

Η Prolog είναι μια γλώσσα λογικού προγραμματισμού γενικής χρήσης που κυρίως χρησιμοποιείται στον τομέα της τεχνητής νοημοσύνης. Δημιουργήθηκε στις αρχές του 1970 από τους Ρόμπερτ Κοβάλσκι και Alain Colmeraue στη Μασσαλία της Γαλλίας [6].

Οι ρίζες της γλώσσας προγραμματισμού Prolog πηγαίνουν πίσω στη κατηγορηματική λογική πρώτης τάξεως, η οποία μπορεί να οριστεί ως τυπική λογική. Η γλώσσα έχει δηλωτικό χαρακτήρα, γεγονός που την καθιστά εντελώς διαφορετική από

τις άλλες γλώσσες προγραμματισμού που υπάρχουν στον τομέα της πληροφορικής. Σε αυτή τη γλώσσα οι σχέσεις, οι κανόνες και τα γεγονότα χρησιμοποιούνται για να εκφράσουν τη λογική ενός προγράμματος. Κάνοντας χρήση των συστατικών αυτών στοιχείων, οι χρήστες μπορούν να εισάγουν τα δικά τους ερωτήματα και να λαμβάνουν τις επιθυμητές απαντήσεις μέσω του προγράμματος που έχει υλοποιηθεί [7] . Ένας υπολογισμός ξεκινά εκτελώντας ένα ερώτημα πάνω σε αυτές τις σχέσεις. Η Prolog ήταν μια από τις πρώτες λογικές γλώσσες προγραμματισμού και παραμένει η πιο δημοφιλής τέτοια γλώσσα σήμερα, με πολλές διαθέσιμες δωρεάν και εμπορικές εφαρμογές. Η γλώσσα έχει χρησιμοποιηθεί για την απόδειξη θεωρημάτων, έμπειρα συστήματα, επαναγραφή όρων, συστήματα τύπου και αυτοματοποιημένο σχεδιασμό, καθώς και για το αρχικό πεδίο χρήσης της, την επεξεργασία φυσικής γλώσσας . Τα σύγχρονα περιβάλλοντα Prolog υποστηρίζουν τη δημιουργία γραφικών διεπαφών χρήστη όπως το περιβάλλον visual-prolog , καθώς και διοικητικών και δικτυωμένων εφαρμογών. Η Prolog είναι κατάλληλη για συγκεκριμένες εργασίες που επωφελούνται από λογικά ερωτήματα που βασίζονται σε κανόνες, όπως αναζήτηση βάσεων δεδομένων, συστήματα φωνητικού ελέγχου και συμπλήρωση προτύπων [8] .

Η γλώσσα prolog αποτελείται από ένα μεγάλο σύνολο δομών δεδομένων από την άποψη της ανθρώπινης λογικής και της γλώσσας. Διαθέτει επίσης μια πολύ ισχυρή σημειογραφία παρέχοντας στον τελικό χρήστη την δυνατότητα κωδικοποίησης εφαρμογών. Τέλος αξίζει να σημειωθούν ορισμένα από τα πεδία εφαρμογών στα οποία η Prolog έχει χρησιμοποιηθεί με μεγάλη επιτυχία:

Ευφυή Συστήματα:

Αυτά μπορούν να οριστούν ως προγράμματα τα οποία επιδεικνύουν λογική, εμπειρική μάθηση, και ικανότητες λήψης λογικών αποφάσεων χωρίς την μεσολάβηση του ανθρώπου. Στόχος είναι η κατανόηση και η αναπαραγωγή κατά το δυνατό του τρόπου που οι άνθρωποι, τα ζώα και άλλοι βιολογικοί οργανισμοί εξελίσσονται και αναπτύσσουν ικανότητες επίλυσης δύσκολων προβλημάτων [9].

Σημασιολογικός Ιστός:

Τα τελευταία χρόνια έχουν γίνει προσπάθειες να συνδυαστούν οι τεχνικές του λογικού προγραμματισμού με τις τεχνολογίες του σημασιολογικού ιστού, με αποτέλεσμα να δημιουργηθεί μία καινούργια μορφή προγραμματισμού με την επωνυμία Περιγραφικός Λογικός Προγραμματισμός (Description Logic Programming). Η ανάπτυξη του

βασίστηκε στην διαπίστωση ότι ο σημασιολογικός ιστός μπορεί να επωφεληθεί μέσω της χρήσης λογικών κανόνων που αναφέρονται σε κάποιο συγκεκριμένο πεδίο. Ο περιγραφικός λογικός προγραμματισμός παρέχει ένα βαθμό εκφραστικότητας σημαντικά μεγαλύτερο από αυτόν της περιγραφικής λογικής του RDF-σχήματος [10] .

Έμπειρα Συστήματα:

Αυτά είναι εξ ορισμού συστήματα που προσπαθούν να επιδείξουν ικανότητες στη λήψη αποφάσεων παρόμοιες με αυτές ενός ειδήμονα (εμπειρογνώμονα) σε ένα γνωστικό τομέα. Πιο απλά, ένα Έμπειρο Σύστημα (ΕΣ) είναι ένα διαλογικό μηχανογραφικό εργαλείο σχεδιασμένο να λύνει δύσκολα προβλήματα λήψης αποφάσεων τα οποία βασίζονται σε γνώση που έχει συγκεντρωθεί από τους ειδήμονες. Υπό αυτό το πρίσμα, ένα ΕΣ αναμένεται να ενεργεί σε όλα του τα σημεία παρόμοια με τον τρόπο με τον οποίο θα ενεργούσε ένας ειδήμονας [11].

Συστήματα επεξεργασίας φυσικής γλώσσας:

Επιτρέπουν στα προγράμματα να κατανοούν, να αναλύουν και να δίνουν απάντηση στις δηλώσεις που υπάρχουν στη φυσική γλώσσα χωρίς να βασίζονται σε μενού επιλογής και επιλεγμένες λέξεις κλειδιά.

Σχεσιακά συστήματα βάσεων δεδομένων:

Μπορούν να οριστούν ως προγράμματα που χρησιμοποιούνται για την δημιουργία, ενημέρωση, διαχείριση και αλληλεπίδρασή με σχεσιακές βάσεις δεδομένων [7]. Όπως, η γλώσσα Datalog, που αποτελεί ένα υποσύνολο της Prolog με απόλυτη δηλωτική σημασιολογία, έχει χρησιμοποιηθεί και ως γλώσσα ερωτήσεων σε επαγωγικές βάσεις δεδομένων [10].

Ο Alain Colmerauer μαζί με τον Philippe Roussel συνέχισαν να εργάζονται και κατάφεραν να αναπτύξουν το πρώτο σύστημα βασισμένο στην Prolog το 1972. Η ελκυστικότητα που γνώρισε η Prolog στον χώρο της πληροφορικής ήταν μεγάλη, καθώς ήταν η πρώτη γλώσσα λογικού προγραμματισμού. Η γλώσσα εξακολουθεί να κατέχει σημαντική θέση μεταξύ των γλωσσών λογικού προγραμματισμού. Επιπλέον έχει πληθώρα εμπορικών και δωρεάν εφαρμογών. Η βασική χρήση της γλώσσας προγραμματισμού Prolog περιλαμβάνει τα εξής: Απόδειξη θεωρημάτων, Επεξεργασία φυσικής γλώσσας (που ήταν το πεδίο χρήσης της Prolog κατά την περίοδο ανάπτυξή της) και Συστήματα εξαγωγής συμπερασμάτων. Τα σημερινά περιβάλλοντα Prolog επεκτείνουν την υποστήριξη για την δημιουργία γραφικών διεπαφών χρήστη, όπως και

δικτυακές και διοικητικές εφαρμογές. Η Prolog ταιριάζει καλύτερα στις εργασίες που μπορούν να αξιοποιήσουν τις λογικές ερωτήσεις που βασίζονται σε κανόνες. Στην κατηγορία αυτών των εργασιών περιλαμβάνονται οι αναζητήσεις σε βάσεις δεδομένων, τα συστήματα φωνητικού ελέγχου κ.α. [7].

Γεγονότα

Ένα γεγονός αναπαριστά μία ιδιότητα ενός αντικειμένου ή μία σχέση μεταξύ δύο ή περισσότερων αντικειμένων ή γενικότερα ένα αντικείμενο [11]. Τα γεγονότα μπορούν να οριστούν ως:

- **Άτομα :** Σταθερές που αποτελούνται από αλφαριθμητικούς χαρακτήρες [12]**:**
  - cat.
  - person.
  - car.
  - hotel.
- **Ατομικοί τύποι ή Σύνθετοι όροι :** Αρχίζουν με ένα άτομο και ακολουθούνται από μια ακολουθία όρων που διαχωρίζονται με κόμματα και περικλείονται από παρενθέσεις [12]:
  - father(john, peter).
  - odd(3).
  - triangle(vertex(10,12),vertex(8,20),vertex(34,23))
  - computer(keyboard(120),monitor(color(tft)).

Όπως βλέπουμε και στα παραπάνω παραδείγματα κάθε γεγονός τελειώνει με μία τελεία. Η κεφαλή του ατομικού τύπου (για παράδειγμα το odd, triangle) ονομάζεται κατηγόρημα ενώ όρισμα ενός κατηγορήματος ή ενός σύνθετου όρου μπορεί να είναι είτε ένα άτομο, είτε ένας αριθμός, είτε μία μεταβλητή [11]

Μεταβλητές

Στην Prolog, το όρισμα ενός κατηγορήματος ή σύνθετου όρου μπορεί εκτός από άτομο ή σύνθετος όρος να είναι και μεταβλητή. Οι μεταβλητές ξεκινούν με κεφαλαίο γράμμα και σε αντίθεση με άλλες γλώσσες προγραμματισμού οι μεταβλητές στη Prolog δεν έχουν τύπο. Μία μεταβλητή μπορεί να ταυτιστεί με οποιοδήποτε άτομο,

κατηγόρημα, σύνθετο όρο ή αριθμό. Η χρήση μεταβλητών στην Prolog είναι ιδιαίτερα σημαντική και εντοπίζεται κυρίως στους κανόνες των προγραμμάτων και στα ερωτήματα(queries) που θέτουν οι χρήστες [11].

Κανόνες

Οι κανόνες (rules) αποτελούν μία μέθοδο αναπαράστασης διαδικαστικής γνώσης που αντιστοιχεί σε έγκυρους συλλογισμούς και έχει τη μορφή IF-THEN. Η δήλωση ή το σύνολο των δηλώσεων μετά τον όρο IF αντιπροσωπεύει κάποιο παρατηρήσιμο πρότυπο . Η δήλωση ή το σύνολο των δηλώσεων μετά τον όρο THEN αντιπροσωπεύει κάποιο εξαγώγιμο συμπέρασμα ή κάποια εκτελεστέα ενέργεια . Επομένως, ένα κανόνας:

- Προσδιορίζει ένα πρότυπο και εξάγει συμπεράσματα σχετικά με το τι αυτό σημαίνει ή
- Προσδιορίζει ένα πρότυπο και συμβουλεύει τι πρέπει να γίνει γι' αυτό ή
- Προσδιορίζει ένα μοτίβο και πραγματοποιεί τις κατάλληλες ενέργειες.

Οι δηλώσεις μετά το IF ονομάζονται συνθήκες, ισχυρισμοί, προϋποθέσεις, πρόγονοι, ενώ αυτά που ακολουθούν μετά το THEN μπορούν να ονομάζονται συμπεράσματα.

IF A (ισχυρισμοί)

and B (πρόγονοι,προηγούμενα)

and ... (προϋποθέσεις)

THEN X (συμπεράσματα)

and Y (συνεπαγόμενα)

and … (ενέργειες)

Αποκοπή

Το πλέον ισχυρό ενσωματωμένο κατηγόρημα ελέγχου της οπισθοδρόμησης στην Prolog, είναι το κατηγόρημα της αποκοπής (cut). Η λειτουργία του είναι να μειώνει το δένδρο αναζήτησης κλαδεύοντας δυναμικά, δηλαδή κατά την ώρα της εκτέλεσης, κλαδιά (μονοπάτια αναζήτησης) τα οποία γνωρίζουμε ότι δεν οδηγούν σε λύση. Το

κατηγόρημα συμβολίζεται ένα θαυμαστικό (!) και εμφανίζεται σαν κανονικός υποστόχος στο σώμα ενός κανόνα, ο οποίος πετυχαίνει πάντα. Η επίδραση της αποκοπής στο μηχανισμό εκτέλεσης είναι να “παγώνει” (freeze) όσες επιλογές έχουν γίνει πριν την εκτέλεση της, για το κατηγόρημα μέσα στο οποίο εμφανίζεται, δηλαδή υποχρεώνει το μηχανισμό οπισθοδρόμησης να αγνοήσει τις όποιες εναλλακτικές λύσεις υπήρχαν για το κατηγόρημα και βρίσκονται αριστερά της αποκοπής [10].

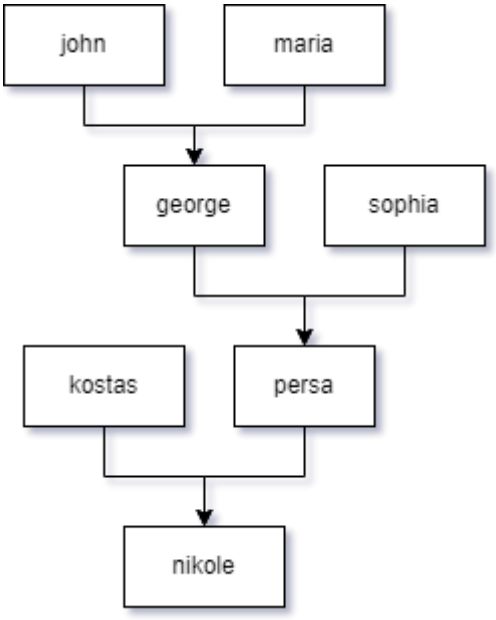
Όταν ο μηχανισμός αναζήτησης της Prolog συναντήσει μία αποκοπή αγνοούνται όλες οι προτάσεις που ανήκουν στη διαδικασία και έπονται της πρότασης που περιέχει την αποκοπή και αγνοούνται όλες οι εναλλακτικές λύσεις των ατομικών τύπων που βρίσκονται πριν από την αποκοπή μέσα στο σώμα του κανόνα που εμφανίζεται [13].

Παράδειγμα προγράμματος σε Prolog

Όπως αναφέραμε προηγουμένως κάθε βασικό πρόγραμμα σε Prolog συντάσσεται με την χρήση δύο βασικών συστατικών στοιχείων, τα γεγονότα και τους κανόνες. Έστω ένα πρόγραμμα που έχουμε να κατασκευάσου θα αναπαριστά το οικογενειακό δέντρο μίας οικογένειας.

Στο σύνολο των γεγονότων του προγράμματος μπορούμε να συμπεριλάβουμε το φύλο του κάθε ατόμου (π.χ. ο george είναι άντρας.) καθώς και την γονική σχέση μεταξύ δύο ατόμων (π.χ. Ο kostas είναι πατέρας της nikole.)

```
father(john, george).
father(george, persa).
father(kostas, nikole).
mother(maria, george).
mother(persa, nikole).
mother(sophia, persa).
woman(persa).
woman(maria).
woman(nikole).
woman(sophia).
man(george).
man(john).
man(kostas).
```



Εικόνα 1 Διάγραμμα παραδείγματος οικογένειας.

Εφόσον έχουμε καταγράψει τα άτομα και σύνθετους όρους του προγράμματος, μπορούμε να επεκτείνουμε την λειτουργικότητα του προσθέτοντας κανόνες όπως για παράδειγμα έναν κανόνα που να εντοπίζει εάν ο Α είναι γονέας του Β. Σύμφωνα με τους παρακάτω κανόνες ο Α είναι γονέας του Β εάν ο Α είναι πατέρας ή μητέρα του Β.

parent(A , B) :- father(A , B).

parent(A , B) :- mother(A , B).

1.3.2 SWI-Prolog

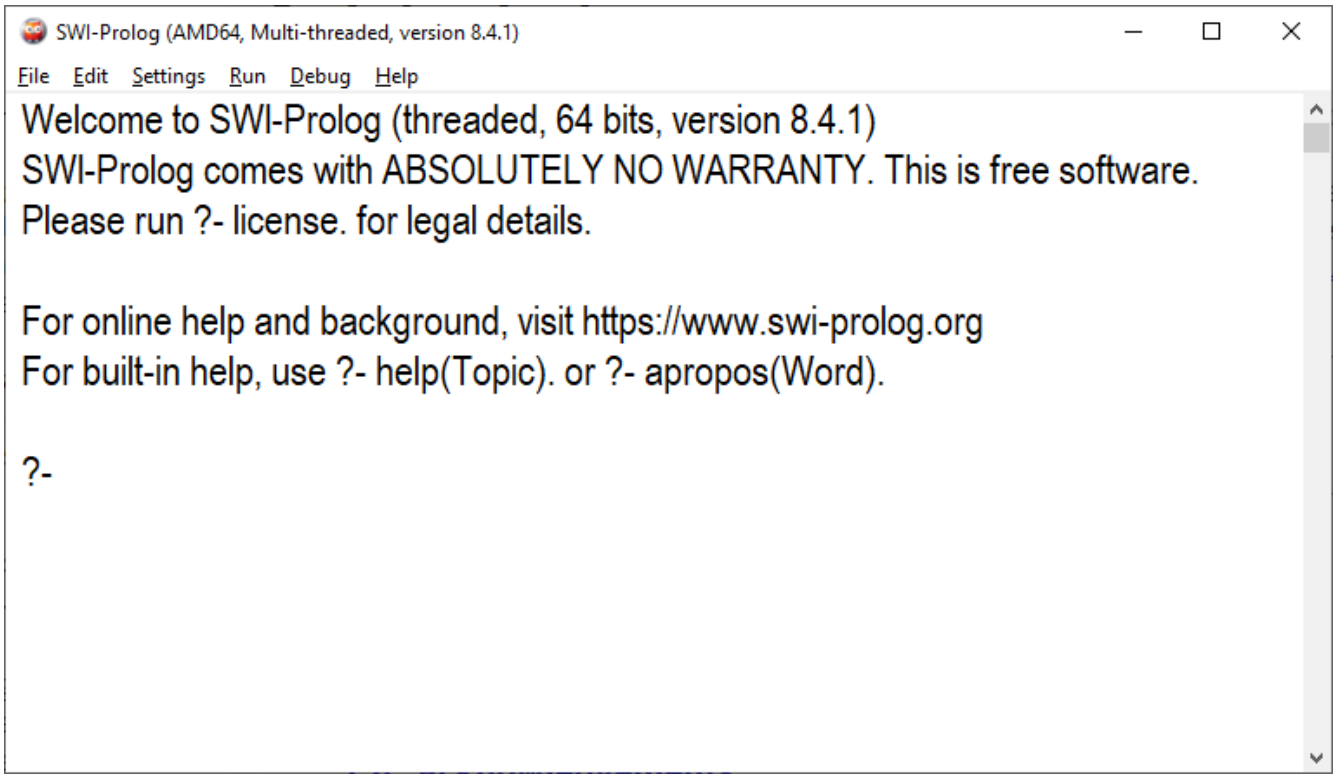
Το SWI-Prolog είναι μια ευέλικτη εφαρμογή της γλώσσας Prolog. Αν και το SWI-Prolog κέρδισε τη δημοτικότητά του κυρίως στην εκπαίδευση, η ανάπτυξή του οφείλεται κυρίως στις ανάγκες ανάπτυξης εφαρμογών. Αυτό διευκολύνεται από μια πλούσια διεπαφή με άλλα στοιχεία πληροφορικής υποστηρίζοντας πολλούς τύπους εγγράφων και πρωτόκολλα (δικτύου), καθώς και μια ολοκληρωμένη διεπαφή χαμηλού επιπέδου με το C που αποτελεί τη βάση για διεπαφές υψηλού επιπέδου σε C++, Java (ομαδοποιημένα), C#, Python κ.λπ. Οι επεκτάσεις τύπου δεδομένων, όπως οι υπαγορεύσεις και οι συμβολοσειρές, καθώς και η πλήρης υποστήριξη για το Unicode και τους μη δεσμευμένους ακέραιους απλοποιούν την ομαλή ανταλλαγή δεδομένων με άλλα στοιχεία.

Παράλληλα η SWI-Prolog στοχεύει στην επεκτασιμότητα. Η ισχυρή υποστήριξή της για πολυνηματικό προγραμματισμό (multi-threading) εκμεταλλεύεται αποτελεσματικά το υλικό πολλαπλών πυρήνων και απλοποιεί την ενσωμάτωση της σε ταυτόχρονες εφαρμογές. Το Just In Time Indexing (JITI) παρέχει διαφανή και αποτελεσματική υποστήριξη για κατηγορήματα που περιλαμβάνουν εκατομμύρια όρους. Επιπλέον η SWI-Prolog ενοποιεί πολλές επεκτάσεις της βασικής γλώσσας που έχουν αναπτυχθεί στην κοινότητα της Prolog, όπως καταλογογράφηση (tabling), περιορισμοί (constraints), καθολικές (global) μεταβλητές, καταστροφική ανάθεση (destructive assignment) και interactors.

Τέλος η SWI-Prolog προσφέρει μια ποικιλία εργαλείων ανάπτυξης, τα περισσότερα από τα οποία μπορούν να συνδυαστούν κατά βούληση. Το εγγενές σύστημα παρέχει έναν editor γραμμένο σε Prolog που αποτελεί στενό κλώνο του Emacs. Επίσης παρέχει σημασιολογική επισήμανση βασισμένη σε ανάλυση που γίνεται σε πραγματικό

χρόνο από το ίδιο το σύστημα της Prolog. Τα συμπληρωματικά εργαλεία περιλαμβάνουν ένα γραφικό debugger, profiler, και ένα cross-referencer. Εναλλακτικά, υπάρχει και η επιλογή για το GNU-Emacs, το Eclipse plugin που ονομάζεται PDT, και ένα VSC plugin [14] .

Στο πλαίσιο της πτυχιακής εργασίας η SWI-Prolog χρησιμοποιήθηκε για την δημιουργία μιας διαγνωστικής εφαρμογής βλαβών για κινητά τηλέφωνα.

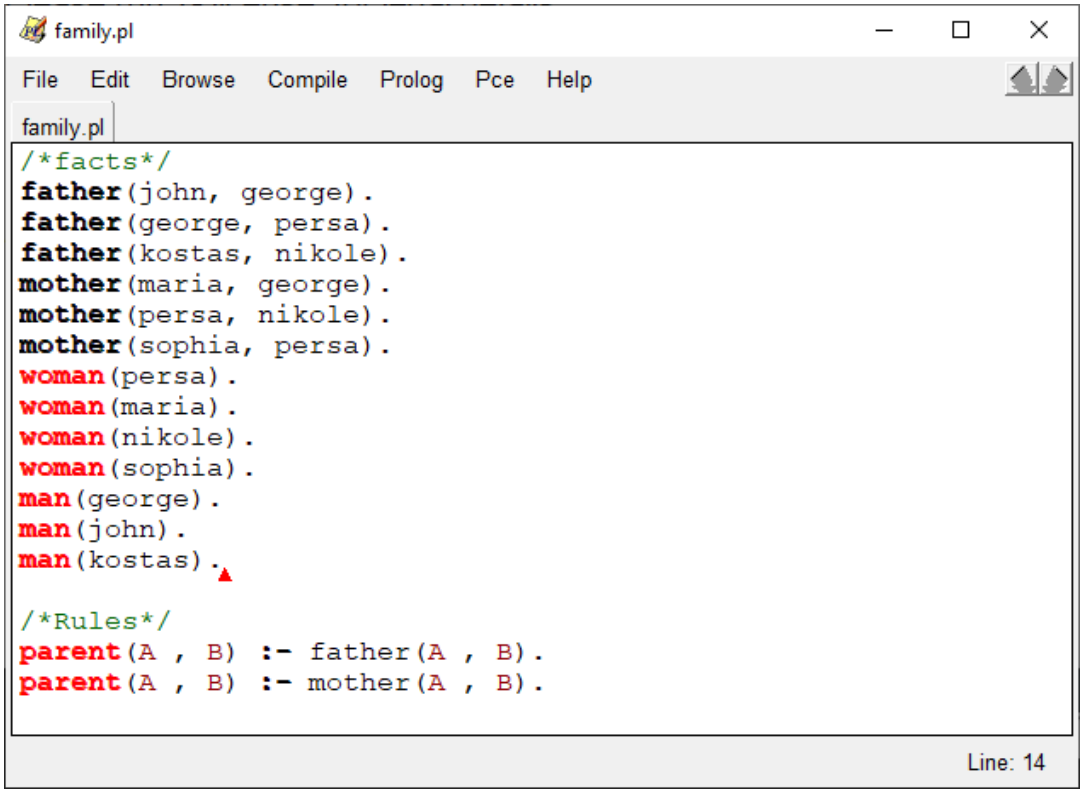


Εικόνα 2 Περιβάλλον κονσόλας του SWI – Prolog.

Πηγή: Εφαρμογή SWI-Prolog

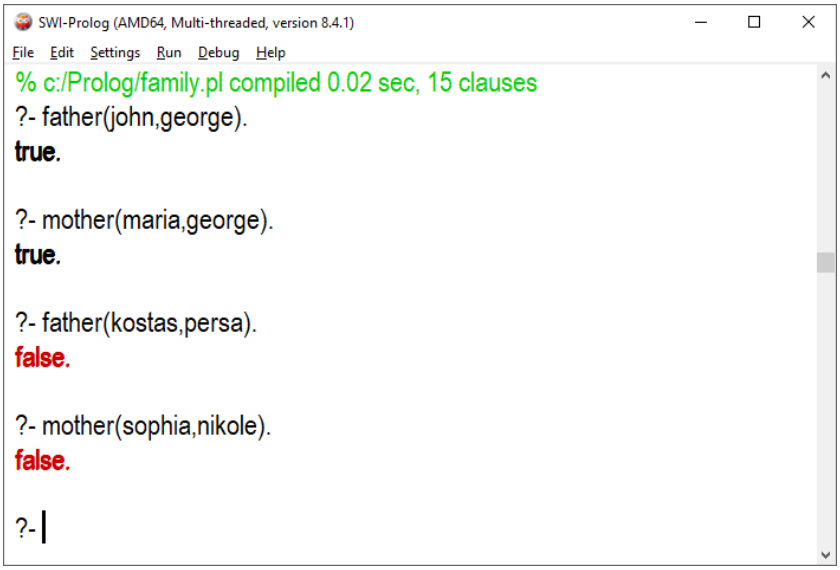
Εκτέλεση παραδείγματος οικογενειακού δέντρου

Χρησιμοποιώντας το σημειωματάριο (editor) του SWI-Prolog δημιουργήσαμε ένα αρχείο με κατάληξη .pl όπου συμπληρώσαμε τους όρους και τους κανόνες του προηγούμενου παραδείγματος με σκοπό την εκτέλεση ερωτημάτων.



Εικόνα 3 Αρχείο family.pl στον SWI-Prolog Editor

Στην συνέχεια φορτώνουμε το πρόγραμμα μας κάνοντας consult το αρχείο του παραδείγματος family.pl που φτιάξαμε και έπειτα μπορούμε να εκτελέσουμε κάποια ερωτήματα (queries) στην γνωσιακή βάση του προγράμματος με σκοπό την εμφάνιση της συγγένειας της οικογένειας.



Εικόνα 4 Στιγμιότυπο ερωτημάτων

Όπως βλέπουμε στο παραπάνω στιγμιότυπο εικόνα 4 έχουμε τέσσερα ερωτήματα όπου η prolog επιστρέφει true εάν βρει αντιστοιχία με τα γεγονότα που έχουν δοθεί ειδάλλως επιστρέφει false.

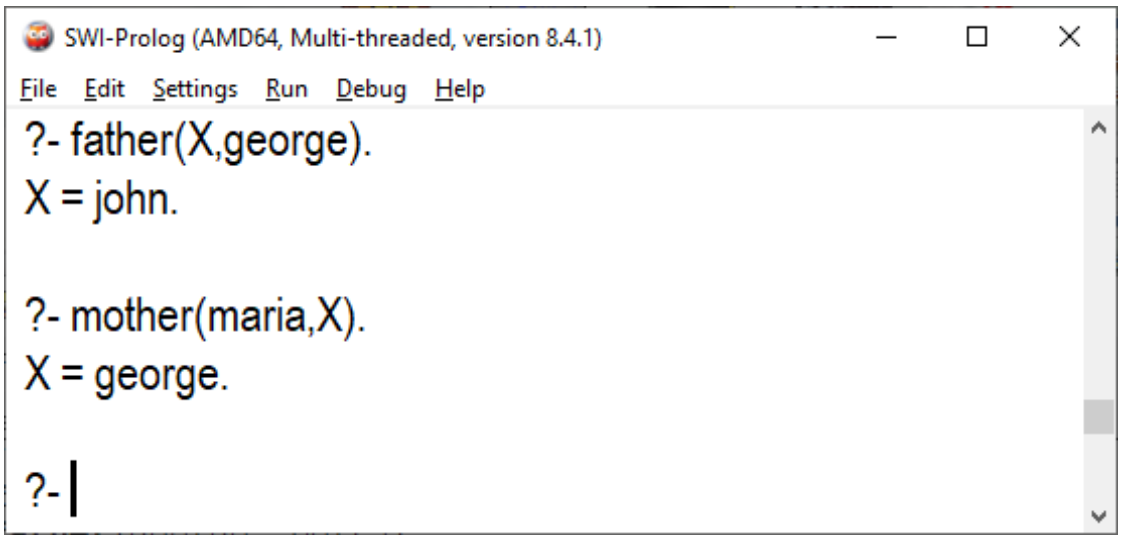
father(john , george). Εξήγηση : Είναι ο john πατέρας του george? Σύμφωνα με την γνωσιακή μας βάση η prolog μας επιστρέφει true.

mother(maria , george). Εξήγηση : Είναι η maria μητέρα του george? Σύμφωνα με την γνωσιακή μας βάση η prolog μας επιστρέφει true.

father(kostas , persa). Εξήγηση : Είναι ο kostas πατέρας της persa? Σύμφωνα με την γνωσιακή μας βάση η prolog μας επιστρέφει false.

mother(sophia , nikole). Εξήγηση : Είναι η sophia μητέρα της nikole? Σύμφωνα με την γνωσιακή μας βάση η prolog μας επιστρέφει false.

Μπορούμε να κάνουμε και ερωτήματα με χρήση μεταβλητής όπως για παράδειγμα βλέπουμε στην εικόνα 5:

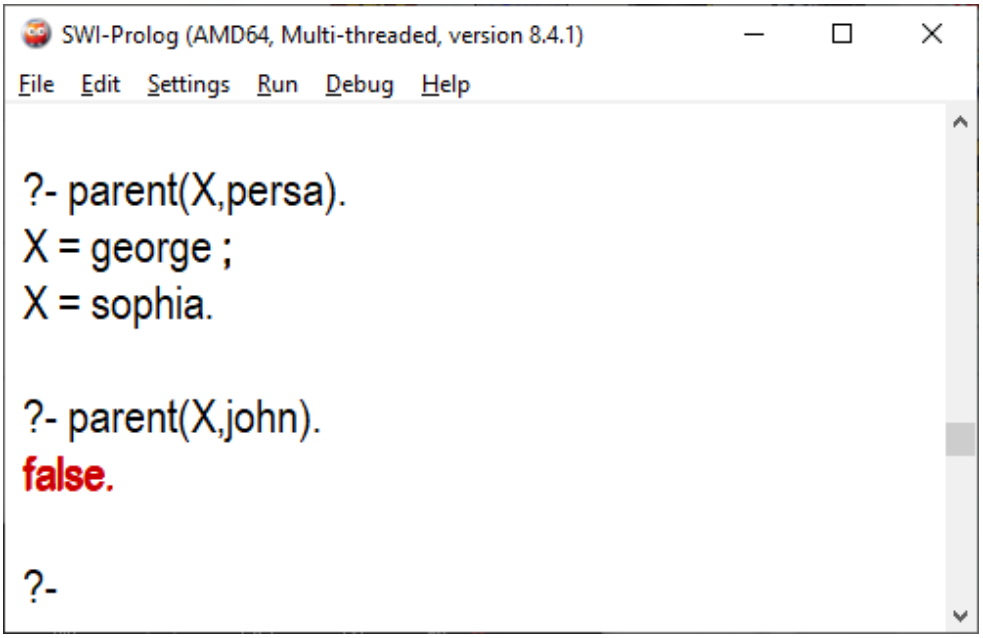


Εικόνα 5 Στιγμιότυπο ερωτημάτων με τη χρήση μεταβλητής

father(X , george). Εξήγηση : ποιος είναι ο πατέρας του george? Σύμφωνα με την γνωσιακή μας βάση η prolog κάνει αντιστοιχία μόνο με ένα γεγονός και μας επιστρέφει X=john.

mother(maria , X). Εξήγηση : ποια είναι τα παιδιά της maria? Σύμφωνα με την γνωσιακή μας βάση η prolog κάνει αντιστοιχία μόνο με ένα γεγονός και μας επιστρέφει X=george.

Στη συνέχεια θα εκτελέσουμε ερωτήματα με τη χρήση του κανόνα parent όπως θα δούμε στην εικόνα 6.



Εικόνα 6 Στιγμιότυπο ερωτημάτων με τη χρήση του κανόνα parent

parent(X,persa). Εξήγηση : ποιος είναι ο γονέας της persa? Σύμφωνα με την γνωσιακή μας βάση η prolog κάνει αντιστοιχία με δυο γεγονότα X=george και X=sophia.

parent(X,john). Εξήγηση : ποιος είναι ο γονέας του john? Σύμφωνα με την γνωσιακή μας βάση η prolog μας επιστρέφει ως απάντηση false αφού δεν υπήρχε καμία αντιστοιχία γονέων του john.

1.3.3 Visual Prolog

Η Visual Prolog είναι μια γλώσσα προγραμματισμού πολλαπλών παραδειγμάτων που βασίζεται στη λογική γλώσσα Prolog. Ο στόχος του Visual Prolog είναι να διευκολύνει τις προγραμματικές λύσεις σύνθετων προβλημάτων με έμφαση στη γνώση.

Η Visual Prolog είναι μια ισχυρή και ασφαλής γλώσσα προγραμματισμού υψηλού επιπέδου που συνδυάζει τα καλύτερα χαρακτηριστικά των λογικών, λειτουργικών και αντικειμενοστραφών παραδειγμάτων προγραμματισμού με συνεπή και κομψό τρόπο.

Η Visual Prolog μπορεί να δημιουργήσει εφαρμογές για τις πλατφόρμες Microsoft Windows 32/64.

Υποστηρίζει προηγμένες λύσεις πελάτη-διακομιστή και τριών επιπέδων. Το Visual Prolog είναι ιδιαίτερα κατάλληλο για την αντιμετώπιση σύνθετων γνώσεων. Χρησιμοποιώντας το ισχυρό σύστημα αντικειμένων, μπορείτε να αρχιτεκτονήσετε την εφαρμογή πολύ άκαμπτα και ταυτόχρονα να επωφεληθείτε από την πολύ χαλαρή σύζευξη. Αυτό επιτρέπει τη μειώσει του κόστους ανάπτυξης και ακόμη περισσότερο το κόστος συντήρησης [15].

Διαφορές μεταξύ της Visual Prolog και της παραδοσιακής Prolog

Οι διαφορές μεταξύ της παραδοσιακής Prolog και της Visual Prolog μπορούν να χωριστούν ευρέως σε αυτές τις κατηγορίες:

- **Διαφορές στη δομή του προγράμματος:** Υπάρχουν ευδιάκριτες, αλλά εύκολα κατανοητές διαφορές μεταξύ της δομής που χρησιμοποιείται στην παραδοσιακή Prolog και αυτής που χρησιμοποιείται στο Visual Prolog. Ουσιαστικά περιλαμβάνει την κατανόηση του τρόπου επισήμανσης των δηλώσεων από τους ορισμούς και την ένδειξη του κύριου στόχου που πρέπει να επιδιώξει το πρόγραμμα χρησιμοποιώντας συγκεκριμένες λέξεις-κλειδιά.
- **Δομή αρχείων (File Considerations):** Η Visual Prolog παρέχει διάφορες ευκολίες για την οργάνωση της δομής του προγράμματος σε διαφορετικά είδη αρχείων. Πολλές φορές, μπορεί να γίνει δύσχρηστη η τοποθέτηση όλων των μερών του προγράμματος στο ίδιο αρχείο. Μπορεί ακόμη να κάνει το πρόγραμμα μη αναγνώσιμο και μερικές φορές λανθασμένο. Η Visual Prolog έχει τη δυνατότητα να διαιρεί τον κώδικα του προγράμματος σε ξεχωριστά αρχεία χρησιμοποιώντας το IDE (Integrated Development Environment) και είναι δυνατή η εγγραφή καθαρών κομματιών κώδικα σε ξεχωριστά αρχεία χρησιμοποιώντας αυτό το IDE. Όταν γίνεται με αυτόν τον τρόπο, τα πράγματα που πρόκειται να χρησιμοποιηθούν συνήθως είναι προσβάσημα σε όλα τα αρχεία. Για παράδειγμα εάν υπάρχει ένας τομέας που πρόκειται να χρησιμοποιηθεί σε πολλά αρχεία, τότε η δήλωση αυτού του

τομέα γίνεται σε ξεχωριστό αρχείο και στη συνέχεια γίνεται πρόσβαση σε αυτό το αρχείο από άλλα αρχεία.

- **Ζητήματα πρόσβασης πεδίου (Scope access issues) :** Ένα πρόγραμμα Visual Prolog μπορεί να επιλέξει τη λειτουργικότητα που έχει αναπτυχθεί σε άλλες λειτουργικές μονάδες χρησιμοποιώντας μια έννοια που ονομάζεται αναγνώριση εύρους (scope identification). Το Visual Prolog διαιρεί τον συνολικό κώδικα του προγράμματος σε ξεχωριστά μέρη, με κάθε τμήμα να ορίζει μία κλάση. Στις αντικειμενοστραφείς γλώσσες, μια κλάση είναι ένα πακέτο κώδικα και τα σχετικά δεδομένα που συγκεντρώνονται.
- **Αντικειμενοστραφής γλώσσα:** Ένα πρόγραμμα Visual Prolog μπορεί να γραφτεί ως αντικειμενοστραφή πρόγραμμα, χρησιμοποιώντας κλασικά αντικειμενοστραφή χαρακτηριστικά. Η τρέχουσα έκδοση του Visual Prolog είναι μια γλώσσα έντονα αντικειμενοστραφή. Ολόκληρος ο κώδικας που έχει αναπτυχθεί για ένα πρόγραμμα τοποθετείται σε κατάλληλες κλάσεις όπως και απαιτείται, αυτό συμβαίνει από προεπιλογή.

Λέξεις κλειδιά

Ένα πρόγραμμα Visual Prolog αποτελείται από κώδικα Prolog ο οποίος είναι σημειωμένος σε διαφορετικές ενότητες με κατάλληλες λέξεις-κλειδιά που ενημερώνουν τον μεταγλωττιστή για τον κώδικα που πρέπει να δημιουργήσει. Για παράδειγμα, υπάρχουν λέξεις-κλειδιά που διαφοροποιούν τις δηλώσεις από τους ορισμούς των τομέων και των τομέων. Συνήθως, πριν από κάθε ενότητα προηγείται μια λέξη-κλειδί. Συνήθως δεν υπάρχει λέξη-κλειδί που να σημαίνει το τέλος μιας συγκεκριμένης ενότητας. Η παρουσία μιας άλλης λέξης-κλειδιού υποδεικνύει το τέλος της προηγούμενης ενότητας και την έναρξη της επόμενης.

Η εξαίρεση σε αυτόν τον κανόνα, είναι οι λέξεις-κλειδιά "implement" και "end implement". Ο κώδικας που περιέχεται μεταξύ αυτών των δύο λέξεων-κλειδιών υποδεικνύει τον κωδικό που θα χρησιμοποιηθεί για μια συγκεκριμένη κλάση.

- **implement** και **end implement:** Αυτή η λέξη κλειδί είναι η μόνη που υπάρχει ως ζευγάρι. Η Visual Prolog αντιμετωπίζει τον κώδικα που γράφεται μεταξύ αυτών των δύο λέξεων-κλειδιών ως τον κώδικα που ανήκει σε μία κλάση. Το όνομα της κλάσης πρέπει να δίνεται μετά τη λέξη-κλειδί υλοποίησης.
- **open:** Αυτή η λέξη-κλειδί χρησιμοποιείται για την επέκταση της ορατότητας του εύρους της τάξης. Πρέπει να χρησιμοποιείται αμέσως μετά τη λέξη-κλειδί υλοποίησης.

- **constants:** Αυτή η λέξη-κλειδί χρησιμοποιείται για την επισήμανση μιας ενότητας του κώδικα που ορίζει ορισμένες τιμές που χρησιμοποιούνται συνήθως στον κώδικα του προγράμματος.
- **domains:** Αυτή η λέξη-κλειδί χρησιμοποιείται για την επισήμανση μιας ενότητας που δηλώνει τους τομείς που θα χρησιμοποιηθούν στον κώδικα. Υπάρχουν πολλές παραλλαγές για τη σύνταξη τέτοιων δηλώσει τομέα.
- **class facts:** Αυτή η λέξη-κλειδί προσδιορίζει μια ενότητα, η οποία δηλώνει τα γεγονότα που θα χρησιμοποιηθούν αργότερα στον κώδικα του προγράμματος. Κάθε γεγονός δηλώνεται με το όνομα που χρησιμοποιείται για να υποδηλώσει το γεγονός και τα ορίσματα που χρησιμοποιούνται για τα αντίστοιχα γεγονότα μαζί με τους τομείς στους οποίους ανήκουν αυτά τα ορίσματα.
- **class predicates:** Αυτή η ενότητα περιέχει τις δηλώσεις των κατηγορημάτων που θα οριστούν αργότερα στην ενότητα ρήτρες του κώδικα. Για άλλη μια φορά, τα ονόματα που θα χρησιμοποιηθούν για αυτά τα κατηγορήματα μαζί με τα ορίσματα και τους τομείς στους οποίους ανήκουν τα ορίσματα, θα υποδεικνύονται σε αυτήν την ενότητα.
- **clauses:** Από όλες τις ενότητες που υπάρχουν σε έναν κώδικα Visual Prolog, αυτή η ενότητα είναι αυτή που μιμείται πολύ ένα παραδοσιακό πρόγραμμα Prolog. Περιέχει τους πραγματικούς ορισμούς των προηγουμένως δηλωμένων κατηγορημάτων. Και θα διαπιστώσετε ότι τα κατηγορήματα που χρησιμοποιούνται εδώ θα ακολουθούν τη σύνταξη όπως δηλώνεται στην ενότητα κατηγορήματα κλάσης.
- **goal:** Αυτή η ενότητα ορίζει το κύριο σημείο εισόδου σε ένα πρόγραμμα Visual Prolog. Στην παραδοσιακή Prolog, όποτε ορίζεται ένα κατηγόρημα στον κώδικα, η μηχανή Prolog μπορεί να λάβει εντολή να ξεκινήσει την εκτέλεση του κώδικα από αυτό το κατηγόρημα και μετά. Ωστόσο, αυτό δεν συμβαίνει με την Visual Prolog. Ως μεταγλωττιστής έχει την ευθύνη να παράγει αποτελεσματικά εκτελούμενο κώδικα για το πρόγραμμα που γράφετε. Αυτός ο κώδικας δεν θα εκτελείται πραγματικά την ίδια στιγμή που ο μεταγλωττιστής κάνει τη δουλειά του. Ως εκ τούτου, ο μεταγλωττιστής πρέπει να γνωρίζει εκ των προτέρων το ακριβές κατηγόρημα από το οποίο θα ξεκινήσει η εκτέλεση του κώδικα, έτσι ώστε αργότερα, όταν το πρόγραμμα καλείται να εκτελεστεί, να το κάνει από το σωστό σημείο εκκίνησης. Το μεταγλωττισμένο πρόγραμμα μπορεί να εκτελεστεί ανεξάρτητα χωρίς τον μεταγλωττιστή Visual Prolog ή το ίδιο το IDE. Για να γίνει αυτό, υπάρχει μια ειδική ενότητα που υποδεικνύεται με τη λέξη-κλειδί goal. Σκεφτείτε το ως ένα ειδικό κατηγόρημα που θα γράφατε χωρίς επιχειρήματα. Αυτό το κατηγόρημα είναι αυτό από το οποίο θα ξεκινήσει ολόκληρη η εκτέλεση του προγράμματος.

Συμπέρασμα

Τα προγράμματα που είναι γραμμένα στη Visual Prolog είναι συχνά πολύ παρόμοια με αυτά που γράφτηκαν στην παραδοσιακή Prolog. Υπάρχουν πολλές λέξεις-κλειδιά που χρησιμοποιούνται για τη διαφοροποίηση των διαφόρων τμημάτων ενός πηγαίου κώδικα της Visual Prolog. Αν και είναι μια αντικειμενοστραφή γλώσσα, είναι δυνατό να αναπτυχθεί κώδικας που αποφεύγει τα αντικειμενοστραφή χαρακτηριστικά της γλώσσας [15].

1.4 FutureDial

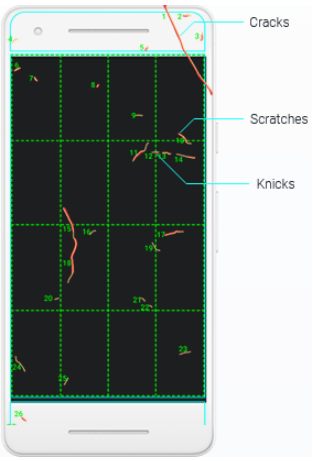
Εργάζεται για μεγάλες εταιρείες όπως η Sprint και η Verizon, το FutureDial είναι ένα αυτοματοποιημένο λογισμικό για τη βιομηχανία κινητών συσκευών. Χρειάζονται λιγότερο από 90 δευτερόλεπτα για τη διάγνωση μιας συσκευής κινητού τηλεφώνου. Παρέχει ακριβή βαθμολόγηση για τον προσδιορισμό της σωστής αξίας μιας συσκευής κινητού τηλεφώνου. Ορισμένες από τις πολλαπλές δοκιμές του περιλαμβάνουν αναγνώριση μοντέλου, επαλήθευση κλοπής/απώλειας, ανίχνευση κλειδώματος, υγεία μπαταρίας, διαγραφή δεδομένων και άλλα. Η επιλογή της FutureDial είναι στην πιο ακριβή πλευρά και είναι πιο κατάλληλη για καταστήματα με μαζικές παραγγελίες.

Το Future Dial είναι μια ολοκληρωμένη πλατφόρμα επεξεργασίας (The SMART Processing Platform) προσαρμοσμένη για τις ανάγκες του κάθε χρήστη. Είναι ένα αρθρωτό σύστημα λήψης, δοκιμής, και βαθμολόγησης φορητών συσκευών που χρησιμοποιούν ρομποτική και τεχνητή νοημοσύνη για την παροχή μιας αυτοματοποιημένης ροής ενός κομματιού για μια ολοκληρωμένη λύση από άκρο σε άκρο.

1.4.1 Λειτουργείες του FutureDial

Προκαταρκτικός έλεγχος συσκευής:

Προσδιορίζει αν αξίζει να διασωθεί ένα τηλέφωνο προτού εισέλθει στην αλυσίδα εφοδιασμού, εξοικονομώντας χρόνο και χρήμα. Αυτοματοποιεί τη διαδικασία αναγνώρισης και επιθεώρησης κινητής συσκευής στο σημείο συλλογής σε λιγότερο από 90



Εικόνα 7 Αναγνώριση αισθητικής κατάστασης συσκευής [16]

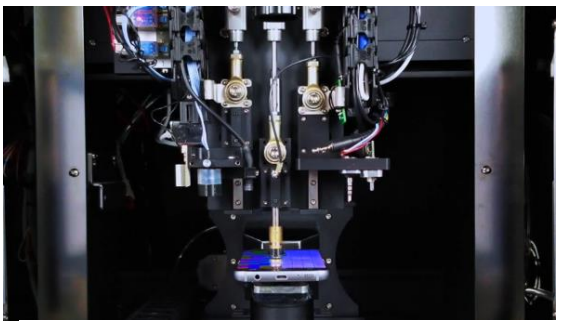
δευτερόλεπτα με μια λύση μηδενικού αποτυπώματος που βασίζεται σε νέφος (Cloud) που εκτελείται σε οποιαδήποτε έξυπνη κινητή συσκευή ή υπολογιστή.

Προσδιορίζει την πραγματική αξία των μεταχειρισμένων κινητών

Λαμβάνει καλύτερες αποτιμήσεις συσκευών με τα ρομπότ βαθμολόγησης εξωτερικής εμφάνισής (ραγίσματα, γρατσουνιές, χτυπήματα). Παρέχει συνεπή και ακριβή βαθμολόγηση, ώστε τόσο οι αγοραστές όσο και οι πωλητές να γνωρίζουν την πραγματική αξία μιας συσκευής. Στην εικόνα 8 βλέπουμε ένα παράδειγμα αποτίμησης του ρομπότ βαθμολόγησης εξωτερικής εμφάνισης

Γρήγορη και ακριβής δοκιμή και εκκαθάριση δεδομένων

Διευκολύνει τον έλεγχο και την αποτελεσματική εκκαθάριση φορητών συσκευών. Τόσο οι αυτοματοποιημένες όσο και οι μη αυτόματες λύσεις δοκιμών της εφαρμογής προσφέρουν εκτεταμένες δυνατότητες δοκιμών, επιταχυνόμενους χρόνους κύκλου δοκιμών και ελάχιστο χειρισμό συσκευών. για εργασίες μεγάλου όγκου, το ρομπότ SMART Test προσφέρει πρωτοφανή αποτελεσματικότητα.



Εικόνα 8 SMART Test Robot [16]

- Αναγνώριση μοντέλου
- Ανίχνευση κλειδώματος
- Λειτουργική διάγνωση
- Κινητή Υπηρεσία
- Υγεία μπαταρίας
- Επαλήθευση κλεμμένης συσκευής
- Επαλήθευση FMIP / FMAP
- Διαγραφή όλων των δεδομένων

All in one solution (Λύση όλα σε ένα)

Από την παραλαβή μέχρι τη μεταπώληση και οτιδήποτε ενδιάμεσο, το FutureDial παρέχει τα εργαλεία που χρειάζεται για να βελτιωθεί η λειτουργία επεξεργασίας της φορητής συσκευής.

Βελτιώσεις:

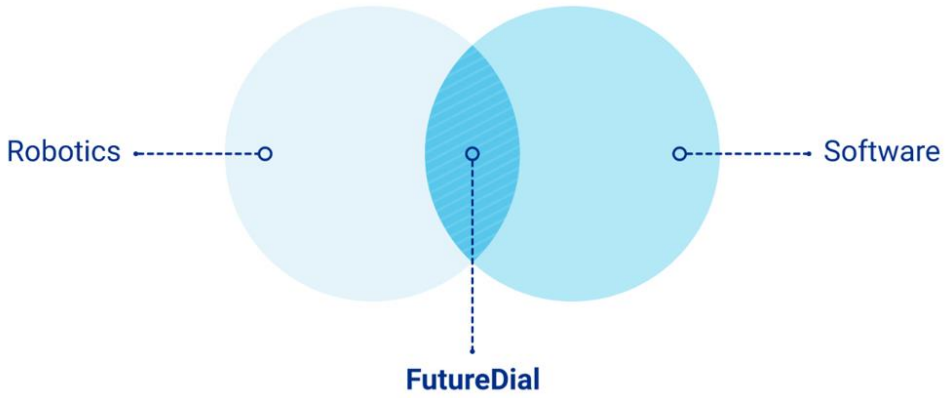
- Εργασιακή παραγωγικότητα
- Συνοχή
- Αξία παραγωγής
- Ασφάλεια εργαζομένων

Περιορίζει:

- Χρόνο επεξεργασίας
- Εργατικό κόστος
- Χειροκίνητες εργασίες
- Λάθη

Σουίτα προσαρμόσιμων λύσεων για την επεξεργασία φορητών συσκευών.

Επεκτείνει τις λύσεις λογισμικού με αποκλειστική ρομποτική αιχμής, τεχνητή νοημοσύνη και μηχανική μάθηση για να προσφέρει μέγιστη αξία και αποτελεσματικότητα [16]. Στην εικόνα 9 βλέπουμε τη συσχέτιση που έχει η FutureDial με την ρομποτική και το λογισμικό.



Εικόνα 9 Σχέση FutureDial με ρομποτική και λογισμικό. [16]

1.5 PiceaSoft

Από διαγνωστικά συσκευών έως τηλέφωνα εμπορίας, η Picea Services προσφέρει μια ολοκληρωμένη λύση επισκευής σε συνεργεία επισκευής κινητών τηλεφώνων. Ορισμένες από τις δυνατότητες του περιλαμβάνουν είναι η μεταφορά δεδομένων,

διαγραφή δεδομένων και ξεκλείδωμα συσκευής. Η διάγνωση διαρκεί 2-3 λεπτά ανά συσκευή και εντοπίζει προβλήματα υλικού και λογισμικού.

Το Picea μπορεί να διαγράψει έως και 32 συσκευές ταυτόχρονα. Διαθέτει αυτόματη βαθμολόγηση οθόνης AI. Ο έλεγχος μετά την επισκευή διαπιστώνει ότι το πρόβλημα έχει επιλυθεί σωστά και ότι ο τεχνικός δεν έχει κάνει κανένα λάθος.

1.5.1 Picea Sevices

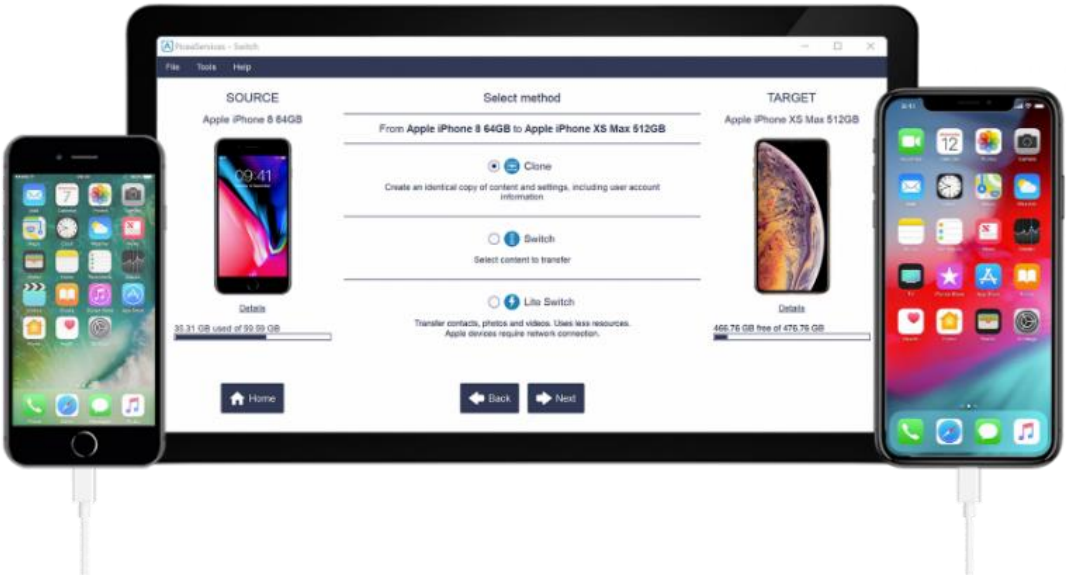
Οι Υπηρεσίες Picea προσφέρουν μεταφορά δεδομένων κινητής συσκευής, διαγνωστικά συσκευών, ανταλλαγή και διαγραφή δεδομένων κινητής τηλεφωνίας για εταιρείες κινητής τηλεφωνίας, καταστήματα λιανικής και σημεία εξυπηρέτησης. Αυτή η σπονδυλωτή λύση μόνο με λογισμικό είναι εύκολη στη χρήση και υποστηρίζει όλες τις συσκευές και τα κύρια λειτουργικά συστήματα.

Εικόνα 10 Picea Services [17]

1.5.2 Picea Switch

Το Picea Switch επιτρέπει τη γρήγορη και εύκολη μεταφορά προσωπικού περιεχομένου μεταξύ οποιωνδήποτε δύο φορητών συσκευών. Υποστηρίζει ασφαλή δημιουργία αντιγράφων ασφαλείας/επαναφοράς σε σύννεφο ή μονάδα flash USB. Στην εικόνα 11 βλέπουμε ένα στιγμιότυπο οθόνης της εφαρμογής Picea Switch που δείχνει της επιλογές μεταφοράς δεδομένων.

- Ομαλή και γρήγορη μεταφορά όλου του προσωπικού περιεχομένου μεταξύ όλων των μεγάλων πλατφορμών τηλεφώνου.
- Εξοικονομεί χρόνο επιλέγοντας την καλύτερη μέθοδο μεταφοράς περιεχομένου τηλεφώνου σε κάθε περίπτωση.



Εικόνα 11 Μέθοδοι μεταφοράς του Picea Switch [17]

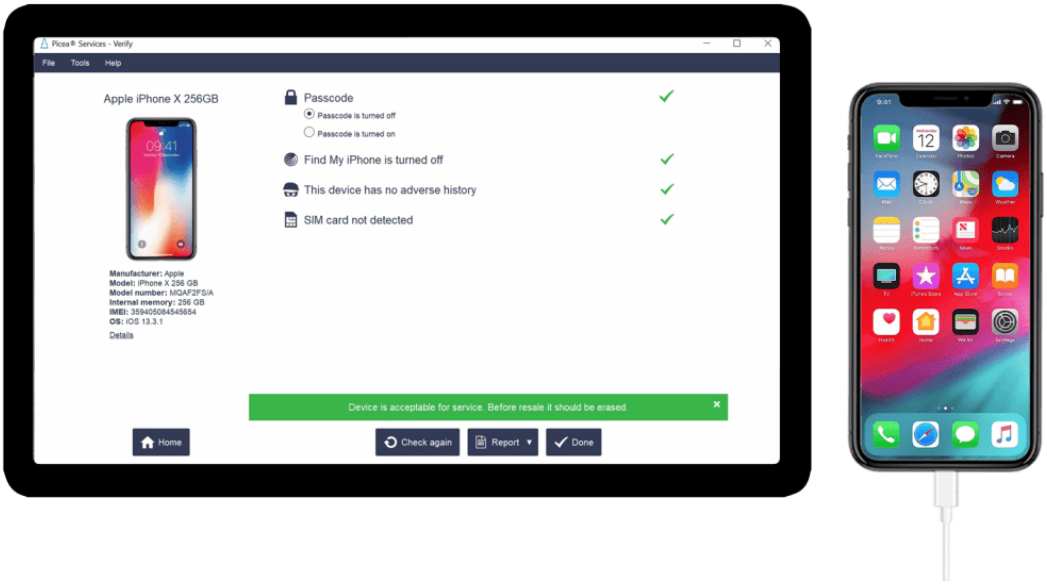
Μέθοδοι μεταφοράς:

1. Clone: Δημιουργεί ένα πανομοιότυπο αντίγραφο τον περιεχομένων της συσκευής, τις ρυθμίσεις περιλαμβάνοντας τις πληροφορίες του λογαριασμού χρήστη
2. Switch: Μεταφορά επιλεγμένου περιεχομένου
3. Lite Switch: Μεταφορά επαφών, φωτογραφιών και βίντεο. Αυτή η μέθοδος χρησιμοποιεί λιγότερους πόρους

1.5.3 Picea Verify

Το Picea Verify ελέγχει εάν οι συσκευές είναι αποδεκτές για μεταπώληση, σέρβις ή αγορά ελέγχοντας για ενεργά κλειδώματα συσκευών, τοποθετημένες κάρτες SIM ή μνήμης και έναντι του GSMA Device Check. Στην εικόνα 12 βλέπουμε τους ελέγχους που εκτελεί το Picea Verify.

- Εύκολος και γρήγορος έλεγχος λογαριασμών χρηστών και κλειδαριών.
- Επαλήθευση συσκευής με πληροφορίες IMEI



Εικόνα 12 Έλεγχος αποδεκτών συσκευών στο Picea Verify [17]

1.5.4 Picea Diagnostics

Το Picea Diagnostics παρέχει αυτόματες και εκτενείς δοκιμές φορητών συσκευών όπως βλέπουμε στην εικόνα 13, συμπεριλαμβανομένης της βαθμολόγησης συσκευών με ελάχιστη ανθρώπινη αλληλεπίδραση. Διαθέτει πλήρως αυτοματοποιημένες περιπτώσεις με τη βοήθεια του χρήστη και ανάλυση λογισμικού εντός 2-3 λεπτών ανά συσκευή. Τα διαγνωστικά βοηθούν τα καταστήματα λιανικής να αποφεύγουν τις επιστροφές χωρίς σφάλματα και τις περιττές παραδόσεις των συσκευών στο κέντρο επισκευής.

- Γρήγορη και απλή διαγνωστική διαδικασία
- Προσδιορίζει προβλήματα Software και Hardware του τηλεφώνου
- Εκτεταμένες λειτουργίες επισκευής για εφαρμογές και διαχείριση περιεχομένου
- Πλήρης διαδρομή ελέγχου για την επαλήθευση και την απόδειξη του σωστού χειρισμού της συσκευής
- Βοηθά στην αποφυγή επιστροφών No Fault Found (NFF) και περιττών παραδόσεων στο κέντρο επισκευής
- Αυτόματη βαθμολόγηση οθόνης με χρήση AI
- Το ιστορικό αναφορών παρέχει βεβαιότητα ότι το τηλέφωνο λειτουργεί πλήρως μετά την επισκευή

- Εξοικονόμηση κόστους όταν τα πιθανά προβλήματα μπορούν να επιλυθούν χωρίς επισκευή

Εικόνα 13 Διαγνωστικοί ελέγχοι του Picea Diagnostics [17]

1.5.5 Picea Trade-In

Το Picea Trade-In Παρέχει ακριβή και δίκαιη αξία συναλλαγών για μεταχειρισμένες κινητές συσκευές. Η τιμή συναλλαγής εκχωρείται αναγνωρίζοντας τη μάρκα και το μοντέλο της, επιβεβαιώνοντας ότι η συσκευή είναι έτοιμη για επαναχρησιμοποίηση και αναλύοντας την κατάσταση της. Στην εικόνα 14 βλέπουμε ένα παράδειγμα του Picea Trade-in

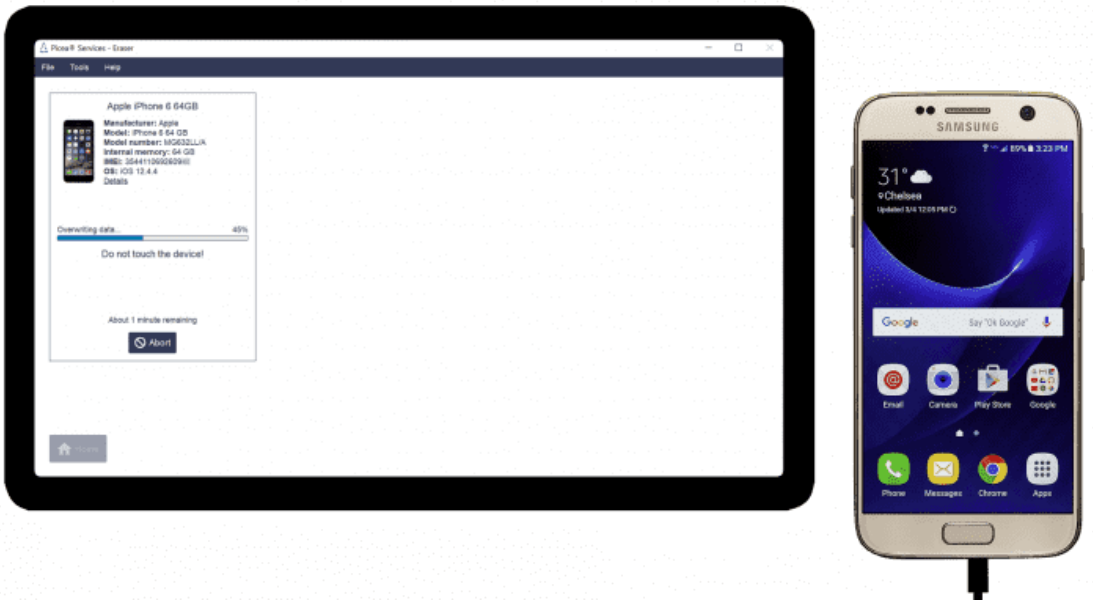
- Ακριβής αξία συναλλαγής για μεταχειρισμένες συσκευές
- Μπορεί να αναπτυχθεί σε καταστήματα χειριστή και λιανοπωλητών
- Βελτιωμένη αποτελεσματικότητα της διαδικασίας συναλλαγών, τόσο στο κατάστημα όσο και στην αποθήκη
- Επαλήθευση έναντι του GSMA Device Check
- Βεβαιώνει ότι οι συσκευές έχουν διαγνωστεί και ότι οι μέθοδοι κλειδώματος έχουν αφαιρεθεί ήδη

Εικόνα 14 Κατάσταση ιστορικού συσκευής του Picea Trade-in [17]

1.5.6 Picea Eraser

Το Picea Eraser επιτρέπει την ασφαλή απόρριψη, επαναχρησιμοποίηση ή μεταπώληση κινητών συσκευών διαγράφοντας οριστικά όλο το ευαίσθητο περιεχόμενο των χρηστών όπως βλέπουμε στο παράδειγμα της εικόνας 15. Επιπλέον μέχρι 32 τηλέφωνα ή tablet μπορούν να διαγραφούν ταυτόχρονα.

- Αναφορές με βάση το cloud, υποστήριξη συστημάτων ERP (Enterprise resource planning)
- Αδειάζει πλήρως τη μνήμη της συσκευής, η αναφορά διαγραφής περιέχει όλες τις λεπτομέρειες της συσκευής
- Αξιόπιστο: κανένα προσωπικό στοιχείο δεν θα καταλήξει σε λάθος χέρια
- Η παλιά συσκευή μπορεί να πουληθεί, να ανακυκλωθεί ή να χαριστεί με ασφάλεια
- Πιστοποίηση ADISA και συμμορφώνεται με τους κανονισμούς προστασίας δεδομένων όπως NIST, EU GDPR και R2



Εικόνα 15 Οριστική διαγραφή αρχείων με το Picea Eraser [17]

1.5.7 Picea Device Data

Τα Device Data καλύπτουν λεπτομερείς και επικυρωμένες πληροφορίες για 10.000 μεμονωμένα μοντέλα συσκευών και είναι διαθέσιμα μέσω του Device Data API. Εξοικονομεί κόστος και επαναφέρει αυτόματα εκτενείς λεπτομέρειες για τις ιδιότητες κινητής συσκευής και τις τεχνικές πληροφορίες στα συστήματα, όπως ιστότοπους ηλεκτρονικού εμπορίου ή ERP

1.5.8 Picea History

Το PiceaHistory παρέχει ορατότητα στο ιστορικό λειτουργίας μιας κινητής συσκευής σε όλα τα σημεία επαφής. Βελτιώνει την εξυπηρέτηση πελατών και την αποτελεσματικότητα της διαδικασίας ελέγχοντας τα αρχεία σέρβις μιας κινητής συσκευής που πραγματοποιήθηκαν στο σπίτι, στο σημείο πώλησης ή στην αποθήκη.

- Βελτιώνει την ορατότητα των λειτουργιών κινητής συσκευής
- Εμφανίζει λεπτομέρειες των λειτουργιών που πραγματοποιήθηκαν νωρίτερα σε μια κινητή συσκευή

1.5.9 Picea Reporting

Το Picea Reporting που βασίζεται σε cloud εμφανίζει στατιστικά στοιχεία σε πραγματικό χρόνο σε επίπεδο καταστήματος και εταιρείας, καθώς και λεπτομερείς πληροφορίες για τη συσκευή. Παρακολουθεί την επιχείρηση, τις πωλήσεις, βελτιστοποιεί τις προ παραγγελίες ή σχεδιάζει καμπάνιες μάρκετινγκ.

- Πληροφορίες αγοράς τηλεφώνου σε πραγματικό χρόνο σε κατασκευαστές τηλεφώνων, χειριστές και άλλους προμηθευτές κινητών τηλεφώνων
- Εργαλείο αναφοράς Ιστού για επιχειρηματικά αναλυτικά στοιχεία
- Γρήγορες, αξιόπιστες, ολοκληρωμένες πληροφορίες, για παράδειγμα, σχετικά με την αποτελεσματικότητα της καμπάνιας μάρκετινγκ

1.5.10 Picea AI-Powered Diagnostics

Το AI-Powered Screen Grading αξιοποιεί τη δύναμη της τεχνητής νοημοσύνης για αυτόματη ανάλυση και ταξινόμηση ζημιών στην οθόνη της κινητής συσκευής. Η ενότητα είναι διαθέσιμη στο PiceaDiagnostics και στο PiceaMobile. Παρέχει πιο αποτελεσματική ανάλυση κατάστασης οθόνης, μειώνει την υποκειμενικότητα και τη διακύμανση στη βαθμολόγηση της οθόνης και βελτιώνει την εμπειρία του πελάτη [17].

1.6 NSYS

Το NSYS είναι ένα σύστημα που βασίζεται σε cloud, ώστε όλα τα παλιά δεδομένα να μπορούν να ανακτηθούν εύκολα. Δεδομένου ότι η ανθρώπινη συμμετοχή είναι ελάχιστη, υπάρχουν λιγότερες πιθανότητες διαγνωστικού λάθους.

Παρέχει πλήρη αυτοματοποίηση της διαχείρισης αποθεμάτων (NSYS Buyback), των διαγνωστικών κινητών συσκευών(NSYS Diagnostics), της διαγραφής δεδομένων(NSYS Data Erasure) , των συναλλαγών επαναγοράς και διαπραγμάτευσης και της διαβάθμισης τηλεφώνου. Στο σχήμα 4 βλέπουμε την σειρά με την οποία πραγματοποιούνται οι εργασίες της NSYS.

Σχήμα 4 Βήματα εκτέλεσης εργασιών του NSYS. [18]

1.6.1 NSYS Buyback

Αυτοματοποιημένο σύστημα για επαναγορά και διαχείριση συναλλαγών στο κατάστημα, σε έναν ιστότοπο και μέσω εφαρμογής.

- Εξάλειψη του ανθρώπινου λάθους μέσω της αυτοματοποίησης της διαδικασίας δοκιμών
- Λήψη ακριβής τιμή της συσκευής μέσω λεπτομερών δοκιμών τόσο της αισθητικής κατάστασης όσο και της λειτουργικότητας

1.6.2 NSYS Autograding

Λύση που τροφοδοτείται από τεχνητή κατάσταση για συνεπή διαβάθμιση τηλεφώνου. Επιθεωρεί αυτόματα την αισθητική κατάσταση των χρησιμοποιημένων και ανακατασκευασμένων συσκευών.

Το Autograding γίνεται με την χρήση εφαρμογής φτιαγμένη για κινητά smartphone και χρησιμοποιεί την κάμερα για να κάνει λήψη όλων τον πλευρών μιας συσκευής και στην συνέχεια την βαθμολογεί με την χρήση αυτοματοποιημένης λύσης που καθοδηγείται από τεχνητή νοημοσύνη. Στην παρακάτω εικόνα 16 βλέπουμε τους βαθμούς αξιολόγησης συσκευών με βάση την φθορά και τα ελαττώματα τους.

Εικόνα 16 Περιγραφή βαθμολόγησης αισθητικής της συσκευής [18]

- Γρήγορη βαθμολόγηση. Αυτοματοποιεί τη διαδικασία ταξινόμησης για να ελευθερώσει χρόνο.
- Καταργεί την υποκειμενικότητα και το ανθρώπινο λάθος με την αυτοματοποιημένη λύση που καθοδηγείται από την τεχνητή νοημοσύνη
- Απλή χρήση εφαρμογής σε διάφορα σημεία πώλησης

1.6.3 NSYS Diagnostics

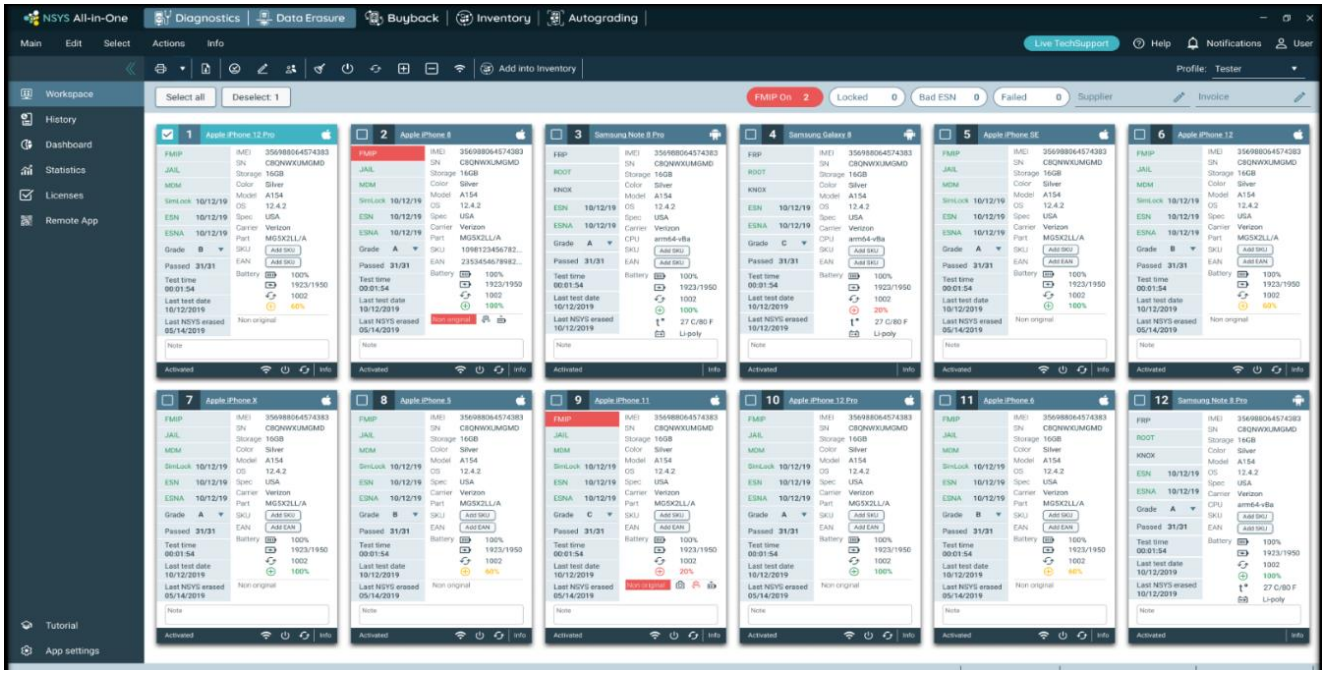
Αξιόπιστο λογισμικό για διαγνωστικά κινητών συσκευών. Περισσότερες από 60 αυτοματοποιημένες και ημιαυτόματες δοκιμές για συσκευές iOS και Android. Παροχή πλήρους, πιστοποιημένων διαγνωστικών μέσα σε 2-3 λεπτά, μείωση του ανθρώπινου λάθους και σημαντική αύξηση της παραγωγικότητας. Έτσι, είναι ιδιαίτερα χρήσιμο για ιδιοκτήτες καταστημάτων επισκευής που έχουν συμβόλαια με σχολεία, καταστήματα μεταφορέων ή άλλους οργανισμούς και λαμβάνουν μαζικές παραγγελίες επισκευής.

- Φιλικό προς το χρήστη περιβάλλον εργασίας
- Οποιαδήποτε συσκευή iOS ή Android
- Ταυτόχρονη δοκιμή πολλαπλών συσκευών
- Ολοκληρωμένη λύση με μεγάλο όγκο δοκιμών λειτουργικότητας
- Πιστοποιημένη διαγραφή δεδομένων
- Διαχείριση της ροής εργασίας με οποιονδήποτε υπολογιστή Windows ή Mac OS X
- Διάγνωση συσκευών χωρίς καλώδια και υπολογιστή

Επιπλέον παρέχει διαγνώστηκες υπηρεσίες μερικές εκ των οποίων είναι οι εξής:

- **IMEI Checks, ESN ESNA Simlock Check:** Δείχνει αν η συσκευή χάθηκε, κλάπηκε ή έχει χρέη κινητής τηλεφωνίας.
- **Data Erasure/Restore:** Διαγράφει όλα τα προσωπικά δεδομένα και ενημερώνει το λειτουργικό σύστημα της συσκευής στην πιο πρόσφατη έκδοση για πολλά smartphones με μερικά κλικ.
- **Cloud Database:** Αποθηκεύει αμέσως όλα τα δεδομένα μετά τη διάγνωση του τηλεφώνου και την αναφορά στο cloud, επιτρέποντάς σας να εργαστείτε από οποιονδήποτε υπολογιστή.
- **Label printing:** Επιτρέπει να δημιουργήσετε το δικό σας σχέδιο αυτοκόλλητων με λεπτομερείς πληροφορίες σχετικά με τις συσκευές.
- **Non-original parts detection:** Ελέγχει την αυθεντικότητα της οθόνης LCD, της μπαταρίας, των πίσω και των μπροστινών καμερών και της μητρικής πλακέτας σε συσκευές iOS

Στην παρακάτω εικόνα 17 βλέπουμε ένα στιγμιότυπο οθόνης της χρήσης της εφαρμογής στην κατηγορία Diagnostics όπου ελέγχονται ταυτόχρονα πολλαπλές συσκευές.

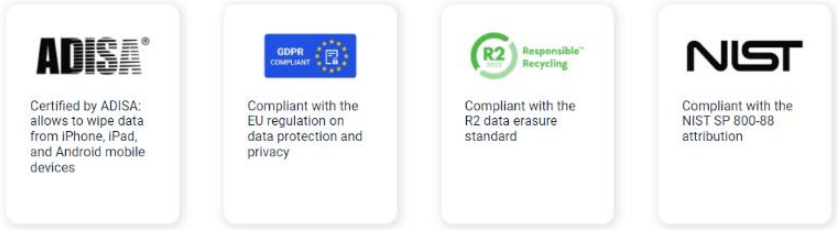


Εικόνα 17 Εφαρμογή NSYS All-in-One στην κατηγορία Diagnostics [18]

1.6.4 NSYS Data Erasure

Λύση για ασφαλή αφαίρεση δεδομένων από κινητές συσκευές. Συμμορφούμενος με NIST, ADISA και R2.

- Διαγραφή των δεδομένων από πολλές συσκευές ταυτόχρονα κατά την επεξεργασία μιας παρτίδας μεταχειρισμένων κινητών τηλεφώνων
- Επιβεβαίωση ότι δεν έχουν απομείνει εμπιστευτικά δεδομένα στις συσκευές από προηγούμενους ιδιοκτήτες κατά το κλείσιμο της συμφωνίας επαναπώλησης ή ανταλλαγής.
- Ανάκτηση πιστοποιητικού κάθε συσκευής για επιβεβαίωση ότι τα δεδομένα έχουν διαγραφεί σύμφωνα με τα διεθνή πρότυπα όπως βλέπουμε στην εικόνα 18



Εικόνα 18 Διεθνή πρότυπα διαγραφής και προστασίας δεδωμένων. [18]

1.6.5 NSYS Inventory

Το πρώτο σύστημα διαχείρισης αποθέματος που σχεδιάστηκε ειδικά για τη βιομηχανία μεταχειρισμένων συσκευών. Καταργεί τεράστια, μη διαχειριστικά υπολογιστικά φύλλα (Spreadsheets). Το απόθεμα NSYS δίνει ένα διαφανές, εύκολο στη χρήση σύστημα αποθέματος που έχει σχεδιαστεί ειδικά για τη βιομηχανία μεταχειρισμένων κινητών συσκευών. Παρέχει τον απόλυτο έλεγχο των ταμειακών ροών. Εντοπίζει τα πιο κερδοφόρα κανάλια πωλήσεων και ακολουθεί όλα τα οικονομικά σας με ένα προηγμένο ενσωματωμένο σύστημα παρακολούθησης χρημάτων. Διαχειρίζεται πολλές τοποθεσίες, ελέγχοντας απρόσκοπτα τη ροή των προϊόντων σας μεταξύ των αποθηκών.

- **IMEI Tracking:** Προσθέτει, αποθηκεύει και παρακολουθεί όλες τις συσκευές με το μοναδικό αριθμό IMEI από προμήθειες έως πωλήσεις

- **Multiple Warehouses:** Διαχειρίζεται όσες αποθήκες χρειαστεί. Ελέγχει τη ροή των προϊόντων μεταξύ των αποθηκών με ακριβή τρόπο με εντολές μετακίνησης και προσαρμογής.
- **ESN Checks & NSYS Diagnostics Integrated:** Μειώνει το κόστος με τον έλεγχο μαύρης λίστας GSMA και τον έλεγχο CarrierLock και εμφανίζει όλα τα διαγνωστικά αποτελέσματα σε ένα παράθυρο.
- **Suppliers and defects analytics:** Εντοπίζει ποια προϊόντα προμηθευτών φέρνουν τα περισσότερα έσοδα. Ερευνάει την πηγή συσκευών καλής ποιότητας με προηγμένη ανάλυση ελαττωμάτων & RMAs.
- **User Accountability & Permissions Management:** Παρακολουθεί τις ενέργειες που πραγματοποιούνται από οποιονδήποτε χρήστη, διαχειρίζεται δικαιώματα και ελέγχει τη διαθεσιμότητα πληροφοριών.
- **Built-In RMA functionality:** Η ενσωματωμένη λειτουργικότητα RMA επιτρέπει τη μείωση αριθμού ελαττωματικών συσκευών. Βοηθάει στη κατανόηση των κύριων λόγων για τις επιστροφές και την αξιοπιστία των προμηθευτών με διορατικά στατιστικά στοιχεία [18].

1.7 M360

Το M360 είναι ένα εξειδικευμένο διαγνωστικό λογισμικό, που καλύπτει τις ανάγκες της βιομηχανίας επισκευής κινητών τηλεφώνων. Είναι συμβατό τόσο με iOS όσο και με Android, καθιστώντας το ιδανικό για συνεργεία επισκευής τρίτων. Το M360 διαθέτει 26 δοκιμές για τον εντοπισμό σφαλμάτων στην κάμερα, την μπαταρία, την αποθήκευση, τη συνδεσιμότητα, τον ήχο, τους αισθητήρες, τα κουμπιά και την οθόνη. Αυτές οι δοκιμές χρειάζονται μόνο 2 λεπτά ανά συσκευή. Αποθηκεύει όλο το ιστορικό στο cloud και ως εκ τούτου, τα δεδομένα μπορούν να κοινοποιηθούν σε πολλά καταστήματα. Αυτό βοηθά τους πελάτες να επισκευάσουν τις συσκευές τους από την πλησιέστερη τοποθεσία. Το M360 μπορεί να μειώσει σημαντικά την πιθανότητα ανθρώπινων σφαλμάτων και να κάνει τις διαδικασίες πιο συνεπείς.

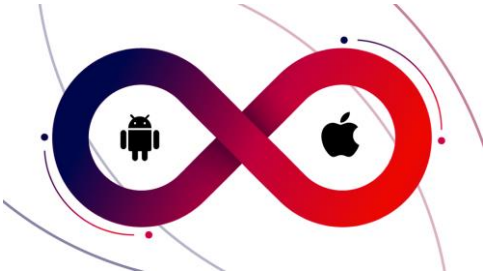
Το M360 αντλεί άμεσα πληροφορίες για οποιαδήποτε συσκευή τη στιγμή που συνδέεται. Η τεχνική διαγραφής δεδομένων του είναι γρήγορη και ασφαλής. Προσφέρει επίσης ετικέτες συσκευών για την διατήρηση της τάξης των επισκευών και των αγορών.

Κάποιες από τις λειτουργείες που μας προσφέρει η εφαρμογή M360 είναι οι εξής:

- Ενσωματωμένος έλεγχος Μαύρης λίστας IMEI (GSMA)
- Εμφανίζει άμεσα ολοκληρωμένες πληροφορίες της συσκευής
- Εκτέλεση πολλών δοκιμών για την λήψη ολοκληρωμένης διάγνωσης
- Εκτύπωση και κοινοποίηση αναφορών στους καταναλωτές
- Γρήγορη εκκαθάριση συσκευής ή σχολαστική διαγραφή με την πιστοποίηση της ADISA
- Παρακολούθηση προηγούμενων δοκιμών και τηλεφώνων με λεπτομερές ιστορικό εργασίας

1.7.1 M360 All-In-One

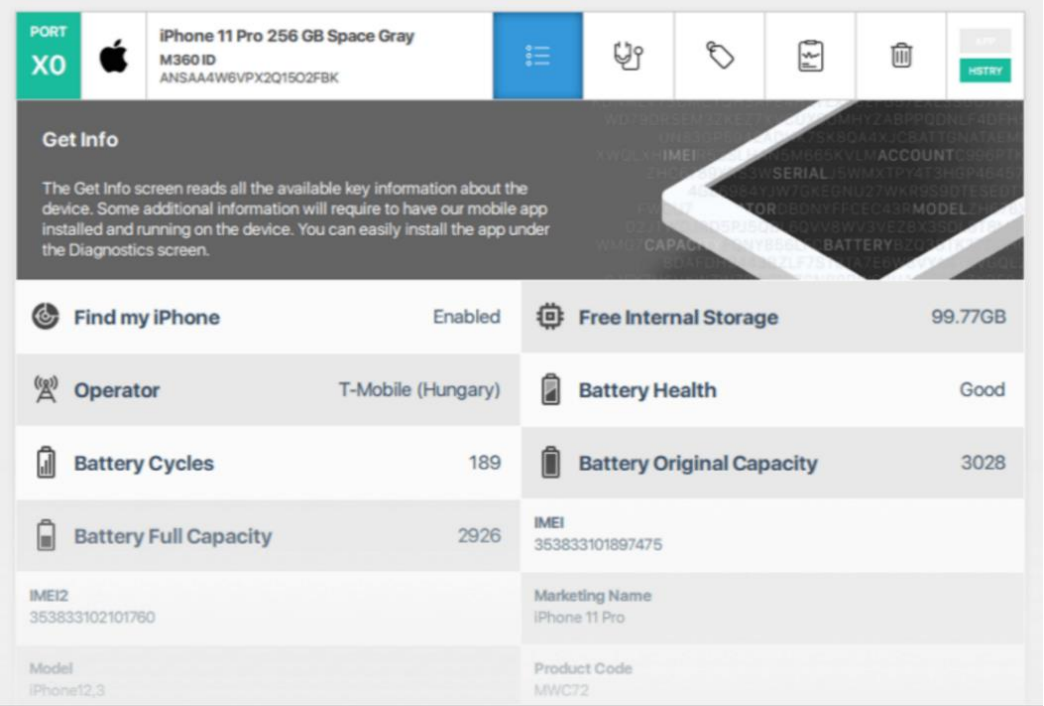
Το M360 είναι all-in-one εργαλείο που υποστηρίζει δεκάδες χιλιάδες συσκευές, ξεκινώντας από το Android 5.0 και το iOS 10.0, τόσο τα τηλέφωνα όσο και τα tablet. Με τον ενσωματωμένο εγκαταστάτη προγραμμάτων οδήγησης των Windows,μας επιτρέπει εύκολα να εγκαταστήσουμε και να ενημερώσουμε όλα τα απαιτούμενα προγράμματα οδήγησης για όλες τις επωνυμίες. Είναι επίσης πολύ γρήγορο στην υποστήριξη του νέου υλικού όπως πτυσσόμενα και νέα τηλέφωνα flip



Εικόνα 19 Υποστήριξη μοντέλων Android και iOS [19]

- **Υποστήριξη συσκευών Android και Apple:** Android 5.0 ή νεότερη έκδοση και iPhones / iPads με iOS 10.0 ή νεότερα
- **Ενσωματωμένο πρόγραμμα εγκατάστασης προγραμμάτων οδήγησης**: το M360 φροντίζει όλους τους απαιτούμενους οδηγούς και τους ενημερώνει

1.7.2 Get Info Screen



Εικόνα 20 Στιγμιότυπο οθόνης M360 Get Info [19]

Ή πρώτη οθόνη που εμφανίζεται όταν συνδέεται οποιοδήποτε τηλέφωνο ή tablet είναι η οθόνη Get Info όπως βλέπουμε στην παραπάνω εικόνα 20. Διαβάζει όλες τις διαθέσιμες βασικές πληροφορίες από τη συσκευή, όπως IMEI, σειριακό αριθμό, Λογαριασμό Google, Εγγύηση Samsung Knox, εμφάνιση της τοποθεσίας του iPhone με τη χρήση του Find My iPhone, την υγεία της μπαταρίας, την αναφορά αποθήκευσης και πολλά άλλα.

- **Υποστήριξη για Android 5.0 και άνω**: εμφανίζει 17 σημαντικές πληροφορίες
- **Υποστήριξη για iOS 10.0 και άνω**: εμφανίζει 15 σημαντικές πληροφορίες
- **Έξυπνη διάταξη**: Διάταξη για μία ή περισσότερες συσκευές

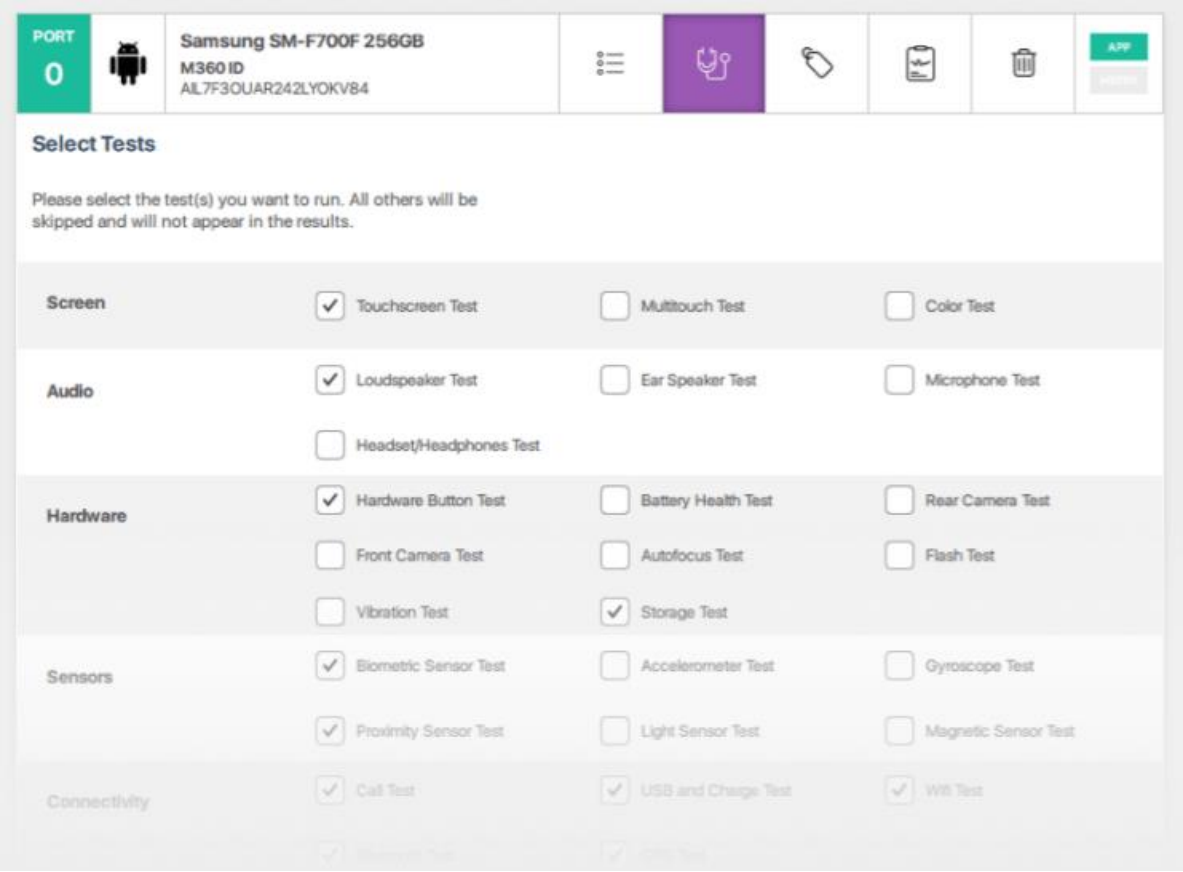
1.7.3 Diagnostics

Εκτελεί διάφορες δοκιμές για να διαγνώσει πολλές πτυχές μιας συσκευής. Κάθε δοκιμή ξεκινά με μια λεπτομερή περιγραφή και οδηγίες για τα βήματα. Επιτρέπει την επιλογή από τα αποθηκευμένα πρότυπα δοκιμών ή την επιλογή δοκιμών μία προς μία

πριν ξεκινήσει. Μόλις ο χρήστης εξοικειωθεί με τη ροή εργασίας δοκιμών, μπορεί να επιταχύνει τα πράγματα με τις εξειδικευμένες δυνατότητες, όπως οι αυτοματοποιημένες επικυρώσεις δοκιμών και η αυτοματοποιημένη εγκατάσταση εφαρμογής.

- **Δυνατότητες ελέγχου του M360:** Οθόνη ηχεία, μικρόφωνα, κουμπιά, κάμερες, βιομετρικοί αισθητήρες, επιταχυνσιόμετρο, γυροσκόπιο, αισθητήρας εγγύτητας, αισθητήρας φωτός, μαγνητικός αισθητήρας, Blue-tooth, Wifi, GPS, λειτουργία κλήσεων και μπαταρίας
- **30 και άνω δοκιμές:** δοκιμή του κάθε υλικού στοιχείου της συσκευής
- **Προσαρμοσμένη ροή εργασίας:** κοινόχρηστα πρότυπα δοκιμών σε όλη την ομάδα και τα καταστήματα
- **Γρήγορές δοκιμές με αυτοματισμό**

Στην εικόνα 21 βλέπουμε ένα στιγμιότυπο οθόνης του M360 στην κατηγορία Diag-nostics με κάποιους από τους ελέγχους.



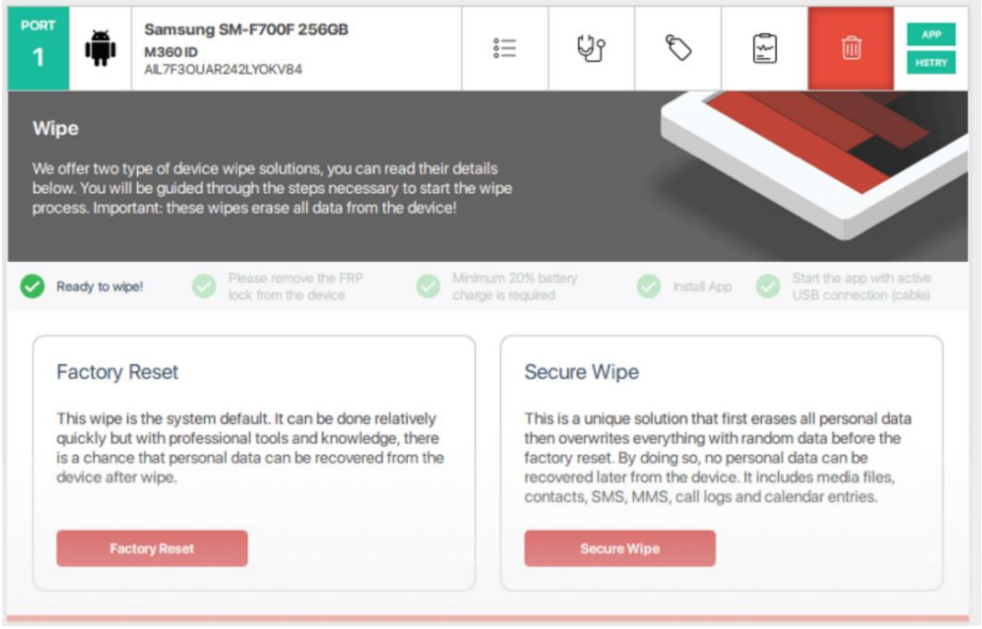
Εικόνα 21 Στιγμιότυπο οθόνης δοκιμών διαγνωστικού [19]

1.7.4 Wipe Solutions

Προσφέρει δύο τύπους λύσεων εκκαθάρισης συσκευών. Το ασφαλές καθάρισμα πρώτα διαγράφει όλα τα προσωπικά μέσα και έγγραφα και στη συνέχεια αντικαθιστά τα πάντα με τυχαία δεδομένα πριν κάνει επαναφορά εργοστασιακών ρυθμίσεων. Με αυτόν τον τρόπο, κανένα προσωπικό στοιχείο δεν μπορεί να ανακτηθεί αργότερα από τη συσκευή. Καλύπτει αρχεία πολυμέσων, επαφές, SMS, MMS, αρχεία καταγραφής κλήσεων και καταχωρήσεις ημερολογίου. Το άλλο είναι η προεπιλογή του συστήματος, επαναφορά στη μέθοδο εργοστασιακών ρυθμίσεων που είναι γρήγορη αλλά όχι εντελώς ασφαλής.

- **Ασφαλής εκκαθάριση:** καθαρίζει τις συσκευές με ασφάλεια, χωρίς δυνατότητα επαναφοράς προσωπικών δεδομένων
- **Γρήγορη εκκαθάριση:** εκκίνηση της προεπιλογής του συστήματος, επαναφορά στη μέθοδο εργοστασιακών ρυθμίσεων με ένα κλικ
- **Καθοδηγούμενη ροή εργασίας:** Ο έλεγχος απαιτήσεων βήμα προς βήμα εξασφαλίζει άψογη εκκαθάριση

Στην παρακάτω εικόνα 22 βλέπουμε ένα στιγμιότυπο οθόνης του M360 στη κατηγορία Wipe και τις δύο επιλογές εκκαθάρισης.

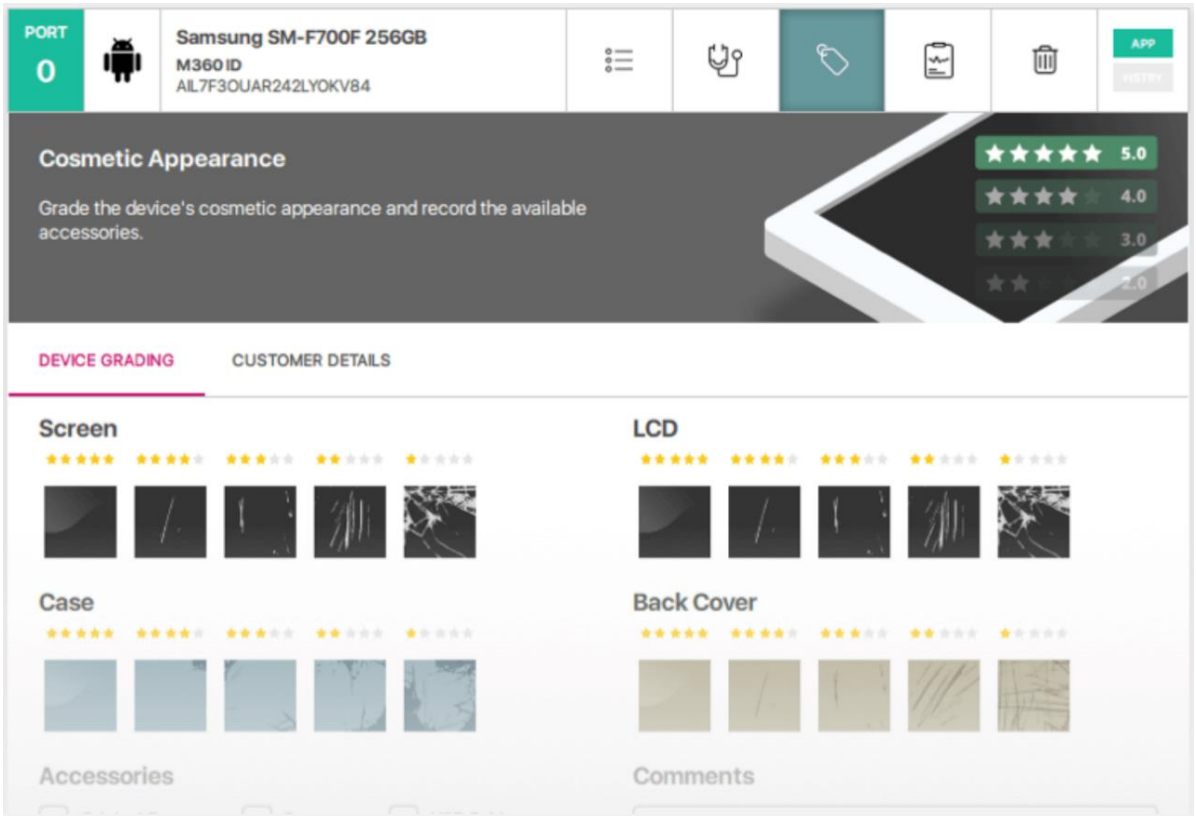


Εικόνα 22 Στιγμιότυπο οθόνης Wipe [19]

1.7.5 Cosmetic Appearance

Η ενότητα ταξινόμησης, μπορεί να καταγράψει την αισθητική εμφάνιση της συσκευής. Αξιολογεί την οθόνη, την οθόνη LCD, τη θήκη και την κατάσταση του πίσω καλύμματος με το παραδοσιακό σύστημα A-B-C-D-F ή με την οπτική μέθοδο της M360. Προσφέρει πλαίσια ελέγχου για την καταχώρηση της λίστας αξεσουάρ και έναν αναπτυσσόμενο επιλογέα για τον ορισμό σημαιών κατάστασης συσκευής, για την επισήμανση εάν μια συσκευή είναι σε λειτουργία ή έχει ήδη συντηρηθεί. Επιτρέπει επίσης την επισύναψη των στοιχείων του πελάτη σε μια συσκευή. Στην παρακάτω εικόνα 23 βλέπουμε ένα στιγμιότυπο της οθόνης του M360 στη κατηγορία Cosmetic Appearance.

- **Καταγραφή κατάστασης:** τέσσερις κατηγορίες, δύο συστήματα βαθμολόγησης, βαθμολόγηση σε δευτερόλεπτα
- **Καταχώρηση αξεσουάρ:** φορτιστής, χαρτιά, κουτί, ακουστικά.
- **Κατάσταση συσκευής και στοιχεία πελάτη:** ορισμός κατάστασης συσκευής στο κατάστημα και αποθήκευση δεδομένων πελατών



Εικόνα 23 Στιγμιότυπο οθόνης Cosmetic Appearance [19]

1.7.6 Diagnostics Report

Το M360 παρέχει πολύ λεπτομερείς αναφορές σχετικά με τις συσκευές και τα αποτελέσματα των δοκιμών τους. Επιτρέπει την εκτύπωση, την αποθήκευση, την κοινοποίηση online ή την εξαγωγή σε αρχείο CSV για περαιτέρω επεξεργασία. Οι πελάτες θα εκτιμήσουν την αναφορά κατάστασης συσκευής τρίτων, που είναι ανεξάρτητη και μπορεί να παραδοθεί αμέσως, πριν ή μετά την επισκευή. Το ιστορικό εργασίας περιέχει κάθε λεπτομέρεια για κάθε συσκευή που έχετε συνδέσει στο παρελθόν. Είναι κοινόχρηστο μεταξύ μιας ομάδας εάν λειτουργεί από πολλές τοποθεσίες ή υπολογιστές και αναγνωρίζει αμέσως συσκευές που έχουν συνδεθεί προηγουμένως. Με τις εκτεταμένες επιλογές αναζήτησης και φιλτραρίσματος, μπορεί εύκολα να βρει οποιαδήποτε εγγραφή από το παρελθόν.

- **Επαγγελματικά αποτελέσματα δοκιμών:** εκτύπωση, αποθήκευση, κοινή χρήση online και εξαγωγή αποτελεσμάτων
- **Λεπτομερές ιστορικό εργασίας:** αποθηκευμένο στο cloud, κοινόχρηστο σε όλους τους υπολογιστές και εύκολο στην αναζήτηση
- **Ετικέτες συσκευών:** εκτυπώνει ετικέτες συσκευών με τις επιλεγμένες λεπτομέρειες για την οργάνωση του αποθέματος [19].

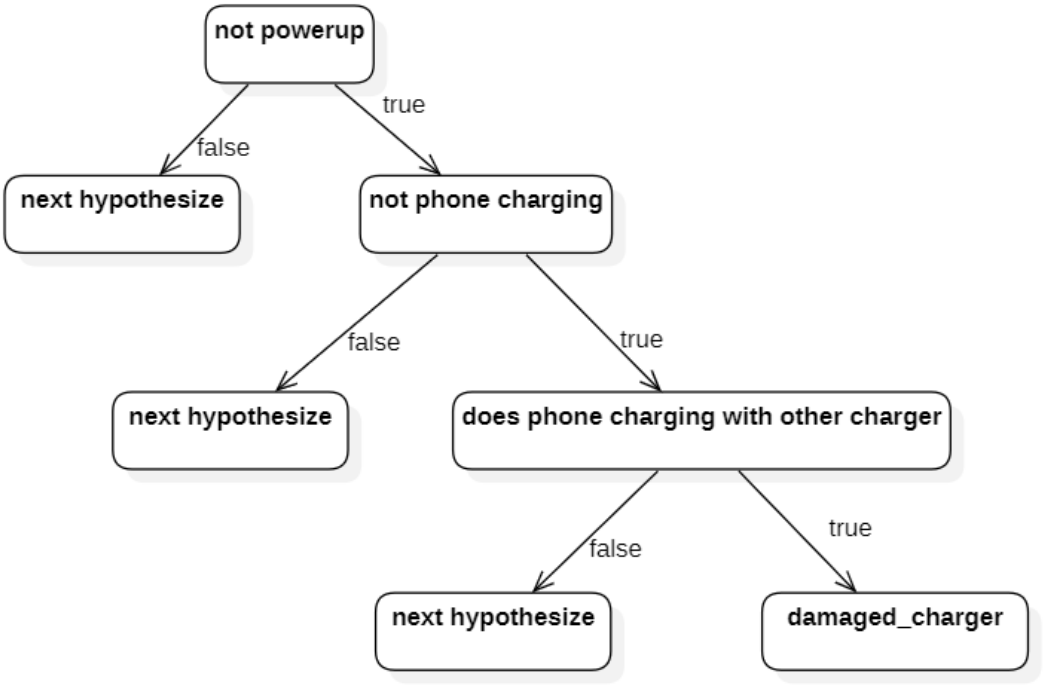
ΚΕΦΑΛΑΙΟ 2^{Ο}: Διαγράμματα Εφαρμογής

2.1 Διαγράμματα δέντρων αποφάσεων

Ένα δέντρο αποφάσεων είναι ένα μοντέλο που μοιάζει με δέντρο που λειτουργεί ως εργαλείο υποστήριξης αποφάσεων, εμφανίζοντας οπτικά τις αποφάσεις και τα πιθανά αποτελέσματα, τις συνέπειες και το κόστος τους. Από εκεί, τα «κλαδιά» μπορούν εύκολα να αξιολογηθούν και να συγκριθούν προκειμένου να επιλεγούν οι καλύτερες πορείες δράσης. Η ανάλυση δέντρου αποφάσεων είναι χρήσιμη για την επίλυση προβλημάτων, την αποκάλυψη πιθανών ευκαιριών και τη λήψη σύνθετων αποφάσεων σχετικά με τη διαχείριση κόστους, τη διαχείριση λειτουργιών, τις στρατηγικές οργάνωσης, την επιλογή έργων και τις μεθόδους παραγωγής.

Η σχεδίαση ενός διαγράμματος δέντρου απόφασης ξεκινά από αριστερά προς τα δεξιά και αποτελείται από κόμβους «έκρηξης» που χωρίζονται σε διαφορετικές διαδρομές. Οι κόμβοι κατηγοριοποιούνται ως κόμβοι ρίζας, οι οποίοι συγκεντρώνουν ολόκληρο το δείγμα και στη συνέχεια χωρίζονται σε πολλαπλά σύνολα. Οι κόμβοι απόφασης, που τυπικά αντιπροσωπεύονται από τετράγωνα, είναι υποκόμβοι που αποκλίνουν σε περαιτέρω δυνατότητες. και ο τερματικός κόμβος, που τυπικά αντιπροσωπεύεται από τρίγωνα, είναι ο τελικός κόμβος που δείχνει το τελικό αποτέλεσμα που δεν μπορεί να κατηγοριοποιηθεί περαιτέρω. Οι κλάδοι ή οι γραμμές αντιπροσωπεύουν τις διάφορες διαθέσιμες εναλλακτικές λύσεις και οι υποκόμβοι μπορούν να εξαλειφθούν μέσω του κλαδέματος [20].

Στα παρακάτω σχήματα θα δούμε την ανάλυση πιθανών βλαβών της εφαρμογής μας με τη χρήση διαγραμμάτων δέντρων αποφάσεων (decision trees).



Εικόνα 24 Υπόθεση βλάβης damaged charger

Στη παραπάνω εικόνα 24 αναλύουμε τα βήματα με τα οποία εκτελείτε ο κώδικας, για την υπόθεση «damage charger» με την χρήση διαγράμματος δέντρου αποφάσεων.

Αρχικά εξετάζεται το πρώτο σύμπτωμα “not powerup” ρωτώντας τον χρήστη εάν είναι αληθές ή ψευδές. Στην περίπτωση που ο χρήστης δώσει αρνητική απάντηση (false) το πρόγραμμα οδηγείτε στην επόμενη υπόθεση (next hypothesize), διαφορετικά σε περίπτωση που ο χρήστης δώσει θετική απάντηση (true) το πρόγραμμα οδηγείτε στο επόμενο σύμπτωμα “not phone charging”.

Έπειτα γίνεται ξανά ερώτηση στο χρήστη όπου στην περίπτωση που δοθεί ψευδής (false) απάντηση το πρόγραμμα οδηγείτε στην επόμενη υπόθεση, αντίθετα σε περίπτωση που δοθεί αληθής (true) απάντηση το πρόγραμμα οδηγείτε στο επόμενο σύμπτωμα “does phone charging with other charger”.

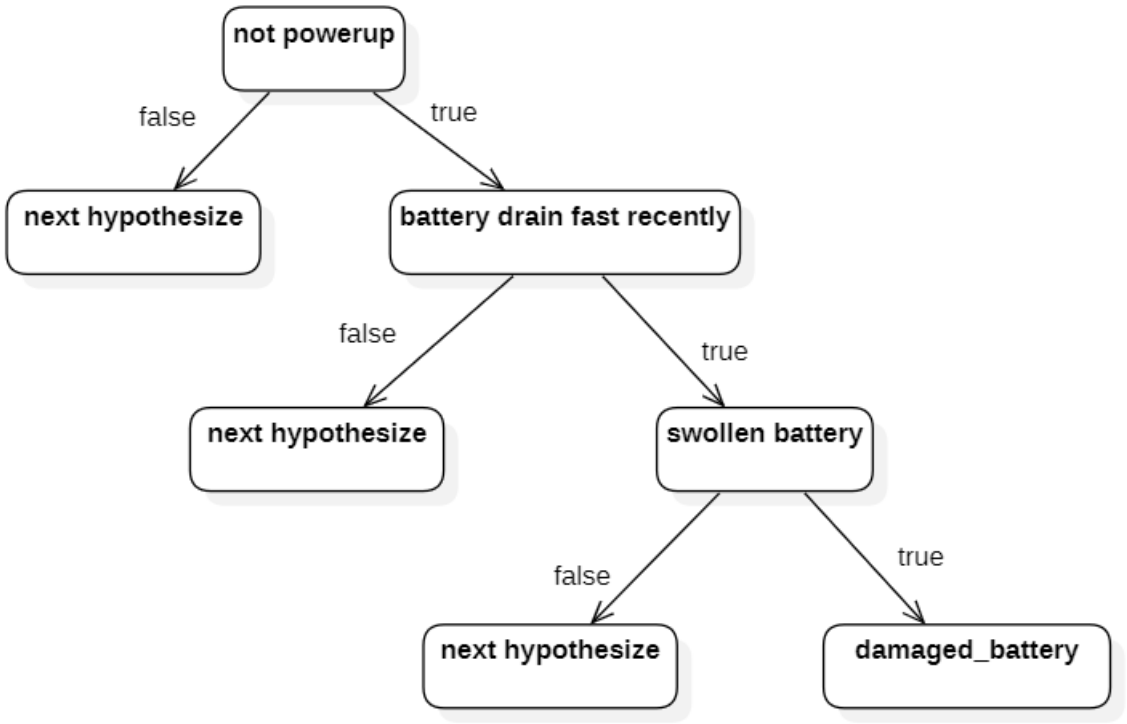
Επιπλέον γίνεται η τελευταία ερώτηση στο χρήστη όπου στην περίπτωση που δοθεί ψευδής (false) απάντηση το πρόγραμμα οδηγείτε στην επόμενη υπόθεση, διαφορετικά σε περίπτωση που δοθεί αληθής (true) εμφανίζει στον χρήστη τη διάγνωση πιθανής βλάβης «damaged charger».

Η εξής υλοποίηση του παραπάνω διαγράμματος βασίζεται στο παρακάτω τμήμα κώδικα της εφαρμογής:

damaged_charger :-not(powerup),

not(verify('phone charging ')),

verify('does phone charging with other charger ').



Εικόνα 25 Υπόθεση βλάβης damaged battery

Στη παραπάνω εικόνα 25 αναλύουμε τα βήματα με τα οποία εκτελείτε ο κώδικας, για την υπόθεση «damage battery» με την χρήση διαγράμματος δέντρου αποφάσεων.

Αρχικά εξετάζεται το πρώτο σύμπτωμα “not powerup” ρωτώντας τον χρήστη εάν είναι αληθές ή ψευδές. Στην περίπτωση που ο χρήστης δώσει αρνητική απάντηση (false) το πρόγραμμα οδηγείτε στην επόμενη υπόθεση (next hypothesize), διαφορετικά σε περίπτωση που ο χρήστης δώσει θετική απάντηση (true) το πρόγραμμα οδηγείτε στο επόμενο σύμπτωμα “battery drain fast recently”.

Έπειτα γίνεται ξανά ερώτηση στο χρήστη όπου στην περίπτωση που δοθεί ψευδής (false) απάντηση το πρόγραμμα οδηγείτε στην επόμενη υπόθεση, αντίθετα σε περίπτωση που δοθεί αληθής (true) απάντηση το πρόγραμμα οδηγείτε στο επόμενο σύμπτωμα “swollen battery”.

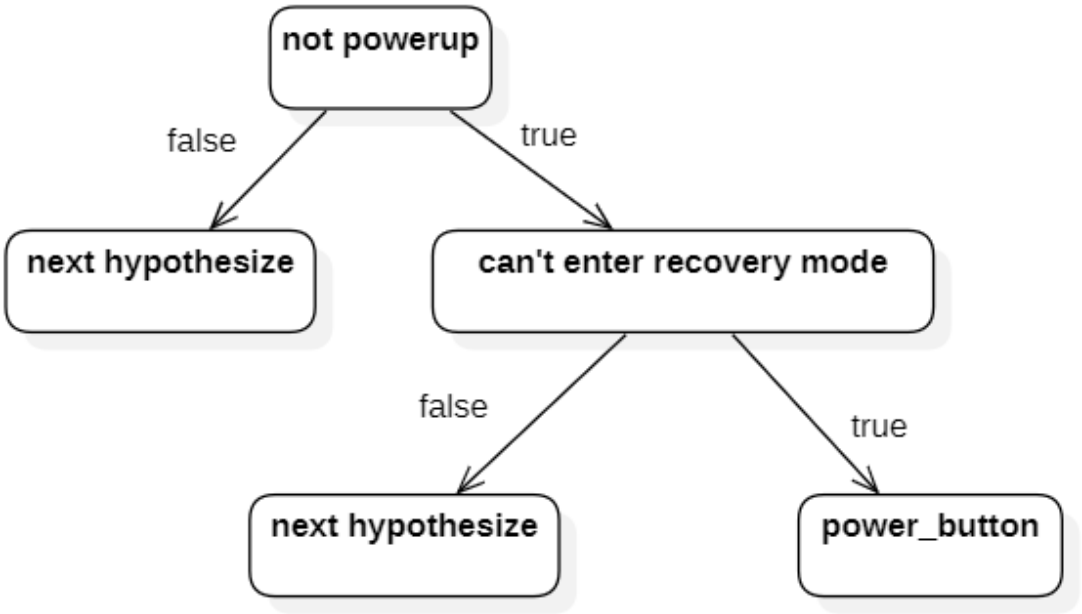
Επιπλέον γίνεται η τελευταία ερώτηση στο χρήστη όπου στην περίπτωση που δοθεί ψευδής (false) απάντηση το πρόγραμμα οδηγείτε στην επόμενη υπόθεση, διαφορετικά σε περίπτωση που δοθεί αληθής (true) εμφανίζει στον χρήστη τη διάγνωση πιθανής βλάβης «damaged battery».

Η εξής υλοποίηση του παραπάνω διαγράμματος βασίζεται στο παρακάτω τμήμα κώδικα της εφαρμογής:

damaged_battery :-not(powerup),

battery_drain,

verify('swollen battery ').



Εικόνα 26 Υπόθεση βλάβης power button

Στη παραπάνω εικόνα 26 αναλύουμε τα βήματα με τα οποία εκτελείτε ο κώδικας, για την υπόθεση «power button» με την χρήση διαγράμματος δέντρου αποφάσεων.

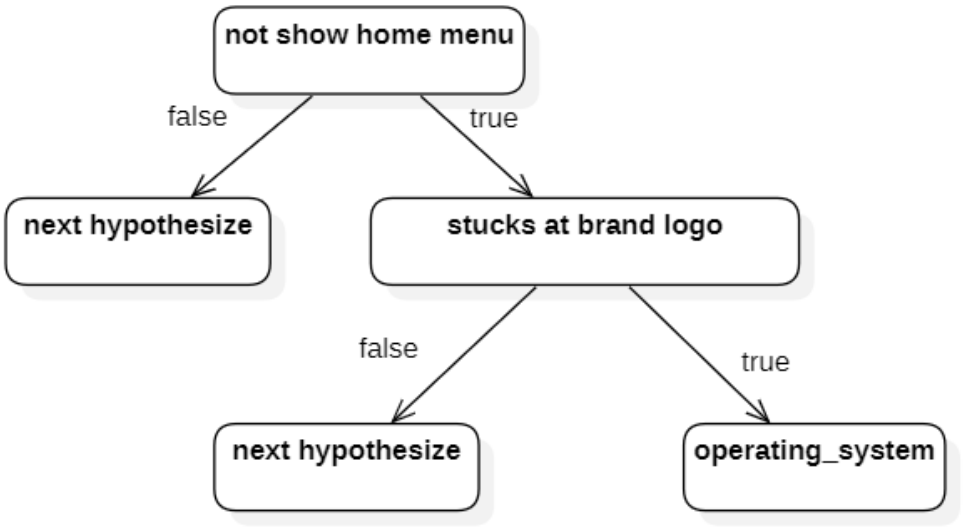
Αρχικά εξετάζεται το πρώτο σύμπτωμα “not powerup” ρωτώντας τον χρήστη εάν είναι αληθές ή ψευδές. Στην περίπτωση που ο χρήστης δώσει αρνητική απάντηση (false) το πρόγραμμα οδηγείτε στην επόμενη υπόθεση (next hypothesize), διαφορετικά σε περίπτωση που ο χρήστης δώσει θετική απάντηση (true) το πρόγραμμα οδηγείτε στο επόμενο σύμπτωμα “cant enter recovery mode”.

Έπειτα γίνεται η τελευταία ερώτηση στο χρήστη όπου στην περίπτωση που δοθεί ψευδής (false) απάντηση το πρόγραμμα οδηγείτε στην επόμενη υπόθεση, διαφορετικά σε περίπτωση που δοθεί αληθής (true) εμφανίζει στον χρήστη τη διάγνωση πιθανής βλάβης «power button».

Η εξής υλοποίηση του παραπάνω διαγράμματος βασίζεται στο παρακάτω τμήμα κώδικα της εφαρμογής:

power_button:-not(powerup),

not(verify('can you enter recovery mode ')).



Εικόνα 27 Υπόθεση βλάβης operating system

Στη παραπάνω εικόνα 27 αναλύουμε τα βήματα με τα οποία εκτελείτε ο κώδικας, για την υπόθεση «operating system» με την χρήση διαγράμματος δέντρου αποφάσεων.

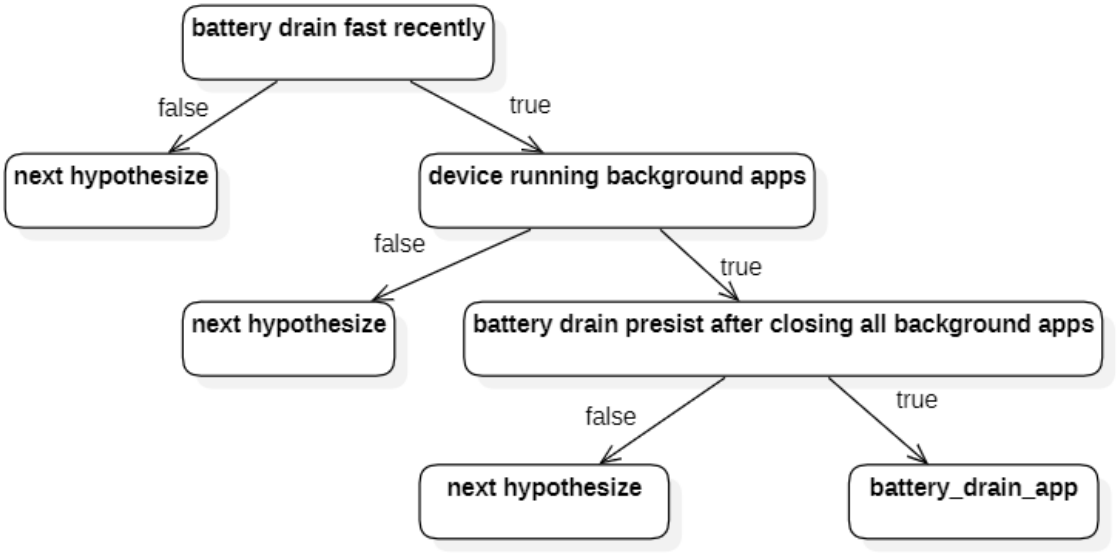
Αρχικά εξετάζεται το πρώτο σύμπτωμα “not show home menu” ρωτώντας τον χρήστη εάν είναι αληθές ή ψευδές. Στην περίπτωση που ο χρήστης δώσει αρνητική απάντηση (false) το πρόγραμμα οδηγείτε στην επόμενη υπόθεση (next hypothesize), διαφορετικά σε περίπτωση που ο χρήστης δώσει θετική απάντηση (true) το πρόγραμμα οδηγείτε στο επόμενο σύμπτωμα “stucks at brand logo”.

Έπειτα γίνεται η τελευταία ερώτηση στο χρήστη όπου στην περίπτωση που δοθεί ψευδής (false) απάντηση το πρόγραμμα οδηγείτε στην επόμενη υπόθεση, διαφορετικά σε περίπτωση που δοθεί αληθής (true) εμφανίζει στον χρήστη τη διάγνωση πιθανής βλάβης «operating system».

Η εξής υλοποίηση του παραπάνω διαγράμματος βασίζεται στο παρακάτω τμήμα κώδικα της εφαρμογής:

operating_system :-not(verify('show home menu ')),

verify('stucks at brand logo ').



Εικόνα 28 Υπόθεση βλάβης battery_drain_app

Στη παραπάνω εικόνα 28 αναλύουμε τα βήματα με τα οποία εκτελείτε ο κώδικας, για την υπόθεση «battery drain app» με την χρήση διαγράμματος δέντρου αποφάσεων.

Αρχικά εξετάζεται το πρώτο σύμπτωμα “battery drain fast recently” ρωτώντας τον χρήστη εάν είναι αληθές ή ψευδές. Στην περίπτωση που ο χρήστης δώσει αρνητική

απάντηση (false) το πρόγραμμα οδηγείτε στην επόμενη υπόθεση (next hypothesize), διαφορετικά σε περίπτωση που ο χρήστης δώσει θετική απάντηση (true) το πρόγραμμα οδηγείτε στο επόμενο σύμπτωμα “device running background apps”.

Έπειτα γίνεται ξανά ερώτηση στο χρήστη όπου στην περίπτωση που δοθεί ψευδής (false) απάντηση το πρόγραμμα οδηγείτε στην επόμενη υπόθεση, αντίθετα σε περίπτωση που δοθεί αληθής (true) απάντηση το πρόγραμμα οδηγείτε στο επόμενο σύμπτωμα “battery drain persist after closing all background apps”.

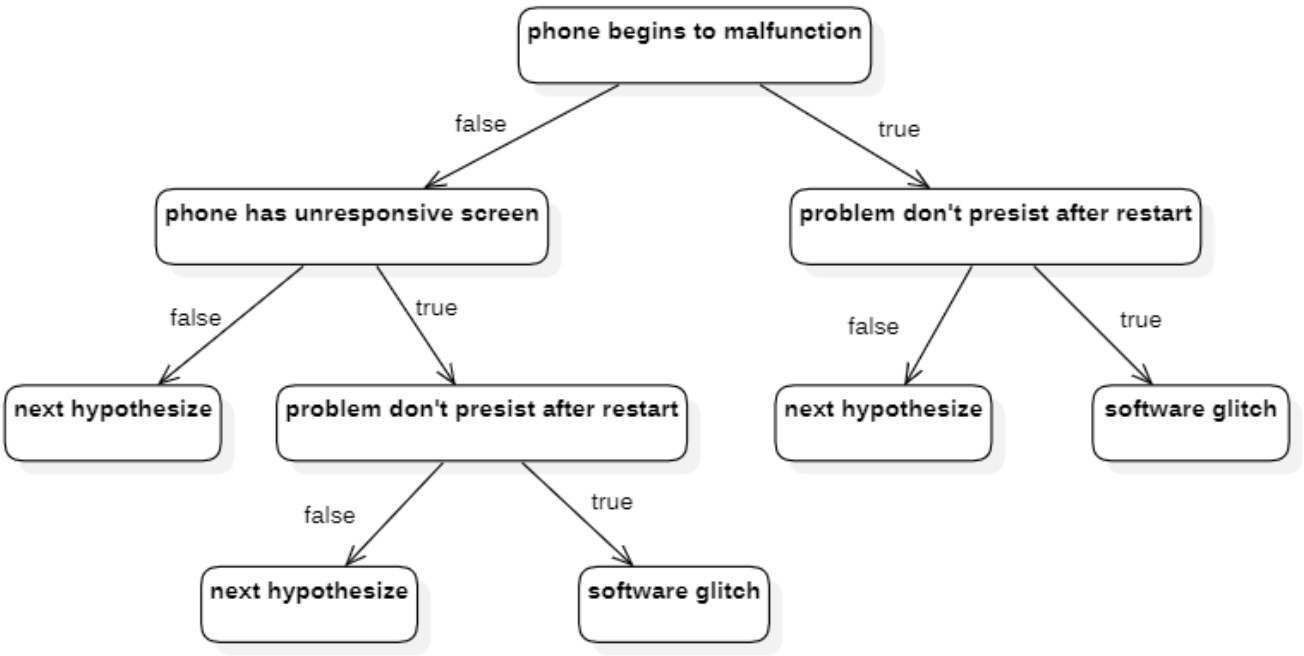
Επιπλέον γίνεται η τελευταία ερώτηση στο χρήστη όπου στην περίπτωση που δοθεί ψευδής (false) απάντηση το πρόγραμμα οδηγείτε στην επόμενη υπόθεση, διαφορετικά σε περίπτωση που δοθεί αληθής (true) εμφανίζει στον χρήστη τη διάγνωση πιθανής βλάβης «damaged battery».

Η εξής υλοποίηση του παραπάνω διαγράμματος βασίζεται στο παρακάτω τμήμα κώδικα της εφαρμογής:

battery_drain_app :-battery_drain,

verify('device running background apps '),

not(verify('battery drain persist after closing all background apps ')).



Εικόνα 29 Υπόθεση βλάβης software glitch

Στην παραπάνω εικόνα 29 αναλύουμε τα βήματα με τα οποία εκτελείτε ο κώδικας, για την υπόθεση «software_glitch» με την χρήση διαγράμματος δέντρου αποφάσεων.

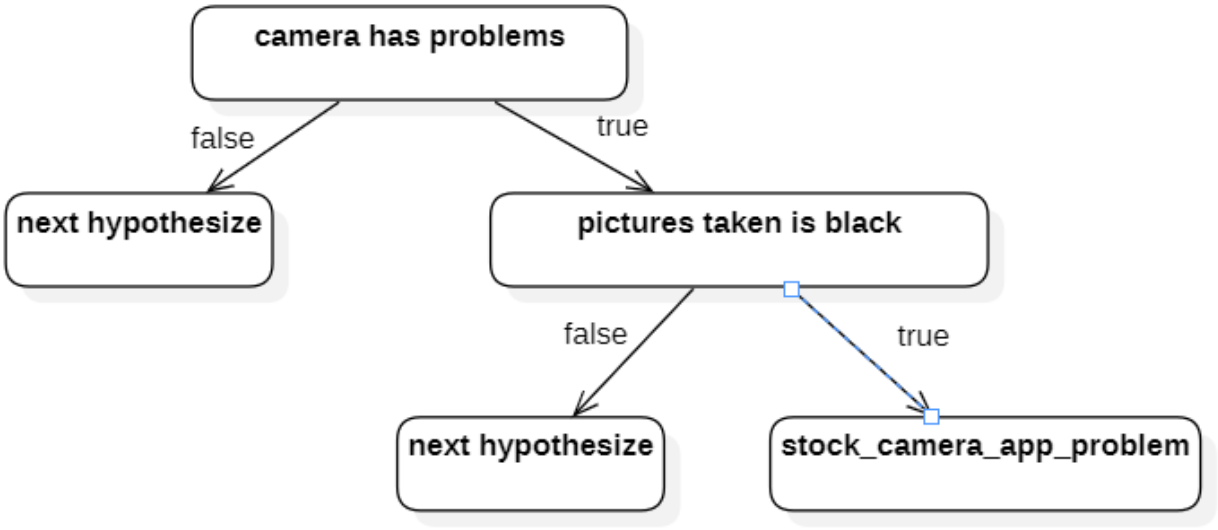
Αρχικά εξετάζεται το πρώτο σύμπτωμα “phone begins to malfunction” ρωτώντας τον χρήστη εάν είναι αληθές ή ψευδές. Στην περίπτωση που ο χρήστης δώσει αρνητική απάντηση (false) το πρόγραμμα οδηγείτε στο επόμενο σύμπτωμα “phone has unresponsive screen”, διαφορετικά σε περίπτωση που ο χρήστης δώσει θετική απάντηση (true) το πρόγραμμα οδηγείτε στο επόμενο σύμπτωμα “problem persist after restart”. Στο σημείο αυτό ο χρήστης θα έχει επιλέξει ένα από τα δύο μονοπάτια του δέντρου, συγκεκριμένα μεταξύ των συμπτωμάτων “phone has unresponsive screen” και “problem persist after restart”.

Στο πρώτο μονοπάτι ο χρήστης θα ερωτηθεί εάν το σύμπτωμα “phone has unresponsive screen” είναι αληθές (true) ή ψευδές (false). Στην περίπτωση που είναι ψευδής το πρόγραμμα οδηγείτε στην επόμενη υπόθεση. Αντίθετα εάν είναι αληθής θα οδηγηθεί στο επόμενο σύμπτωμα “problem persist after restart” Επιπλέων γίνεται η τελευταία ερώτηση στο χρήστη όπου στην περίπτωση που δοθεί ψευδής (false) απάντηση το πρόγραμμα οδηγείτε στην επόμενη υπόθεση, διαφορετικά σε περίπτωση που δοθεί αληθής (true) εμφανίζει στον χρήστη τη διάγνωση πιθανής βλάβης «software_glitch».

Στο δεύτερο μονοπάτι θα γίνει η τελευταία ερώτηση στο χρήστη εάν το σύμπτωμα “problem persist after restart” είναι αληθές (true) ή ψευδές (false). Στην περίπτωση που δοθεί ψευδής (false) απάντηση το πρόγραμμα οδηγείτε στην επόμενη υπόθεση, διαφορετικά σε περίπτωση που δοθεί αληθής (true) εμφανίζει στον χρήστη τη διάγνωση πιθανής βλάβης «software_glitch».

Η εξής υλοποίηση του παραπάνω διαγράμματος βασίζεται στο παρακάτω τμήμα κώδικα της εφαρμογής:

```
software_glitch :-(verify('phone begins to malfunction ');
                   verify('phone has unrepsonsive screen ')),
                   not(verify('problem persist after restart ')).
```



Εικόνα 30 Υπόθεση βλάβης stock camera app problem

Στην παραπάνω εικόνα 30 αναλύουμε τα βήματα με τα οποία εκτελείτε ο κώδικας, για την υπόθεση «stock camera app problem» με την χρήση διαγράμματος δέντρου αποφάσεων.

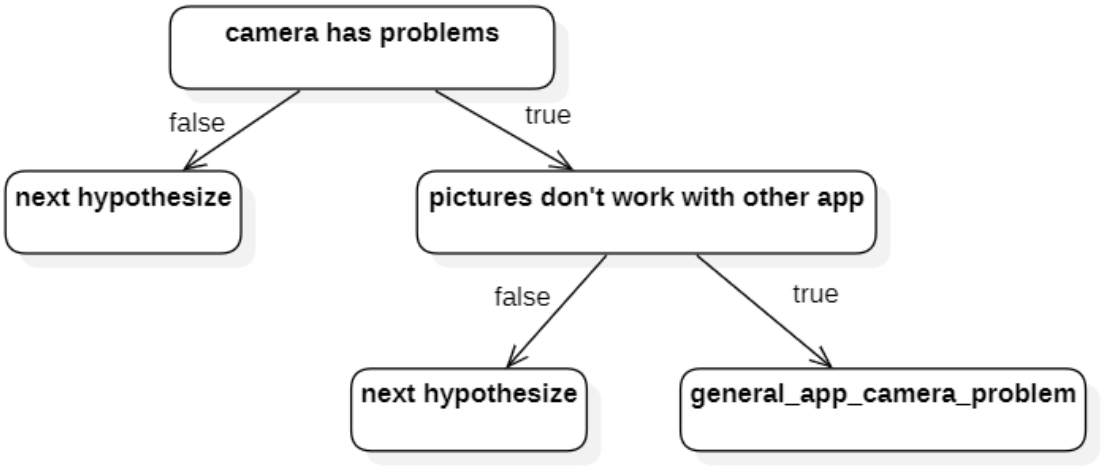
Αρχικά εξετάζεται το πρώτο σύμπτωμα “camera has problems” ρωτώντας τον χρήστη εάν είναι αληθές ή ψευδές. Στην περίπτωση που ο χρήστης δώσει αρνητική απάντηση (false) το πρόγραμμα οδηγείτε στην επόμενη υπόθεση (next hypothesize), διαφορετικά σε περίπτωση που ο χρήστης δώσει θετική απάντηση (true) το πρόγραμμα οδηγείτε στο επόμενο σύμπτωμα “pictures taken is black”.

Έπειτα γίνεται η τελευταία ερώτηση στο χρήστη όπου στην περίπτωση που δοθεί ψευδής (false) απάντηση το πρόγραμμα οδηγείτε στην επόμενη υπόθεση, διαφορετικά σε περίπτωση που δοθεί αληθής (true) εμφανίζει στον χρήστη τη διάγνωση πιθανής βλάβης «stock camera app problem».

Η εξής υλοποίηση του παραπάνω διαγράμματος βασίζεται στο παρακάτω τμήμα κώδικα της εφαρμογής:

stock_camera_app_problem :-camera_problems,

verify('pictures taken is black ').



Εικόνα 31 Υπόθεση βλάβης general app camera problem

Στην παραπάνω εικόνα 31 αναλύουμε τα βήματα με τα οποία εκτελείτε ο κώδικας, για την υπόθεση «general camera app problem» με την χρήση διαγράμματος δέντρου αποφάσεων.

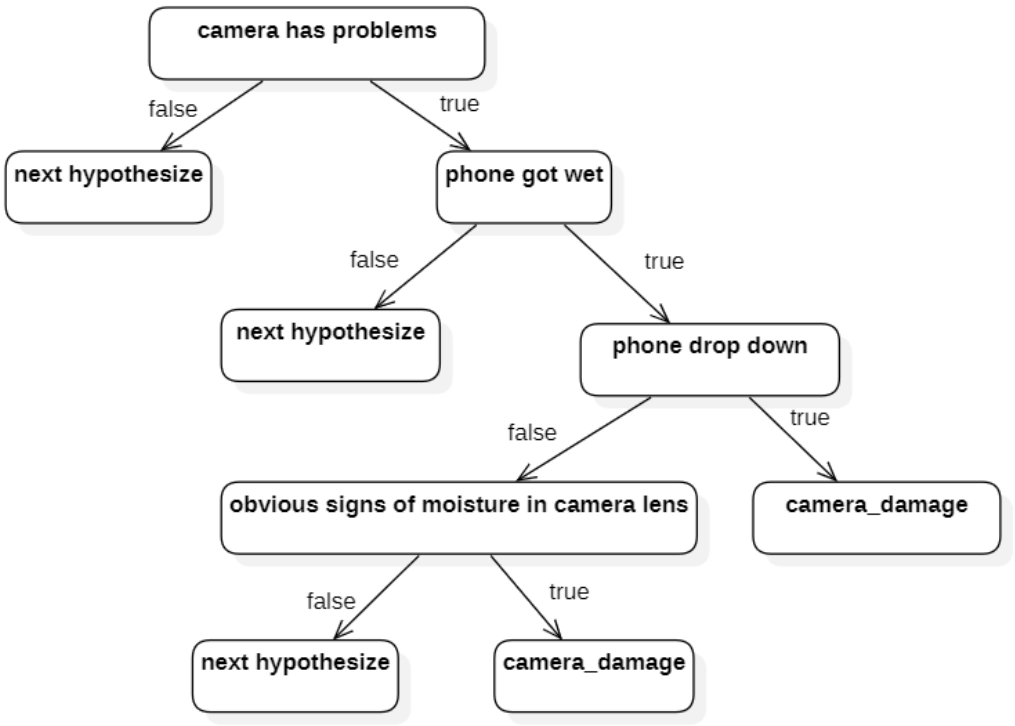
Αρχικά εξετάζεται το πρώτο σύμπτωμα “camera has problems” ρωτώντας τον χρήστη εάν είναι αληθές ή ψευδές. Στην περίπτωση που ο χρήστης δώσει αρνητική απάντηση (false) το πρόγραμμα οδηγείτε στην επόμενη υπόθεση (next hypothesize), διαφορετικά σε περίπτωση που ο χρήστης δώσει θετική απάντηση (true) το πρόγραμμα οδηγείτε στο επόμενο σύμπτωμα “pictures don’t work with other app”.

Έπειτα γίνεται η τελευταία ερώτηση στο χρήστη όπου στην περίπτωση που δοθεί ψευδής (false) απάντηση το πρόγραμμα οδηγείτε στην επόμενη υπόθεση, διαφορετικά σε περίπτωση που δοθεί αληθής (true) εμφανίζει στον χρήστη τη διάγνωση πιθανής βλάβης «general camera app problem».

Η εξής υλοποίηση του παραπάνω διαγράμματος βασίζεται στο παρακάτω τμήμα κώδικα της εφαρμογής:

general_app_camera_problem :-camera_problems,

not(verify('pictures work with other app ')).



Εικόνα 32 Υπόθεση βλάβης camera damage

Στην παραπάνω εικόνα 32 αναλύουμε τα βήματα με τα οποία εκτελείτε ο κώδικας, για την υπόθεση «camera damage» με την χρήση διαγράμματος δέντρου αποφάσεων.

Αρχικά εξετάζεται το πρώτο σύμπτωμα “camera has problems” ρωτώντας τον χρήστη εάν είναι αληθές ή ψευδές. Στην περίπτωση που ο χρήστης δώσει αρνητική απάντηση (false) το πρόγραμμα οδηγείτε στην επόμενη υπόθεση, διαφορετικά σε περίπτωση που ο χρήστης δώσει θετική απάντηση (true) το πρόγραμμα οδηγείτε στο επόμενο σύμπτωμα “phone got wet”.

Έπειτα γίνεται ξανά ερώτηση στο χρήστη όπου στην περίπτωση που δοθεί ψευδής (false) απάντηση το πρόγραμμα οδηγείτε στην επόμενη υπόθεση, αντίθετα σε περίπτωση που δοθεί αληθής (true) απάντηση το πρόγραμμα οδηγείτε στο επόμενο σύμπτωμα “phone drop down”. Στο σημείο αυτό ο χρήστης θα έχει επιλέξει ένα από τα δύο μονοπάτια του δέντρου, συγκεκριμένα μεταξύ του συμπτώματος “obvious signs of moisture in camera lens” και της πιθανής βλάβης «camera damage».

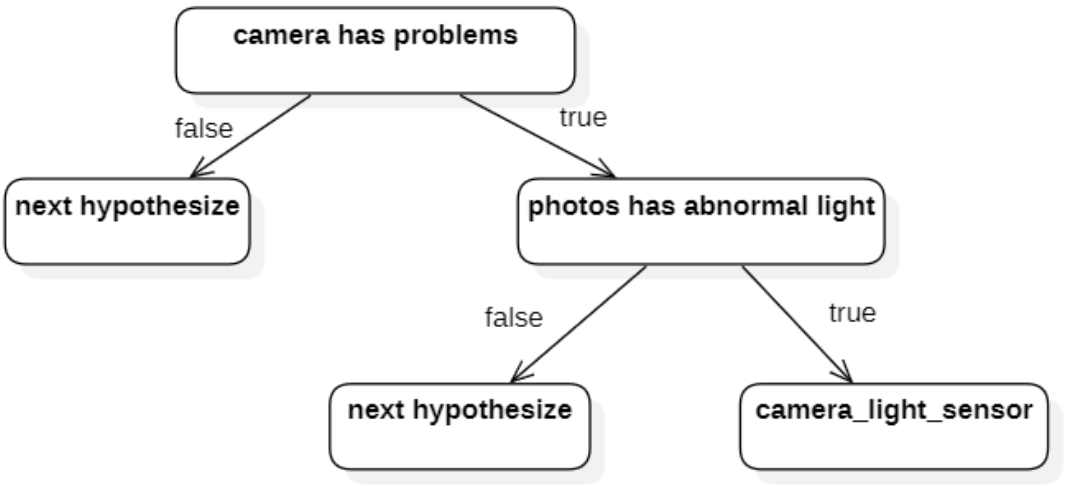
Στο πρώτο μονοπάτι γίνεται η τελευταία ερώτηση στο χρήστη εάν το σύμπτωμα «obvious signs of moisture in camera lens» είναι αληθές (true) ή ψευδές (false). Στην

περίπτωση που είναι ψευδής το πρόγραμμα οδηγείτε στην επόμενη υπόθεση. Αντίθετα εάν είναι αληθής θα οδηγηθεί στη διάγνωση πιθανής βλάβης «camera damage».

Στο δεύτερο μονοπάτι το πρόγραμμα θα εμφανίσει στον χρήστη τη διάγνωση πιθανής βλάβης «camera damage».

Η εξής υλοποίηση του παραπάνω διαγράμματος βασίζεται στο παρακάτω τμήμα κώδικα της εφαρμογής:

```
camera_damage :-camera_problems,
                wet_phone ,
               (verify('phone drop down ');
                verify('obvious signs of moisture in camera lens ')).
```



Εικόνα 33 Υπόθεση βλάβης camera light sensor

Στην παραπάνω εικόνα 33 αναλύουμε τα βήματα με τα οποία εκτελείτε ο κώδικας, για την υπόθεση «camera light sensor» με την χρήση διαγράμματος δέντρου αποφάσεων.

Αρχικά εξετάζεται το πρώτο σύμπτωμα “camera has problems” ρωτώντας τον χρήστη εάν είναι αληθές ή ψευδές. Στην περίπτωση που ο χρήστης δώσει αρνητική απάντηση (false) το πρόγραμμα οδηγείτε στην επόμενη υπόθεση (next hypothesize),

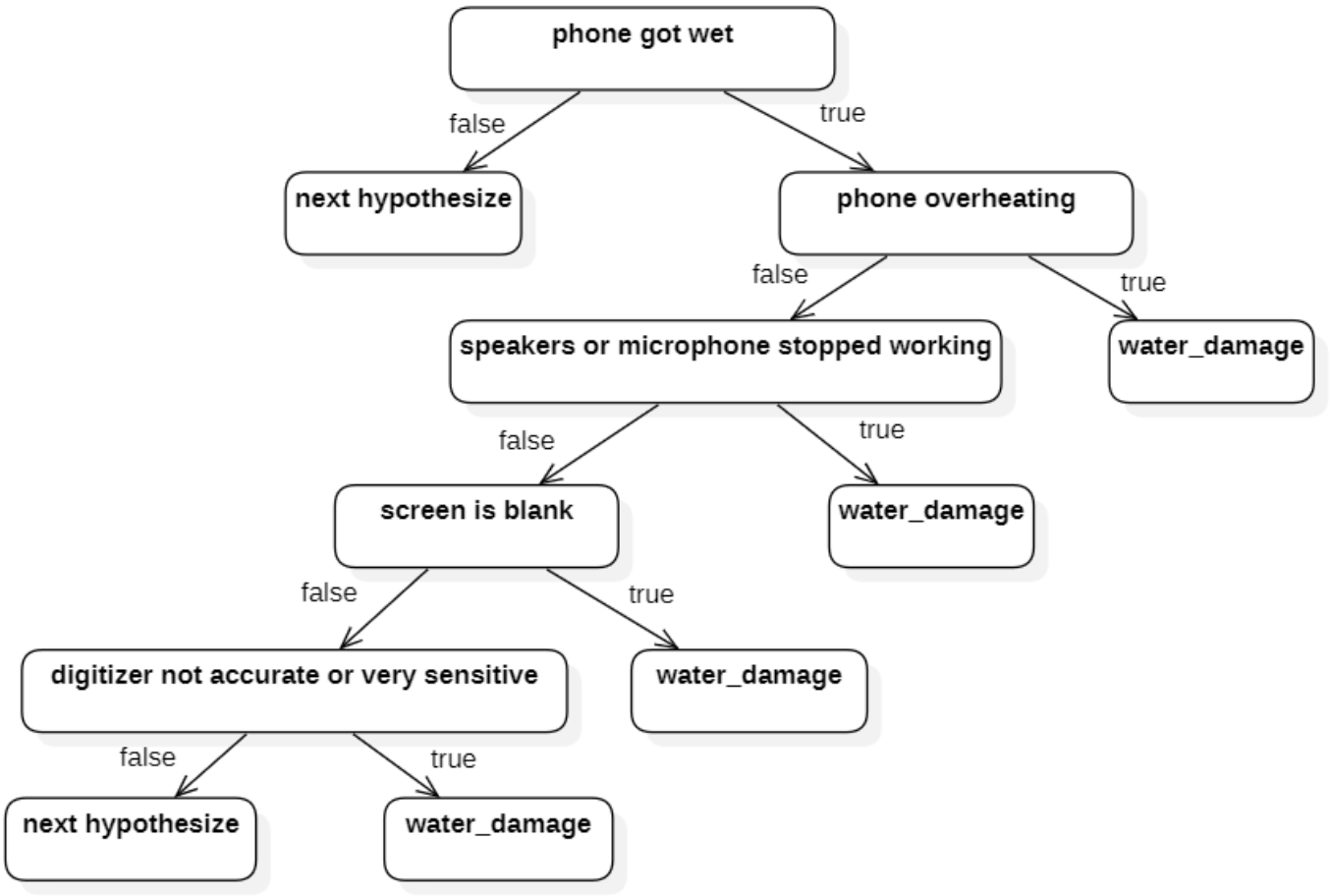
διαφορετικά σε περίπτωση που ο χρήστης δώσει θετική απάντηση (true) το πρόγραμμα οδηγείτε στο επόμενο σύμπτωμα “photos has abnormal light”.

Έπειτα γίνεται η τελευταία ερώτηση στο χρήστη όπου στην περίπτωση που δοθεί ψευδής (false) απάντηση το πρόγραμμα οδηγείτε στην επόμενη υπόθεση, διαφορετικά σε περίπτωση που δοθεί αληθής (true) εμφανίζει στον χρήστη τη διάγνωση πιθανής βλάβης «camera light sensor».

Η εξής υλοποίηση του παραπάνω διαγράμματος βασίζεται στο παρακάτω τμήμα κώδικα της εφαρμογής:

camera_light_sensor :-camera_problems,

verify('photos has abnormal light ').



Εικόνα 34 Υπόθεση βλάβης water damage

Στην παραπάνω εικόνα 34 αναλύουμε τα βήματα με τα οποία εκτελείτε ο κώδικας, για την υπόθεση «water damage» με την χρήση διαγράμματος δέντρου αποφάσεων.

Αρχικά εξετάζεται το πρώτο σύμπτωμα “phone got wet” ρωτώντας τον χρήστη εάν είναι αληθές ή ψευδές. Στην περίπτωση που ο χρήστης δώσει αρνητική απάντηση (false) το πρόγραμμα οδηγείτε στην επόμενη υπόθεση (next hypothesize), διαφορετικά σε περίπτωση που ο χρήστης δώσει θετική απάντηση (true) το πρόγραμμα οδηγείτε στο επόμενο σύμπτωμα “phone overheating”.

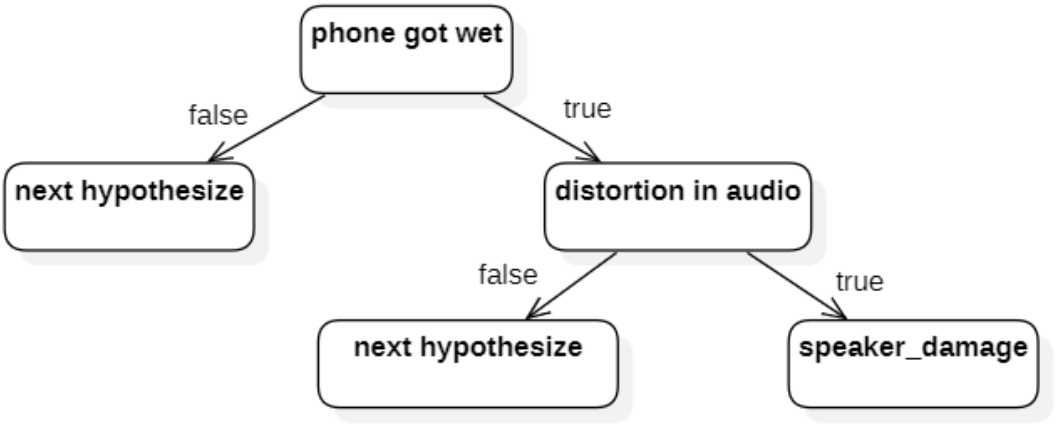
Έπειτα γίνεται ξανά ερώτηση στο χρήστη όπου στην περίπτωση που δοθεί ψευδής (false) απάντηση το πρόγραμμα οδηγείτε στο επόμενο σύμπτωμα “speakers or microphone stopped working”. Αντίθετα σε περίπτωση που δοθεί αληθής (true) απάντηση το πρόγραμμα εμφανίζει στον χρήστη τη διάγνωση πιθανής βλάβης «water damage».

Στη συνέχεια γίνεται ξανά ερώτηση στο χρήστη όπου στην περίπτωση που δοθεί ψευδής (false) απάντηση το πρόγραμμα οδηγείτε στο επόμενο σύμπτωμα “screen is black”. Αντίθετα σε περίπτωση που δοθεί αληθής (true) απάντηση το πρόγραμμα εμφανίζει στον χρήστη τη διάγνωση πιθανής βλάβης «water damage».

Επίσης γίνεται ξανά ερώτηση στο χρήστη όπου στην περίπτωση που δοθεί ψευδής (false) απάντηση το πρόγραμμα οδηγείτε στο επόμενο σύμπτωμα “digitizer not accurate or very sensitive”. Αντίθετα σε περίπτωση που δοθεί αληθής (true) απάντηση το πρόγραμμα εμφανίζει στον χρήστη τη διάγνωση πιθανής βλάβης «water damage». Τέλος γίνεται η τελευταία ερώτηση στο χρήστη όπου στην περίπτωση που δοθεί ψευδής (false) απάντηση το πρόγραμμα οδηγείτε στην επόμενη υπόθεση, διαφορετικά σε περίπτωση που δοθεί αληθής (true) εμφανίζει στον χρήστη τη διάγνωση πιθανής βλάβης «water damage».

Η εξής υλοποίηση του παραπάνω διαγράμματος βασίζεται στο παρακάτω τμήμα κώδικα της εφαρμογής:

```
water_damage :-wet_phone,
                (verify('phone overheating ');
                 verify('speakers or microphone stopped working ');
                 verify('screen is blank ');
                 verify('digitizer not accurate or very sensitive ')).
```



Εικόνα 35 Υπόθεση βλάβης speaker damage

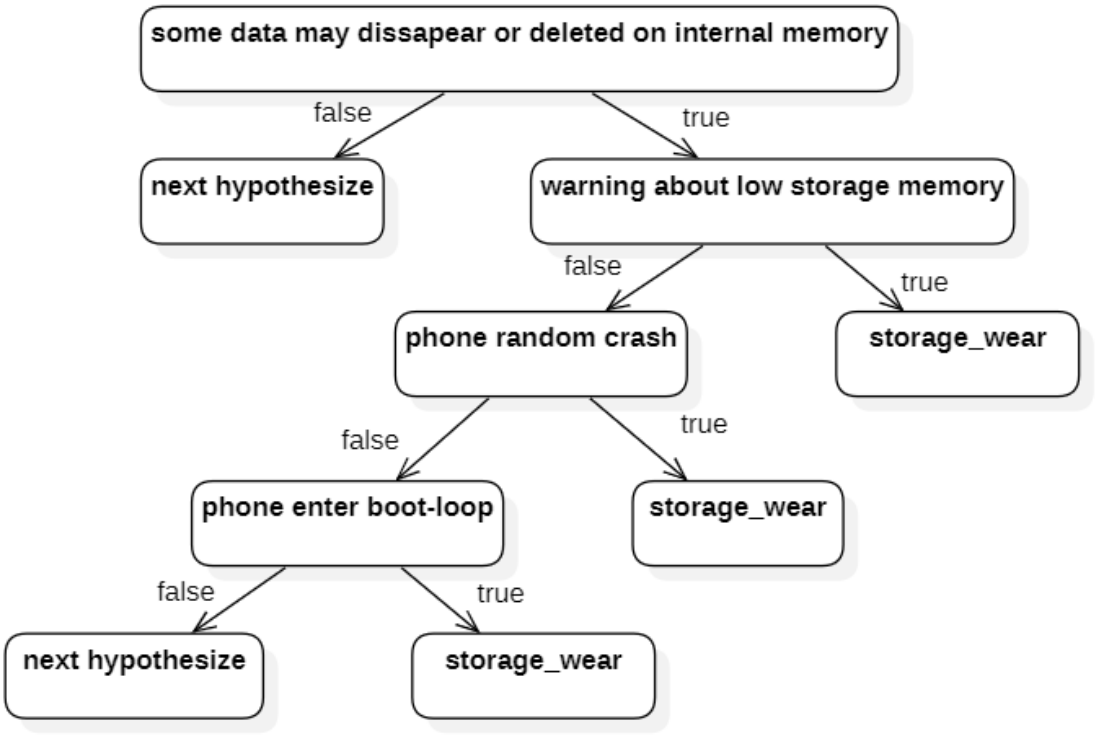
Στην παραπάνω εικόνα 35 αναλύουμε τα βήματα με τα οποία εκτελείτε ο κώδικας, για την υπόθεση «speaker damage» με την χρήση διαγράμματος δέντρου αποφάσεων.

Αρχικά εξετάζεται το πρώτο σύμπτωμα “phone got wet” ρωτώντας τον χρήστη εάν είναι αληθές ή ψευδές. Στην περίπτωση που ο χρήστης δώσει αρνητική απάντηση (false) το πρόγραμμα οδηγείτε στην επόμενη υπόθεση (next hypothesize), διαφορετικά σε περίπτωση που ο χρήστης δώσει θετική απάντηση (true) το πρόγραμμα οδηγείτε στο επόμενο σύμπτωμα “distortion in audio”.

Έπειτα γίνεται η τελευταία ερώτηση στο χρήστη όπου στην περίπτωση που δοθεί ψευδής (false) απάντηση το πρόγραμμα οδηγείτε στην επόμενη υπόθεση, διαφορετικά σε περίπτωση που δοθεί αληθής (true) εμφανίζει στον χρήστη τη διάγνωση πιθανής βλάβης «speaker damage».

Η εξής υλοποίηση του παραπάνω διαγράμματος βασίζεται στο παρακάτω τμήμα κώδικα της εφαρμογής:

```
speaker_damage      :-wet_phone,
                      verify('distortion in audio').
```



Εικόνα 36 Υπόθεση βλάβης storage wear

Στην παραπάνω εικόνα 36 αναλύουμε τα βήματα με τα οποία εκτελείτε ο κώδικας, για την υπόθεση «storage wear» με την χρήση διαγράμματος δέντρου αποφάσεων.

Αρχικά εξετάζεται το πρώτο σύμπτωμα “some data may disappear or deleted on internal memory” ρωτώντας τον χρήστη εάν είναι αληθές ή ψευδές. Στην περίπτωση που ο χρήστης δώσει αρνητική απάντηση (false) το πρόγραμμα οδηγείτε στην επόμενη υπόθεση (next hypothesize), διαφορετικά σε περίπτωση που ο χρήστης δώσει θετική απάντηση (true) το πρόγραμμα οδηγείτε στο επόμενο σύμπτωμα “warning about low storage memory”.

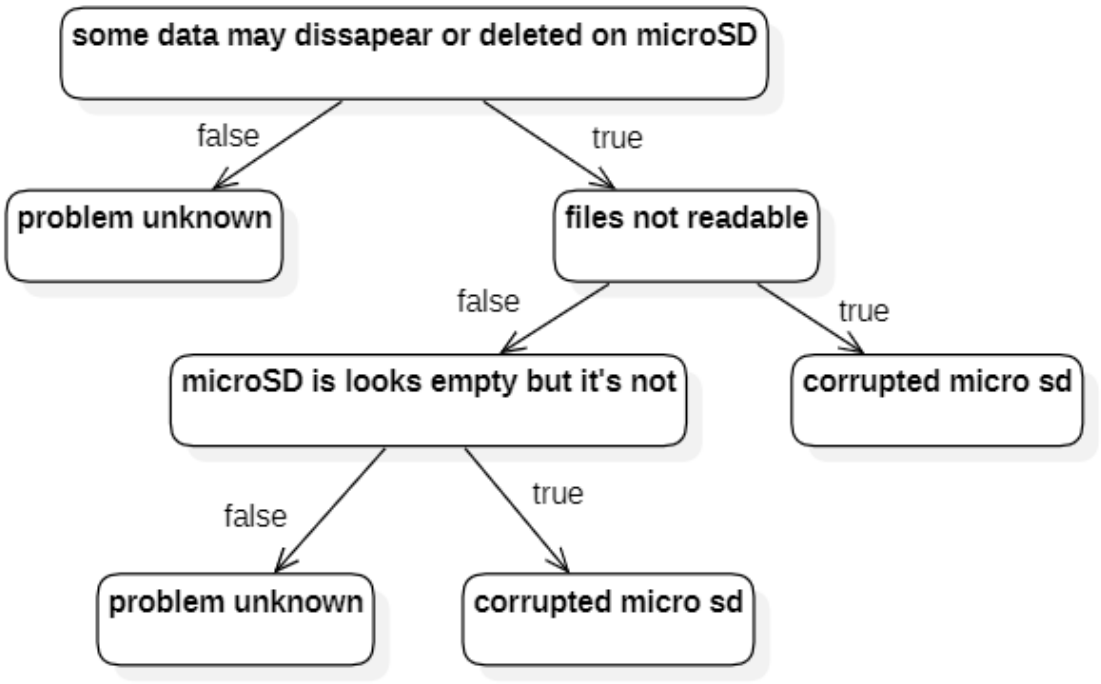
Έπειτα γίνεται ξανά ερώτηση στο χρήστη όπου στην περίπτωση που δοθεί ψευδής (false) απάντηση το πρόγραμμα οδηγείτε στο επόμενο σύμπτωμα “phone random crash”. Αντίθετα σε περίπτωση που δοθεί αληθής (true) απάντηση το πρόγραμμα εμφανίζει στον χρήστη τη διάγνωση πιθανής βλάβης «storage wear».

Στη συνέχεια γίνεται ξανά ερώτηση στο χρήστη όπου στην περίπτωση που δοθεί ψευδής (false) απάντηση το πρόγραμμα οδηγείτε στο επόμενο σύμπτωμα “phone enter boot-loop”. Αντίθετα σε περίπτωση που δοθεί αληθής (true) απάντηση το πρόγραμμα εμφανίζει στον χρήστη τη διάγνωση πιθανής βλάβης «storage wear».

Τέλος γίνεται η τελευταία ερώτηση στο χρήστη όπου στην περίπτωση που δοθεί ψευδής (false) απάντηση το πρόγραμμα οδηγείτε στην επόμενη υπόθεση, διαφορετικά σε περίπτωση που δοθεί αληθής (true) εμφανίζει στον χρήστη τη διάγνωση πιθανής βλάβης «storage wear».

Η εξής υλοποίηση του παραπάνω διαγράμματος βασίζεται στο παρακάτω τμήμα κώδικα της εφαρμογής:

```
storage_wear :-verify('some data may dissapear or deleted on internal memory
                '),
                 (verify('warning about low storage memory ');
                  verify('phone random crash ');
                  verify('phone enter boot-loop ')).
```



Εικόνα 37 Υπόθεση βλάβης corrupted micro sd

Στην παραπάνω εικόνα 37 αναλύουμε τα βήματα με τα οποία εκτελείτε ο κώδικας, για την υπόθεση «corrupted micro sd» με την χρήση διαγράμματος δέντρου αποφάσεων.

Αρχικά εξετάζεται το πρώτο σύμπτωμα “some data may disappear or deleted on microSD” ρωτώντας τον χρήστη εάν είναι αληθές ή ψευδές. Στην περίπτωση που ο χρήστης δώσει αρνητική απάντηση (false) το πρόγραμμα οδηγείτε στον τερματισμό εμφανίζοντας ότι η βλάβη δεν βρέθηκε (problem unknown), διαφορετικά σε περίπτωση που ο χρήστης δώσει θετική απάντηση (true) το πρόγραμμα οδηγείτε στο επόμενο σύμπτωμα ”files not readable”.

Έπειτα γίνεται ξανά ερώτηση στο χρήστη όπου στην περίπτωση που δοθεί ψευδής (false) απάντηση το πρόγραμμα οδηγείτε στο επόμενο σύμπτωμα “microSD is looks empty but its not”.

Τέλος γίνεται η τελευταία ερώτηση στο χρήστη όπου στην περίπτωση που δοθεί ψευδής (false) απάντηση το πρόγραμμα οδηγείτε στον τερματισμό εμφανίζοντας ότι η βλάβη δεν βρέθηκε (problem unknown) διαφορετικά σε περίπτωση που δοθεί αληθής (true) εμφανίζει στον χρήστη τη διάγνωση πιθανής βλάβης «corrupted micro sd».

Η εξής υλοποίηση του παραπάνω διαγράμματος βασίζεται στο παρακάτω τμήμα κώδικα της εφαρμογής:

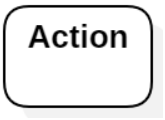
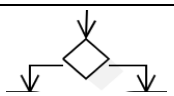
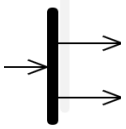
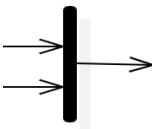
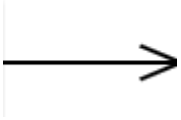
```
corrupted_micro_sd      :-verify('some data may dissapear or deleted on microSD '),
                          (verify('files not readable ');
                           verify('microSD is looks empty but it\'s not ')).
```

2.2 Διαγράμματα δραστηριοτήτων

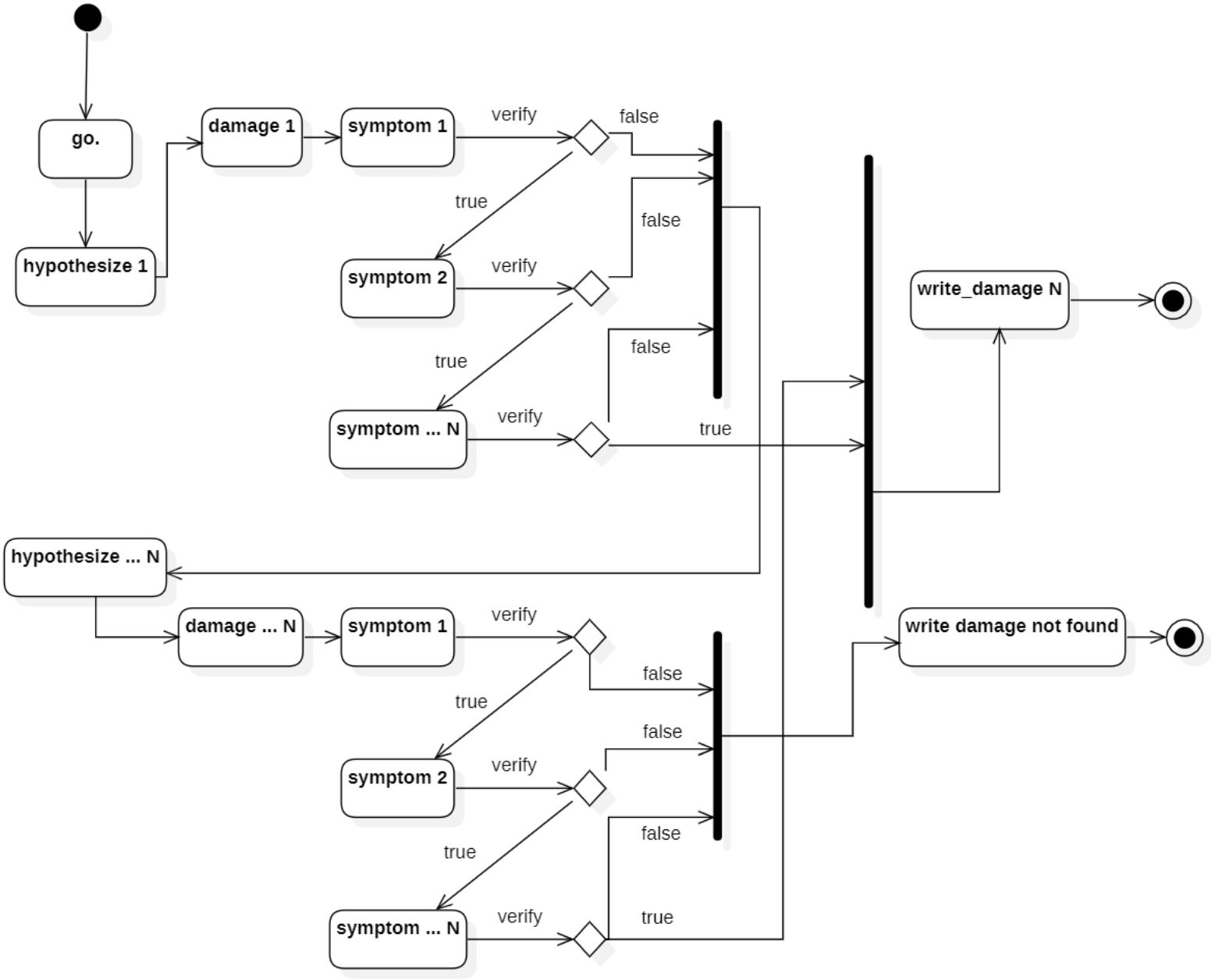
Τα Διαγράμματα δραστηριότητας (Activity diagrams) είναι γραφικές αναπαραστάσεις των ροών εργασίας (workflows) των σταδιακών δραστηριοτήτων και δράσεων που προορίζονται να μοντελοποιήσουν τόσο τις υπολογιστικές όσο και τις οργανωτικές διαδικασίες (π.χ. ροές εργασίας) [21].

Τα διαγράμματα δραστηριότητας είναι κατασκευασμένα από ένα περιορισμένο αριθμό σχημάτων, που συνδέονται με βέλη. Στον πίνακα 1 θα δούμε τα βασικά στοιχεία των διαγραμμάτων δραστηριοτήτων.

Πίνακας 1 Στοιχεία Διαγράμματος Δραστηριοτήτων

Αρχική Δραστηριότητα: Αναπαριστά το σημείο εκκίνησης ή την πρώτη δραστηριότητα μιας ροής	
Δραστηριότητα	
Αποφάσεις: Σημείο απόφασης που οδηγεί σε δύο ή περισσότερους δρόμους ανάλογα με την απόφαση	
Τελική Δραστηριότητα: Το τέλος ενός διαγράμματος δραστηριοτήτων	
Διακλάδωση (Fork)	
Ένωση (Join)	
Τα βέλη αντιπροσωπεύουν τη σειρά με την οποία συμβαίνουν οι δραστηριότητες	

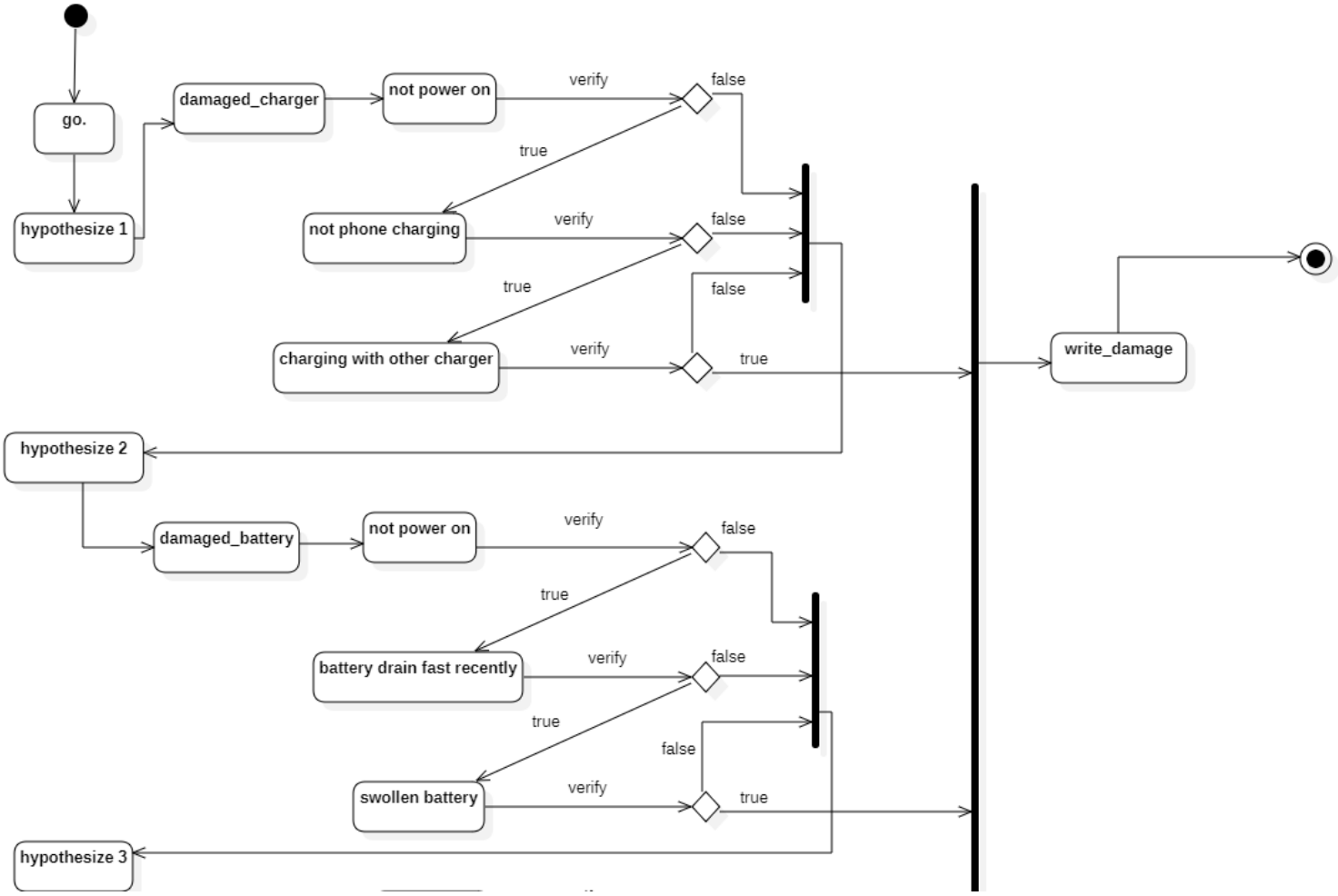
Στο παρακάτω πίνακα 1 θα δούμε τη βασική δομή των διαγραμμάτων δραστηριοτήτων της εφαρμογής μας καθώς και την ανάλυση των βημάτων.



Εικόνα 38 Βασική δομή διαγράμματος δραστηριοτήτων εφαρμογής διαγνωστικού κινητού τηλεφώνου

Όπως βλέπουμε Στην παραπάνω εικόνα 38 που είναι η γενική δομή της εφαρμογής υπάρχουν οι δραστηριότητες hypothesize, damage, symptom, write damage και write damage not found. Η έναρξη του προγράμματος πραγματοποιείτε από τον κανόνα go. όπου η είναι αρχική δραστηριότητα στο διάγραμμα δραστηριοτήτων και συμβολίζεται με την μαύρη βούλα. Στην συνέχεια ξεκινάει η διαδικασία της διάγνωσης με το πρώτο hypothesize το οποίο ελέγχει μια βλάβη κάνοντας διαδοχικές ερωτήσεις διερευνώντας τα πιθανά συμπτώματα της. Το κάθε σύμπτωμα αποτελεί μια ερώτηση. Στην πορεία εκτελείτε ο κανόνας επαλήθευσης (verify) στην ενέργεια απόφασης (ρόμβος) για κάθε

σύμπτωμα και ο χρήστης απαντάει στα συμπτώματα με αληθές (true) ή ψευδές (false). Συνήθως η αληθής απάντηση οδηγεί στην επόμενη ερώτηση συμπτώματος ή στην ολοκλήρωση του προγράμματος με εύρεση πιθανής βλάβης (write damage), ενώ η ψευδής απάντηση συνήθως οδηγεί σε ένωση (Join) όπου κατευθύνει το χρήστη σε επόμενη υπόθεση (hypothesize) δηλαδή πιθανή βλάβη (damage) ή στην ολοκλήρωση του προγράμματος με αποτυχία εύρεσης βλάβης (write damage not found)ς. Τέλος οι δύο αυτές περιπτώσεις ολοκλήρωσης προγράμματος οδηγούν στην τελική δραστηριότητα που συμβολίζεται με μαύρη βούλα σε κύκλο.



Εικόνα 39 Υποθέσεις βλαβών damage charger και damaged battery

Το πρόγραμμα ξεκινάει από την αρχική δραστηριότητα και περιμένει από το χρήστη να καλέσει τον κανόνα go για να αρχίσει ο έλεγχος των υποθέσεων (hypothesizes).

Στην υπόθεση 1 γίνεται διάγνωση της βλάβης «damage charger» με τα εξής συμπτώματα:

- not power on
- not phone charging
- does phone charging with other charger

Προκειμένου η υπόθεση «damage charger» να είναι αληθής και να οδηγηθεί στην ένωση (join) που καταλήγει στην εμφάνιση πιθανής βλάβης, θα πρέπει και τα τρία παραπάνω συμπτώματα να είναι αληθές.

Σε περίπτωση που κάποιο από τα συμπτώματα είναι ψευδές, το πρόγραμμα οδηγείτε στην ένωση (join) και κατά συνέπεια στον έλεγχο της επόμενης υπόθεσης (hypothesize).

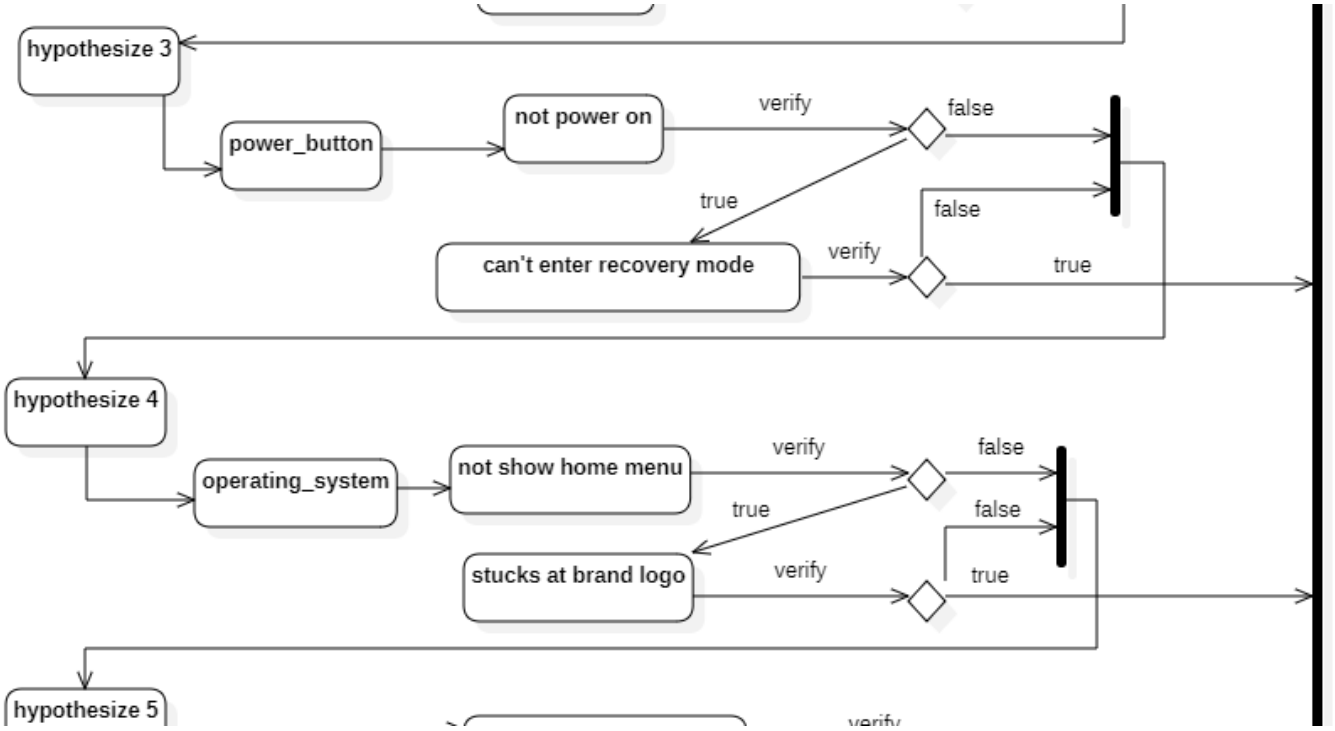
Στην υπόθεση 2 γίνεται διάγνωση της βλάβης «damage battery» με τα εξής συμπτώματα:

- not power on
- battery drain fast recently
- swollen battery

Προκειμένου η υπόθεση «damage battery» να είναι αληθής και να οδηγηθεί στην ένωση (join) που καταλήγει στην εμφάνιση πιθανής βλάβης, θα πρέπει και τα τρία παραπάνω συμπτώματα να είναι αληθές.

Σε περίπτωση που κάποιο από τα συμπτώματα είναι ψευδές, το πρόγραμμα οδηγείτε στην ένωση (join) και κατά συνέπεια στον έλεγχο της επόμενης υπόθεσης(hypothesize).

Το σύμπτωμα “not power on” είναι κοινό σε παραπάνω από μια υπόθεση, αυτό σημαίνει ότι το πρόγραμμα ρωτάει τον χρήστη μόνο την πρώτη φορά για την απάντηση του και στη συνέχεια σε επόμενες υποθέσεις αγνοεί την ερώτηση του παραπάνω συμπτώματος αφού γνωρίζει ήδη την απάντηση. Το παραπάνω ισχύει και για το σύμπτωμα battery “drain fast recently”.



Εικόνα 40 Υποθέσεις βλαβών power button και operating system

Στην υπόθεση 3 γίνεται διάγνωση της βλάβης «power button» με τα εξής συμπτώματα:

- not power on
- can't enter recovery mode

Προκειμένου η υπόθεση «power button» να είναι αληθής και να οδηγηθεί στην ένωση (join) που καταλήγει στην εμφάνιση πιθανής βλάβης, θα πρέπει και τα δυο παραπάνω συμπτώματα να είναι αληθές.

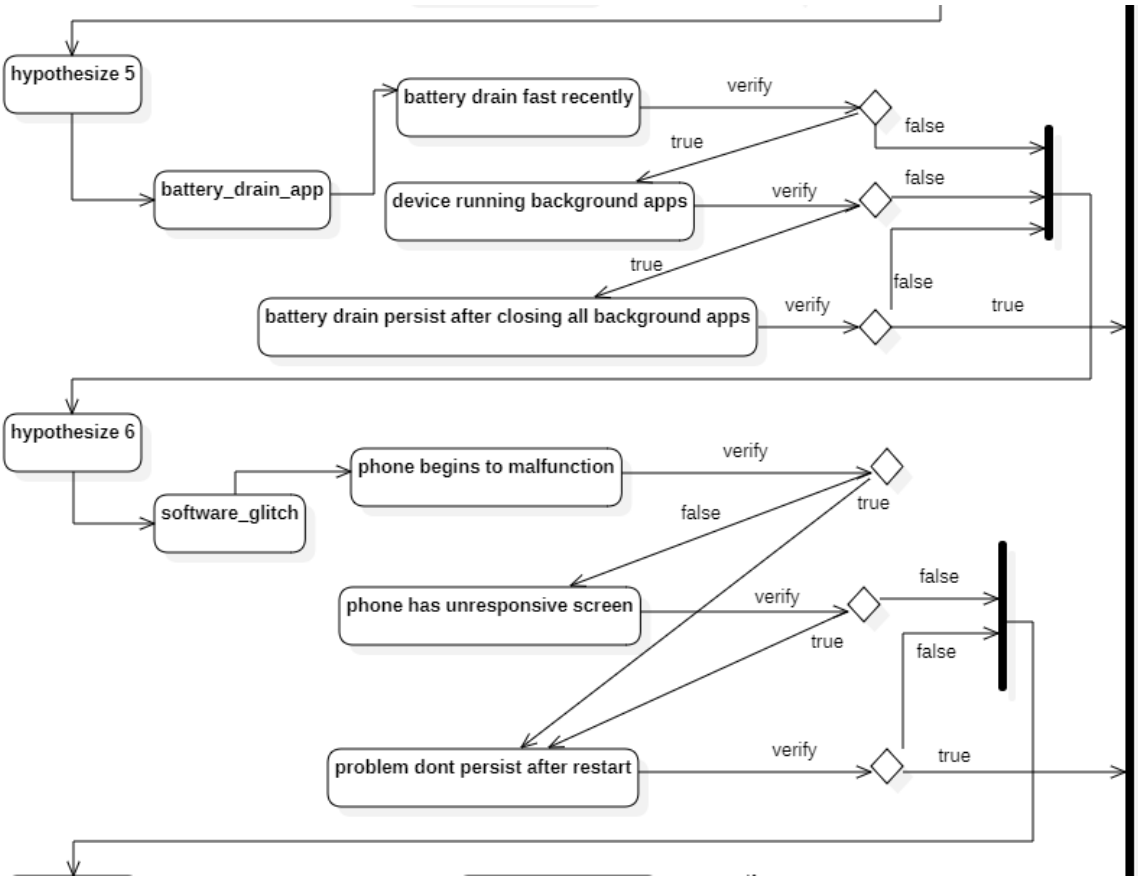
Σε περίπτωση που κάποιο από τα συμπτώματα είναι ψευδές, το πρόγραμμα οδηγείτε στην ένωση (join) και κατά συνέπεια στον έλεγχο της επόμενης υπόθεσης(hypothesize).

Στην υπόθεση 4 γίνεται διάγνωση της βλάβης «operating system» με τα εξής συμπτώματα:

- not show home menu
- stucks at brand logo

Προκειμένου η υπόθεση «operating system» να είναι αληθής και να οδηγηθεί στην ένωση (join) που καταλήγει στην εμφάνιση πιθανής βλάβης, θα πρέπει και τα δυο

παραπάνω συμπτώματα να είναι αληθές. Σε περίπτωση που κάποιο από τα συμπτώματα είναι ψευδές, το πρόγραμμα οδηγείτε στην ένωση (join) και κατά συνέπεια στον έλεγχο της επόμενης υπόθεσης(hypothesize).



Εικόνα 41 Υποθέσεις βλαβών battery drain app και software glitch

Στην υπόθεση 5 γίνεται διάγνωση της βλάβης «battery drain app» με τα εξής συμπτώματα:

- battery drain fast recently
- device running background apps
- battery drain persist after closing all background apps

Προκειμένου η υπόθεση «battery drain app» να είναι αληθής και να οδηγηθεί στην ένωση (join) που καταλήγει στην εμφάνιση πιθανής βλάβης, θα πρέπει και τα δυο παραπάνω συμπτώματα να είναι αληθές.

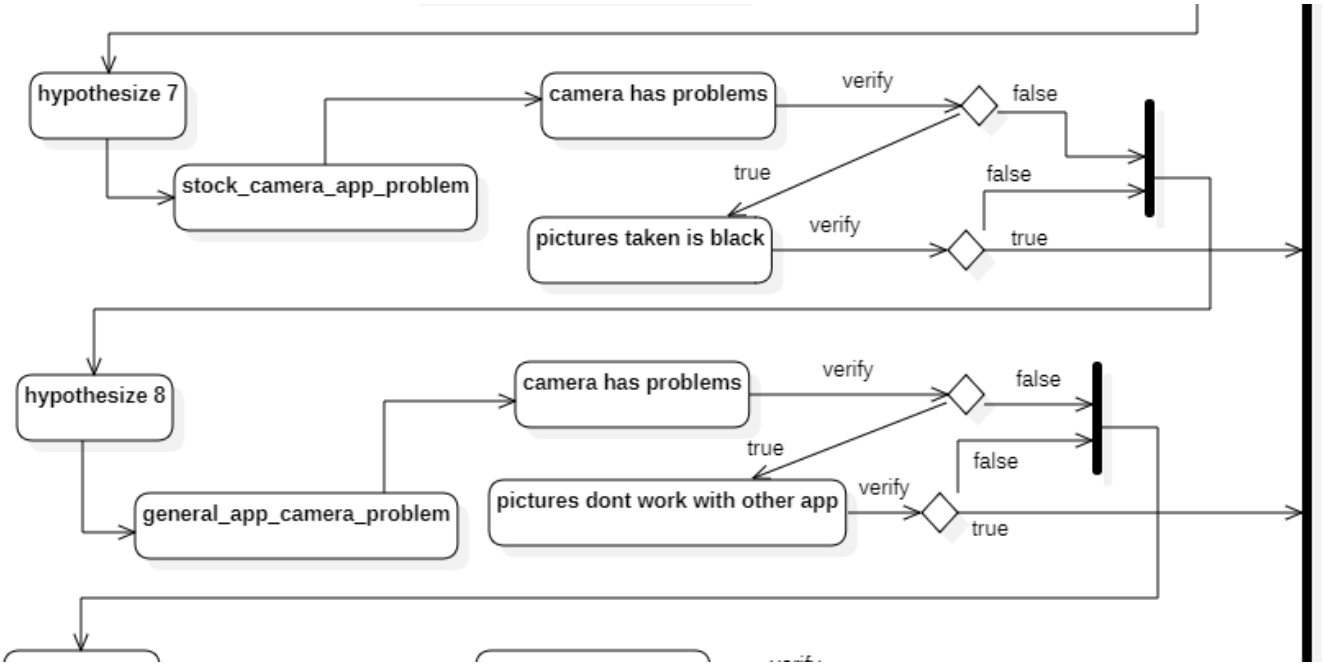
Σε περίπτωση που κάποιο από τα συμπτώματα είναι ψευδές, το πρόγραμμα οδηγείτε στην ένωση (join) και κατά συνέπεια στον έλεγχο της επόμενης υπόθεσης(hypothesize).

Στην υπόθεση 5 γίνεται διάγνωση της βλάβης «battery drain app» με τα εξής συμπτώματα:

- phone begins to malfunction
- phone has unresponsive screen
- problem don’t persist after restart

Προκειμένου η υπόθεση «battery drain app» να είναι αληθής και να οδηγηθεί στην ένωση (join) που καταλήγει στην εμφάνιση πιθανής βλάβης, θα πρέπει το σύμπτωμα “phone begins to malfunction” ή το “phone has unresponsive screen” σε συνδυασμό με το σύμπτωμα “problem don’t persist after restart “ να είναι αληθές.

Σε περίπτωση που ένα από τα δύο πρώτα ή το τρίτο σύμπτωμα είναι ψευδές, το πρόγραμμα οδηγείτε στην ένωση (join) και κατά συνέπεια στον έλεγχο της επόμενης υπόθεσης(hypothesize).



Εικόνα 42 Υποθέσεις βλαβών stock camera app problem και general app camera problem

Στην υπόθεση 7 γίνεται διάγνωση της βλάβης «power button» με τα εξής συμπτώματα:

- camera has problems
- picture taken is black

Προκειμένου η υπόθεση «stock camera app problem» να είναι αληθής και να οδηγηθεί στην ένωση (join) που καταλήγει στην εμφάνιση πιθανής βλάβης, θα πρέπει και τα δυο παραπάνω συμπτώματα να είναι αληθές.

Σε περίπτωση που κάποιο από τα συμπτώματα είναι ψευδές, το πρόγραμμα οδηγείτε στην ένωση (join) και κατά συνέπεια στον έλεγχο της επόμενης υπόθεσης(hypothesize).

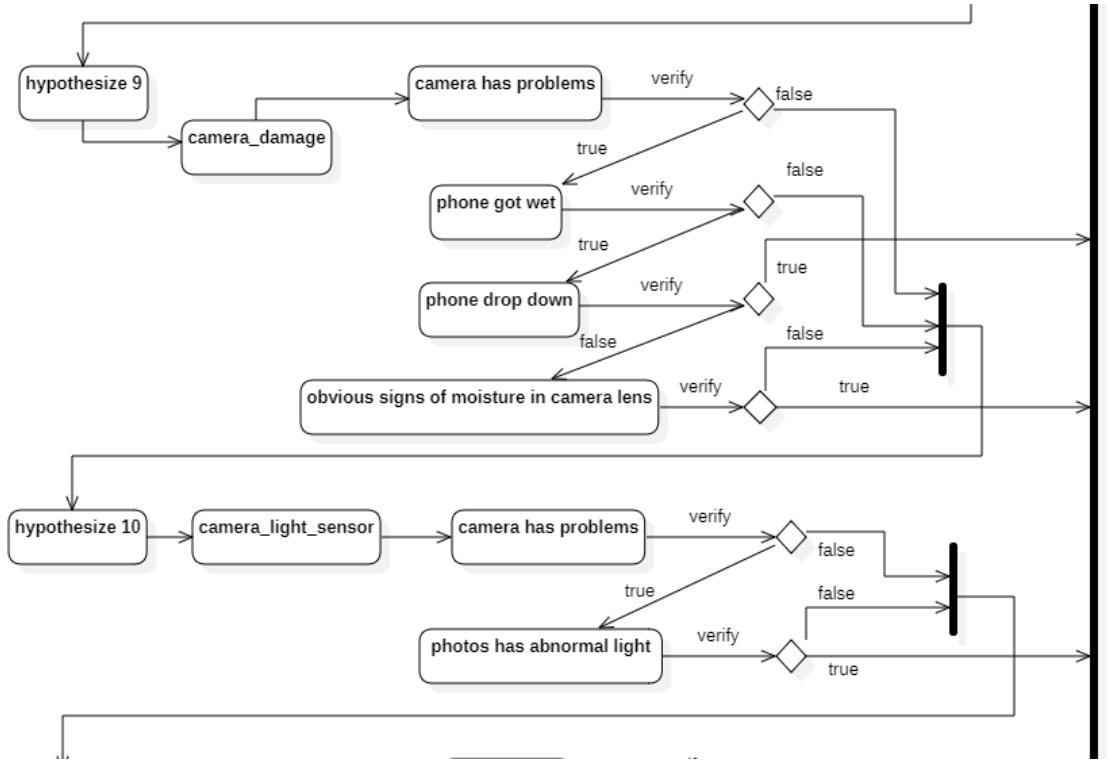
Στην υπόθεση 8 γίνεται διάγνωση της βλάβης «operating system» με τα εξής συμπτώματα:

- camera has problems
- pictures don’t work with other app

Προκειμένου η υπόθεση «operating system» να είναι αληθής και να οδηγηθεί στην ένωση (join) που καταλήγει στην εμφάνιση πιθανής βλάβης, θα πρέπει και τα δυο παραπάνω συμπτώματα να είναι αληθές.

Σε περίπτωση που κάποιο από τα συμπτώματα είναι ψευδές, το πρόγραμμα οδηγείτε στην ένωση (join) και κατά συνέπεια στον έλεγχο της επόμενης υπόθεσης(hypothesize).

Το σύμπτωμα “camera has problems” είναι κοινό σε παραπάνω από μια υπόθεση, αυτό σημαίνει ότι το πρόγραμμα ρωτάει τον χρήστη μόνο την πρώτη φορά για την απάντηση του και στη συνέχεια σε επόμενες υποθέσεις αγνοεί την ερώτηση του παραπάνω συμπτώματος αφού γνωρίζει ήδη την απάντηση.



Εικόνα 43 Υποθέσεις βλαβών camera damage και camera light sensor

Στην υπόθεση 9 γίνεται διάγνωση της βλάβης «power button» με τα εξής συμπτώματα:

- camera has problems
- phone got wet
- phone drop down
- obvious signs of moisture in camera lens

Προκειμένου η υπόθεση «camera damage» να είναι αληθής και να οδηγηθεί στην ένωση (join) που καταλήγει στην εμφάνιση πιθανής βλάβης, θα πρέπει συμπτώματα “camera has problems” και “phone got wet” σε συνδυασμό με το σύμπτωμα "phone drop down” ή “obvious signs of moisture in camera lens” να είναι αληθές.

Σε περίπτωση που ένα από τα δύο πρώτα ή και τα δύο τελευταία συμπτώματα είναι ψευδές, το πρόγραμμα οδηγείτε στην ένωση (join) και κατά συνέπεια στον έλεγχο της επόμενης υπόθεσης(hypothesize).

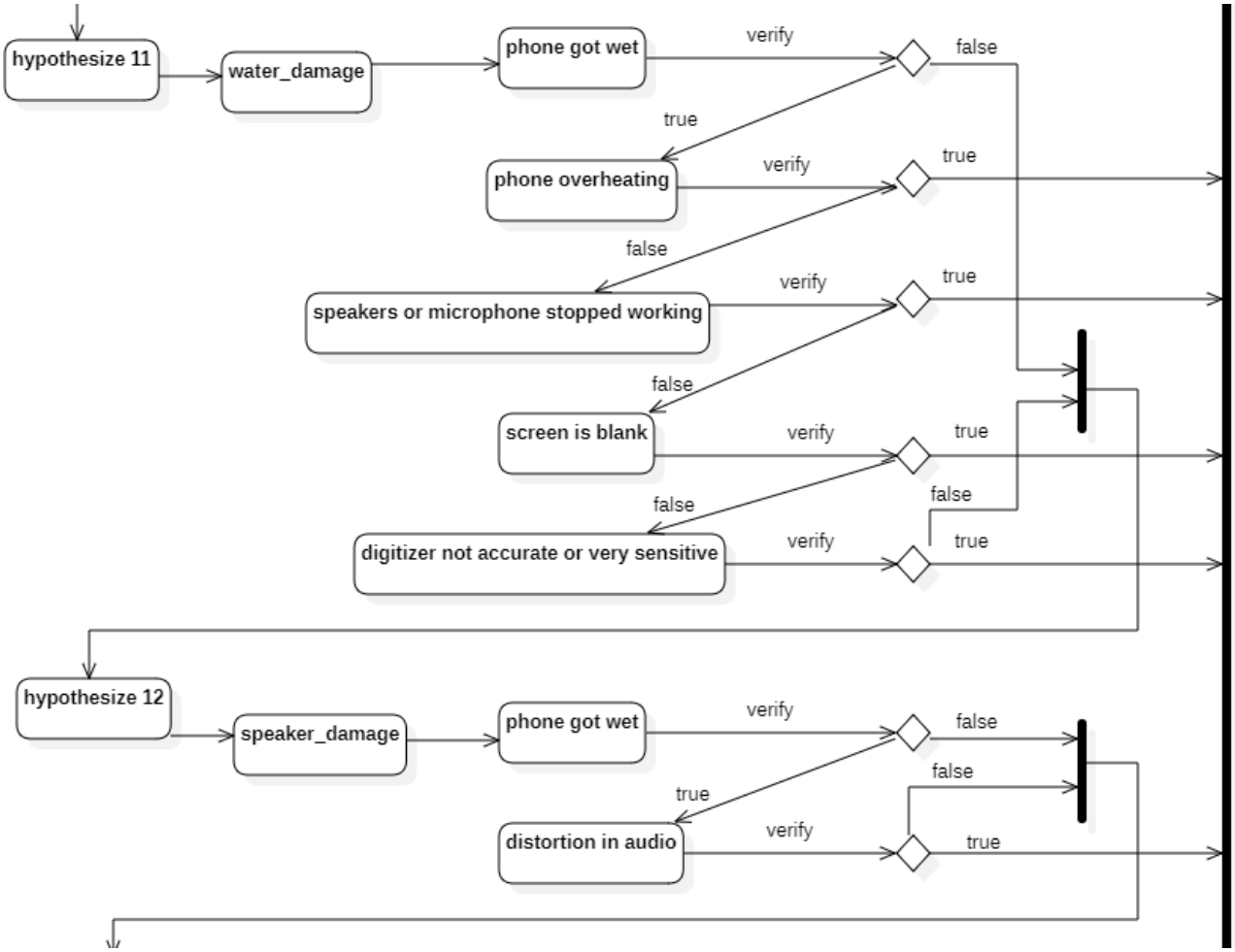
Στην υπόθεση 10 γίνεται διάγνωση της βλάβης «camera_light_sensor» με τα εξής συμπτώματα:

- camera has problems
- photos has abnormal light

Προκειμένου η υπόθεση «camera_light_sensor» να είναι αληθής και να οδηγηθεί στην ένωση (join) που καταλήγει στην εμφάνιση πιθανής βλάβης, θα πρέπει και τα δυο παραπάνω συμπτώματα να είναι αληθές.

Σε περίπτωση που κάποιο από τα συμπτώματα είναι ψευδές, το πρόγραμμα οδηγείτε στην ένωση (join) και κατά συνέπεια στον έλεγχο της επόμενης υπόθεσης(hypothesize).

Το σύμπτωμα “phone got wet” είναι κοινό σε παραπάνω από μια υπόθεση, αυτό σημαίνει ότι το πρόγραμμα ρωτάει τον χρήστη μόνο την πρώτη φορά για την απάντηση του και στη συνέχεια σε επόμενες υποθέσεις αγνοεί την ερώτηση του παραπάνω συμπτώματος αφού γνωρίζει ήδη την απάντηση.



Εικόνα 44 Υποθέσεις βλαβών water damage και speaker damage

Στην υπόθεση 11 γίνεται διάγνωση της βλάβης «water damage» με τα εξής συμπτώματα:

- phone got wet
- phone overheating
- speakers or microphone stopped working
- screen is blank
- digitizer not accurate or very sensitive

Προκειμένου η υπόθεση «water damage» να είναι αληθής και να οδηγηθεί στην ένωση (join) που καταλήγει στην εμφάνιση πιθανής βλάβης, θα πρέπει το σύμπτωμα «phone got wet» σε συνδυασμό με ένα από τα υπόλοιπα τέσσερα να είναι αληθές.

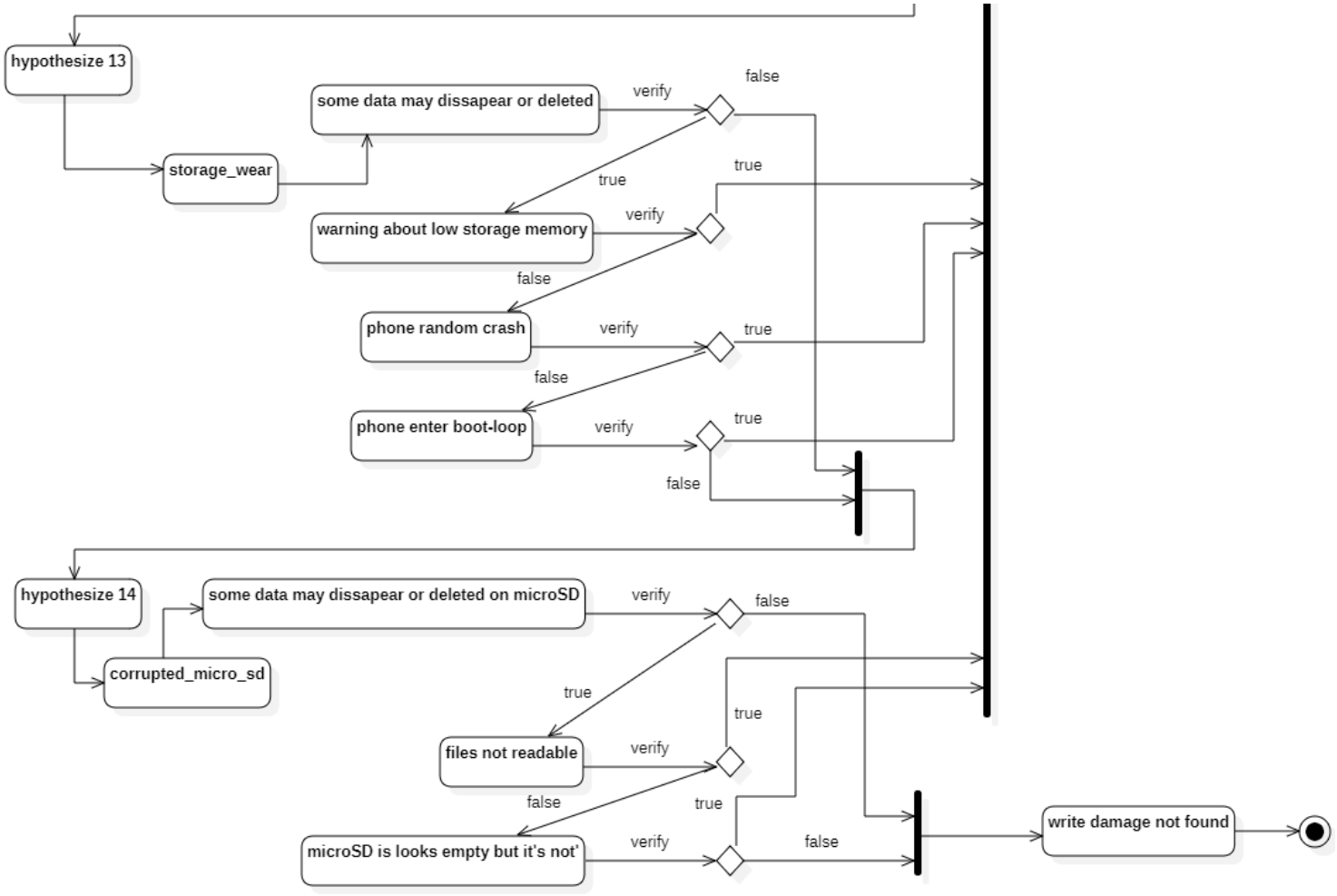
Σε περίπτωση που το πρώτο ή όλα τα υπόλοιπα συμπτώματα είναι ψευδές, το πρόγραμμα οδηγείτε στην ένωση (join) και κατά συνέπεια στον έλεγχο της επόμενης υπόθεσης(hypothesize).

Στην υπόθεση 12 γίνεται διάγνωση της βλάβης «speaker damage» με τα εξής συμπτώματα:

- phone got wet
- distortion in audio

Προκειμένου η υπόθεση «speaker damage» να είναι αληθής και να οδηγηθεί στην ένωση (join) που καταλήγει στην εμφάνιση πιθανής βλάβης, θα πρέπει και τα δυο παραπάνω συμπτώματα να είναι αληθές.

Σε περίπτωση που κάποιο από τα συμπτώματα είναι ψευδές, το πρόγραμμα οδηγείτε στην ένωση (join) και κατά συνέπεια στον έλεγχο της επόμενης υπόθεσης(hypothesize).



Εικόνα 45 Υποθέσεις βλαβώvs storage wear και corrupted micro sd

Στην υπόθεση 13 γίνεται διάγνωση της βλάβης «storage wear» με τα εξής συμπτώματα:

- some data may dissapear or deleted
- warning about low storage memory
- phone random crash
- phone enter boot-loop

Προκειμένου η υπόθεση « storage wear» να είναι αληθής και να οδηγηθεί στην ένωση (join) που καταλήγει στην εμφάνιση πιθανής βλάβης, θα πρέπει το σύμπτωμα “some data may disappear” ή “deleted on internal memory” σε συνδυασμό με ένα από τα υπόλοιπα να είναι αληθές.

Σε περίπτωση που το πρώτο ή όλα τα υπόλοιπα είναι ψευδές, το πρόγραμμα οδηγείτε στην ένωση (join) και κατά συνέπεια στον έλεγχο της επόμενης υπόθεσης(hypothesize).

Στην υπόθεση 14 γίνεται διάγνωση της βλάβης «corrupted_micro_sd» με τα εξής συμπτώματα:

- some data may dissapear or deleted on microSD
- files not readable
- microSD is looks empty but it's not

Προκειμένου η υπόθεση «corrupted_micro_sd» να είναι αληθής και να οδηγηθεί στην ένωση (join) που καταλήγει στην εμφάνιση πιθανής βλάβης, θα πρέπει το σύμπτωμα “some data may disappear” ή “deleted on microSD” σε συνδυασμό με ένα από τα υπόλοιπα να είναι αληθές.

Σε περίπτωση που το πρώτο ή όλα τα υπόλοιπα είναι ψευδές, το πρόγραμμα οδηγείτε στην ένωση (join) και καταλήγει στην τελική δραστηριότητα «write damage not found» όπου δεν υπάρχει εύρεση πιθανής βλάβης.

ΚΕΦΑΛΑΙΟ 3Ο: Ανάλυση Και Παρουσίαση Εφαρμογής

Η σχεδίαση της διαγνωστικής εφαρμογής πραγματοποιήθηκε στη γλώσσα προγραμματισμού Prolog και συγκεκριμένα στο περιβάλλον SWI-Prolog. Τα κύρια μέρη της εφαρμογής είναι οι κανόνες go, hypothesize, ask, verify και undo.

3.1 Κανόνας go

```
go :- hypothesize(Damage),
      write('I think the damage could be: '),
      write(Damage),
      nl,
      undo.
```

Η έναρξη του διαγνωστικού γίνεται με τον κανόνα go. Η πρώτη ενέργεια που εκτελείται είναι ο κανόνας hypothesize με την μεταβλητή Damage όπου θα ελέγξει όλες τις πιθανές υποθέσεις της εφαρμογής που έχει η γνωσιακή μας βάση. Αφού καταλήξει σε ένα συμπέρασμα βλάβης τότε το εμφανίζει στο χρήστη μέσω της εντολής write. Τέλος καλείται η εντολή undo όπου αναίρει της απαντήσεις που έδωσε ο χρήστης στα ερωτήματα συμπτωμάτων έτσι ώστε το πρόγραμμα να είναι έτοιμο για επανεκτέλεση.

3.2 Κανόνας hypothesize

```
hypothesize(damaged_charger)                 :- damaged_charger, !.
hypothesize(damaged_battery)                 :- damaged_battery, !.
hypothesize(power_button)                    :- power_button, !.
hypothesize(operating_system)                :- operating_system, !.
hypothesize(battery_drain_app)               :- battery_drain_app, !.
hypothesize(software_glitch)                 :- software_glitch, !.
hypothesize(stock_camera_app_problem)        :- stock_camera_app_problem, !.
hypothesize(general_app_camera_problem):- general_app_camera_problem, !.
hypothesize(camera_damage)                   :- camera_damage, !.
hypothesize(camera_light_sensor)             :- camera_light_sensor, !.
hypothesize(water_damage)                    :-water_damage, !.
hypothesize(speaker_damage)                  :-speaker_damage, !.
```

```
hypothesize(storage_wear)              :-storage_wear, !.
hypothesize(corrupted_micro_sd)        :-corrupted_micro_sd, !.
```

Η ενότητα των hypothesize περιλαμβάνει όλες τις πιθανές περιπτώσεις βλαβών οι οποίες εκτελούνται διαδοχικά. Στο τέλος κάθε υπόθεσης υπάρχει το σύμβολο (!) που συμβολίζει την αποκοπή. Η αποκοπή βεβαιώνει ότι σε περίπτωση που βρεθεί κάποια βλάβη τότε θα γίνει κλάδεμα από το δέντρο των υπόλοιπων υποθέσεων και δεν θα κάνει περιττές ερωτήσεις στο χρήστη.

3.3 Κανόνες αναγνώρισης ζημιών

```
damaged_charger :-not(powerup),
                 not(verify('phone charging ')),
                 verify('does phone charging with other charger ').

damaged_battery :-not(powerup),
                  battery_drain,
                  verify('swollen battery ').

power_button    :-not(powerup),
                  not(verify('can you enter recovery mode ')).

operating_system :-not(verify('show home menu ')),
                   verify('stucks at brand logo ').

battery_drain_app :-battery_drain,
                 verify('device running background apps '),
                 not(verify('battery drain persist after closing all background
apps ')).

software_glitch  :-(verify('phone begins to malfunction ');
                 verify('phone has unrepsonsive screen ')) ,!,
                 not(verify('problem persist after restart ')).

stock_camera_app_problem :-camera_problems,
                  verify('pictures taken is black ').

general_app_camera_problem :-camera_problems,
                      not(verify('pictures work with other app ')).

camera_damage    :-camera_problems,
                 wet_phone ,
```

	(verify('phone drop down '); verify('obvious signs of moisture in camera lens ')).
camera_light_sensor	:-camera_problems, verify('photos has abnormal light ').
water_damage	:-wet_phone, (verify('phone overheating '); verify('speakers or microphone stopped working '); verify('screen is blank '); verify('digitizer not accurate or very sensitive ')).
speaker_damage	:-wet_phone, verify('distortion in audio').
storage_wear	:-verify('some data may dissapear or deleted on internal memory '), (verify('warning about low storage memory '); verify('phone random crash '); verify('phone enter boot-loop ')).
corrupted_micro_sd	:-verify('some data may dissapear or deleted on microSD '), (verify('files not readable '); verify('microSD is looks empty but it\'s not ')).

Οι κανόνες αναγνώρισης ζημιών περιέχουν όλες τις πιθανές βλάβες όπως επίσης και τα συμπτώματα της κάθε μιας. Τα συμπτώματα ελέγχονται με την χρήση του κανόνα verify όπου παίρνει σαν μεταβλητή το ερώτημα μεταξύ παρενθέσεων. Μερικά από τα συμπτώματα περιέχουν τη λέξη not όπου αντιστρέφει την απάντηση του χρήστη στα σημεία που απαιτείται για να κάνει την υπόθεση αληθή.

3.4 Κοινοί κανόνες συμπτωμάτων

powerup	:- verify('phone power on '),!.
battery_drain	:- verify('battery drain fast recently '),!.
camera_problems	:- verify('camera has problems '),!.
wet_phone	:- verify('phone got wet '),!.

Στην ενότητα κοινοί κανόνες συμπτωμάτων τοποθετούνται ερωτήματα σε μορφή κανόνων τα οποία εμφανίζονται σε παραπάνω από μια βλάβη.

3.5 Κανόνας verify

```
verify(S) :-
  (yes(S)
   ->
   true ;
   (no(S)
    ->
    fail ;
    ask(S))).
```

Ο κανόνας verify περιέχει την μεταβλητή (S) η οποία παίρνει ως τιμή το κάθε σύμπτωμα κάθε υπόθεσης. Εκτελείται για κάθε ερώτημα στο χρήστη και ο σκοπός του είναι να κάνει αντιστοίχιση της απάντησης yes με true και της απάντησης no με false. Τέλος καλεί τον κανόνα ask με την ίδια μεταβλητή (S).

3.6 Κανόνας ask

```
ask(Question) :-
   write('Does the damage have the following symptom: '),
   write(Question),
   write('? '),
   read(Response),
   nl,
   ( (Response == yes ; Response == y)
     ->
      assert(yes(Question)) ;
      assert(no(Question)), fail).
```

Ο κανόνας ask καλείται στο τέλος του κανόνα verify και ο σκοπός του είναι να εμφανίσει το ολοκληρωμένο ερώτημα στον χρήστη με τη χρήση του write. Στη συνέχεια διαβάζει την απάντηση yes ή no που έδωσε ο χρήστης. Τέλος γίνεται ο ανάλογος ισχυρισμός σε κάθε ερώτηση έτσι ώστε να εξασφάλιση ότι το κάθε σύμπτωμα θα ερωτηθεί μια φορά.

3.7 Κανόνας undo

```
undo :- retract(yes(_)),fail.
undo :- retract(no(_)),fail.
undo.
```

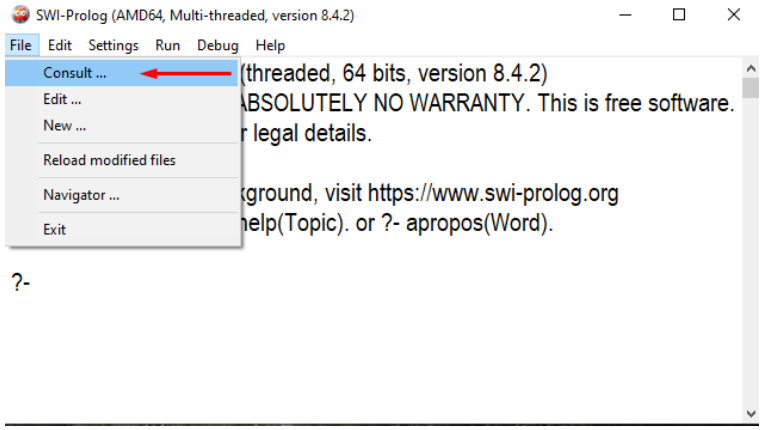
Ο κανόνας undo εκτελείται στο τέλος του κανόνα go δηλαδή μόλις ολοκληρωθεί η διάγνωση έτσι ώστε να αναιρεθούν όλες οι αναθέσεις που έγιναν από το κανόνα ask.

3.8 Εκτέλεση του κώδικα με παραδείγματα.

Για την εκτέλεση της διαγνωστικής εφαρμογής πρέπει να υπάρχει αποθηκευμένο αρχείο με κατάληξη .pl και να περιέχει τον κώδικα από το παράρτημα. Έπειτα πρέπει να γίνει consult στο περιβάλλον της SWI-Prolog με τα εξής βήματα.

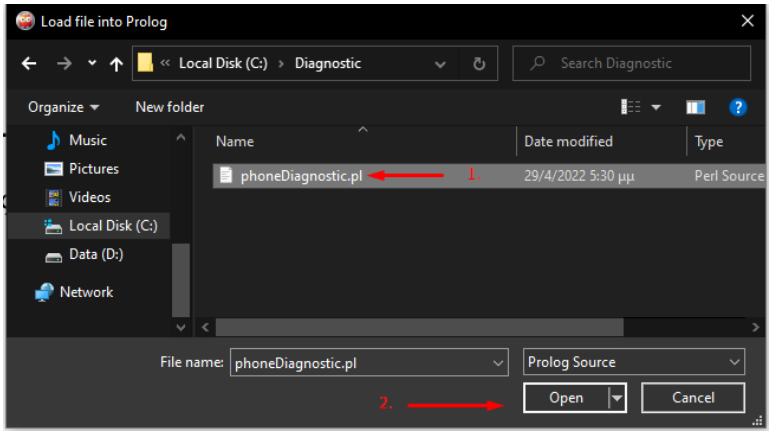
3.8.1 Βήματα εκτέλεσης της εφαρμογής

Στο 1ο βήμα ο χρήστης ανοίγει το SWI-Prolog και επιλέγει File -> Consult.



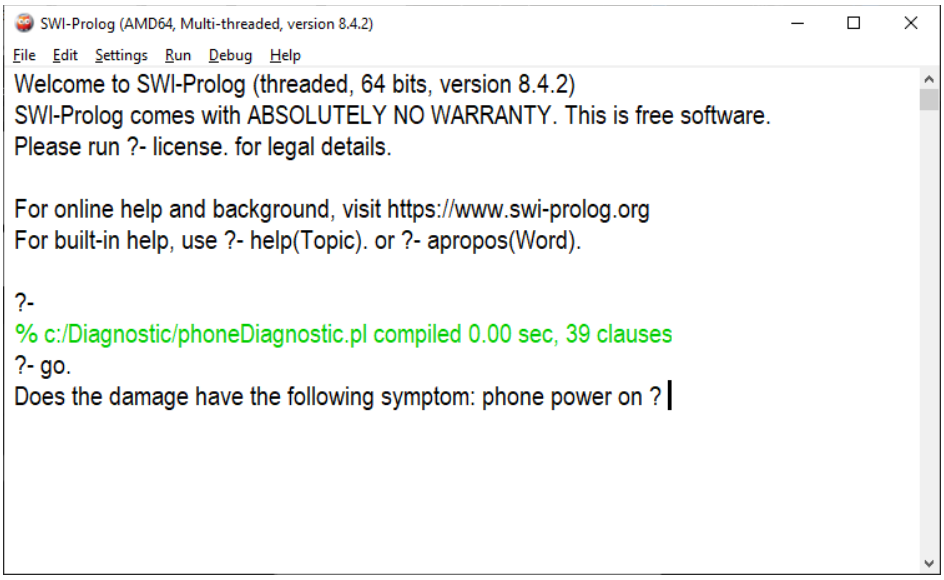
Εικόνα 46 Βήμα 1ο εκτέλεσης εφαρμογής

Στο 2ο βήμα ο χρήστης επιλέγει το αρχείο και το ανοίγει.



Εικόνα 47 Βήμα 2ο εκτέλεσης εφαρμογής

Στο 3ο βήμα ο χρήστης πρέπει να εκτελέσει το ερώτημα go. για να ξεκινήσουν οι ερωτήσεις των συμπτωμάτων.



Εικόνα 48 Βήμα 3ο εκτέλεσης εφαρμογής

3.8.2 Παραδείγματα εκτέλεσης εφαρμογής

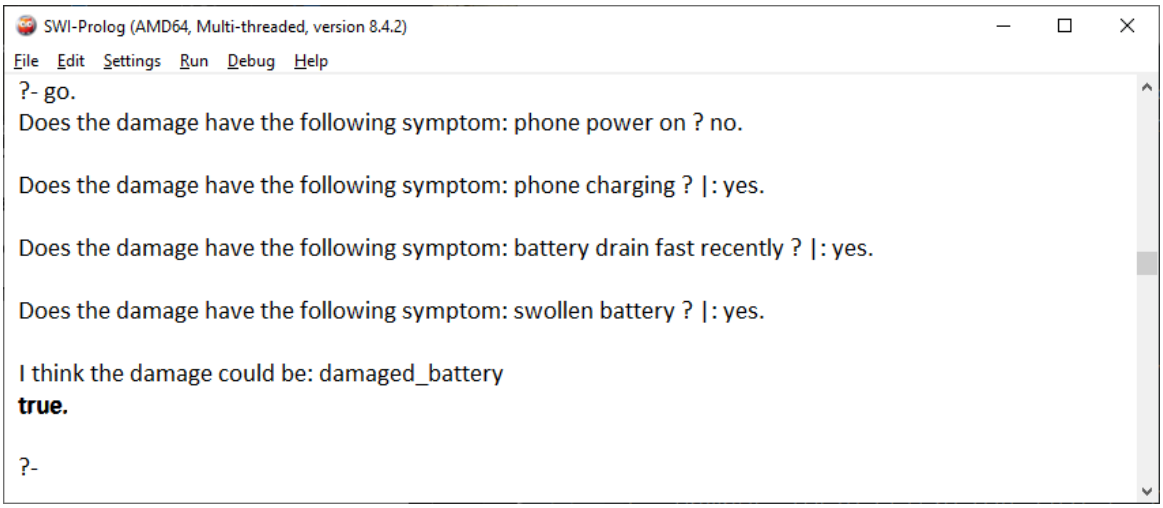
Στο 1ο παράδειγμα θα δούμε την περίπτωση του «damaged battery» όπου ο χρήστης θα έχει απαντήσει στα εξής ερωτήματα τις απαντήσεις του πίνακα 2. Για να

ισχύει η πιθανή βλάβη «damaged battery» στα σημεία του κώδικα που περιέχει not πριν από το verify θα πρέπει να δοθεί απάντηση no.

Πίνακας 2 Συμπτώματα για πιθανή βλάβη damage battery

Ερώτημα	Απάντηση
phone power on	no
phone charging	yes
battery drain fast recently	yes
swollen battery	yes

Στην παρακάτω εικόνα βλέπουμε την εκτέλεση του παραδείγματος «damaged battery».



Εικόνα 49 Εκτέλεση υπόθεσης damaged battery

Παρατηρήσεις 1ου παραδείγματος:

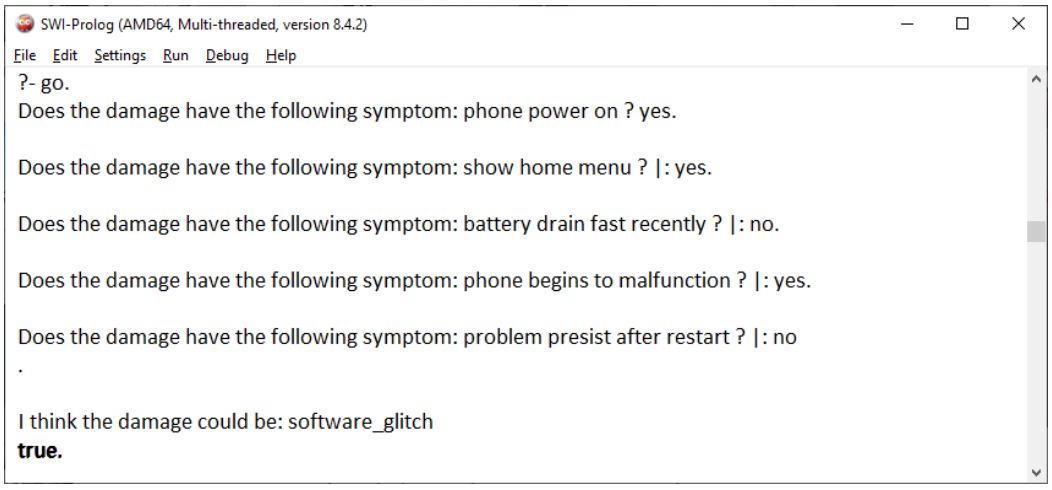
1. Βλέπουμε ότι οι υποθέσεις «damaged_charger» και «damaged_battery» έχουν κοινό σύμπτωμα το “phone power on” και ερώτηση γίνεται μόνο μια φορά την πρώτη φορά που το συναντάμε.
2. Η απάντηση yes στο ερώτημα «phone charging» κάνει την υπόθεση «damage charger» ψευδές γιατί υπάρχει το not πριν από την ερώτηση, οπότε γίνεται αποκοπή της υπόθεσης και εκτελούνται τα ερωτήματα της επόμενης υπόθεσης «damaged battery».

Στο 2ο παράδειγμα θα δούμε δύο περιπτώσεις του «software glitch». Στην πρώτη περίπτωση ο χρήστης θα έχει απαντήσει στα εξής ερωτήματα τις απαντήσεις του πίνακα 3. Για να ισχύει η πιθανή βλάβη «software glitch» στα σημεία του κώδικα που περιέχει not πριν από το verify θα πρέπει να δοθεί απάντηση no. Σε αυτό το παράδειγμα επιλέχθηκε η σύντομη διαδρομή ερωτήσεών.

Πίνακας 3 Συμπτώματα για πιθανή βλάβη software glitch (Περίπτωση 1η)

Ερώτημα	**Απάντηση**
phone power on	yes
show home menu	yes
battery drain fast recently	no
phone begins to malfunction	yes
problem persist after restart	no

Στην παρακάτω εικόνα βλέπουμε την εκτέλεση του παραδείγματος 1ης περίπτωσης «software glitch».



Εικόνα 50 Εκτέλεση υπόθεσης software glitch (Περίπτωση 1η)

Παρατηρήσεις 2ου παραδείγματος 1ης περίπτωσης:

1. Εφόσον δώσαμε στην ερώτηση phone power on την απάντηση yes, γίνεται αποκοπή των υποθέσεων «damaged charger», «damaged battery» και «power button» γιατί περιέχουν σαν πρώτο σύμπτωμα το ίδιο ερώτημα.

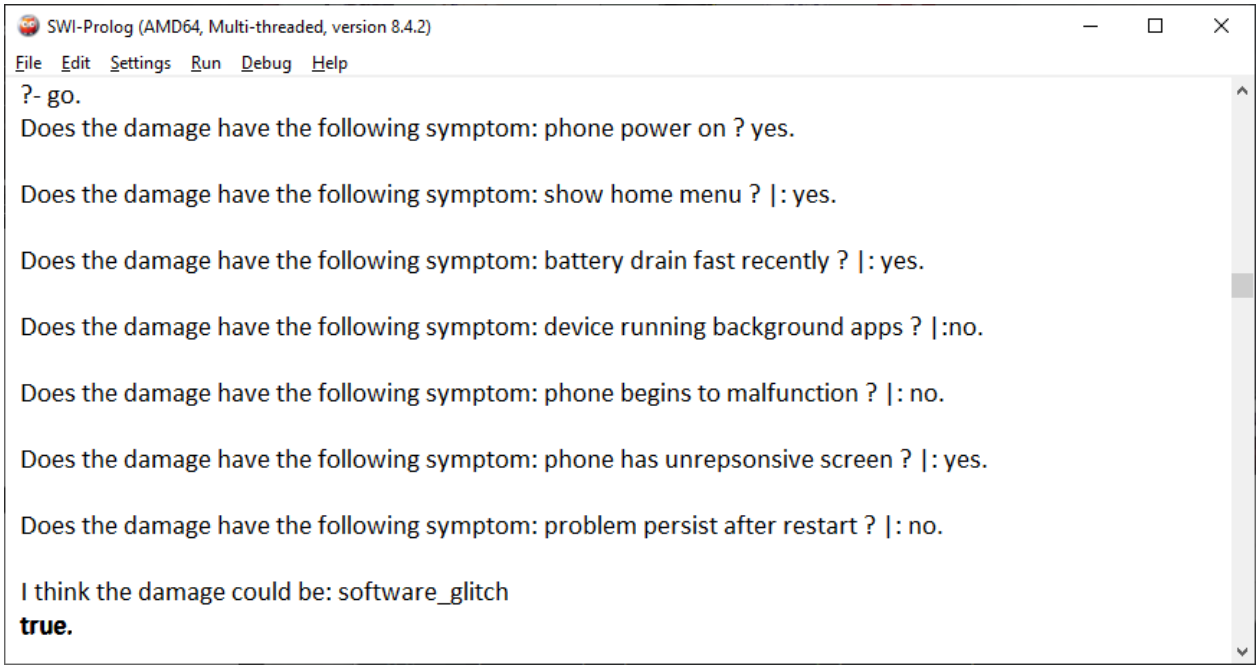
2. Παρατηρούμε ότι το σύμπτωμα “battery drain fast recently” θα ερωτηθεί στο χρήστη για πρώτη φορά στην υπόθεση «battery drain app» γιατί προηγούμενος έγινε αποκοπή από την υπόθεση «damage battery».

Στη δεύτερη περίπτωση του παραδείγματος 2 ακολουθούμε μια πιο σύνθετη διαδρομή που περιέχει παραπάνω ερωτήσεις όπως βλέπουμε στον πίνακα 4 μέχρι να καταλήξει στο ίδιο συμπέρασμα, δηλαδή στην πιθανή βλάβη «software glitch».

Πίνακας 4 Συμπτώματα για πιθανή βλάβη software glitch (Περίπτωση 2η)

Ερώτημα	**Απάντηση**
phone power on	yes
show home menu	yes
battery drain fast recently	yes
device running background apps	no
phone begins to malfunction	no
phone has unrepsonsive screen	yes
problem persist after restart	no

Στην παρακάτω εικόνα βλέπουμε την εκτέλεση του παραδείγματος 2ης περίπτωσης «software glitch».



Εικόνα 51 Εκτέλεση υπόθεσης software glitch(Περίπτωση 2η)

Παρατήρηση 2ου παραδείγματος 2ης περίπτωσης:

Στην υπόθεση «software_glitch» παρατηρούμε ότι τα δύο πρώτα συμπτώματα ενώνονται με το λογικό «ή» (or) όποτε ενώ ο χρήστης έδωσε απάντηση no στο σύμπτωμα “phone begins to malfunction” δεν γίνεται αποκοπή της υπόθεσης, αντίθετα το πρόγραμμα ρωτάει το επόμενο σύμπτωμα και εφόσον η απάντηση είναι yes ρωτάει και την τελευταία ερώτηση η οποία έχει το not μπροστά και με την απάντηση no η υπόθεση βγαίνει αληθής.

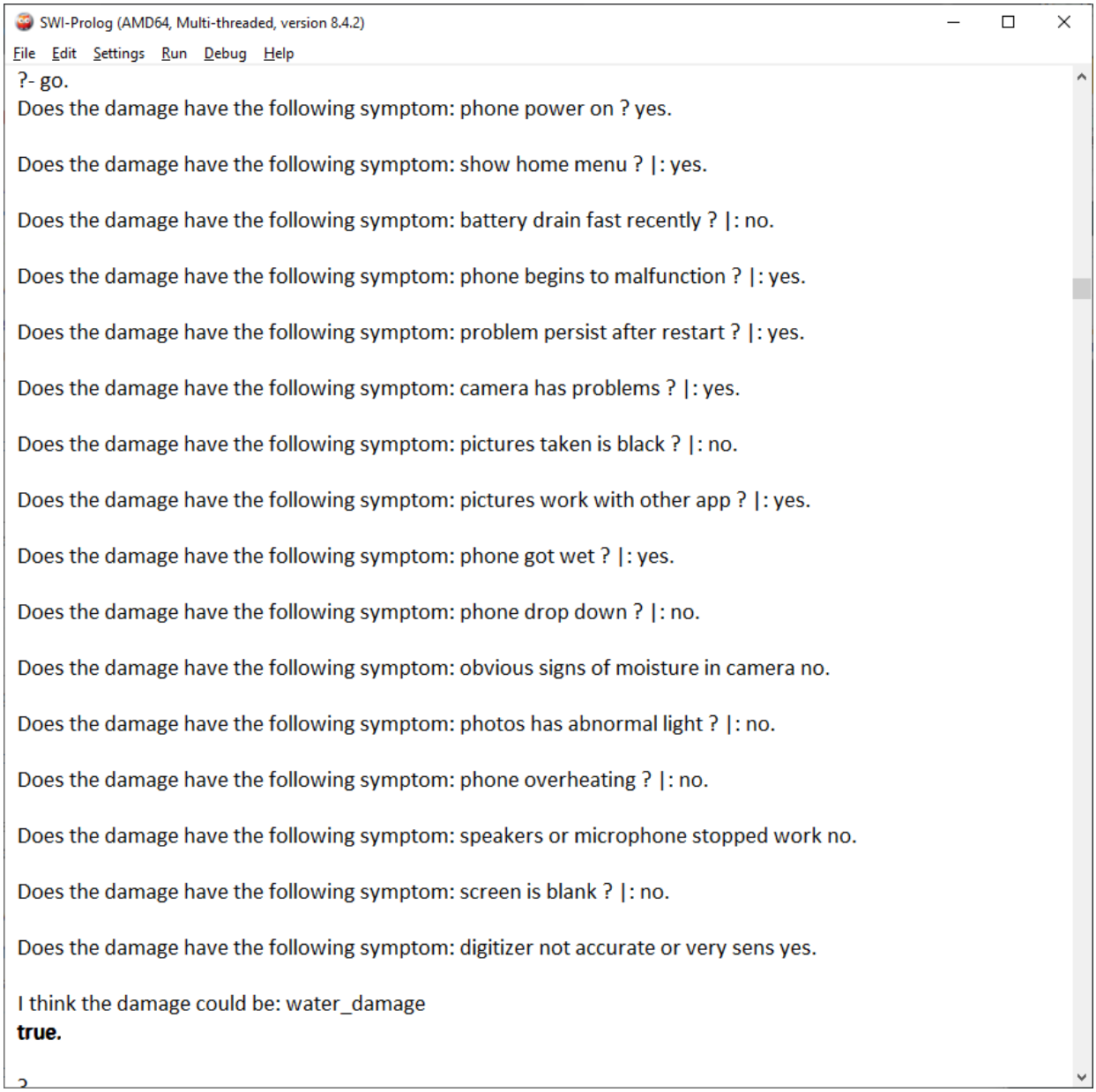
Στο 3ο παράδειγμα θα δούμε την περίπτωση του «water_damage» όπου ο χρήστης θα έχει απαντήσει στα εξής ερωτήματα τις απαντήσεις του πίνακα 5. Για να ισχύει η πιθανή βλάβη «water_damage» στα σημεία του κώδικα που περιέχει not πριν από το verify θα πρέπει να δοθεί απάντηση no.

Πίνακας 5 Συμπτώματα για πιθανή βλάβη water damage

Ερώτημα	Απάντηση
phone power on	yes
show home menu	yes

battery drain fast recently	no
phone begins to malfunction	yes
phone has unrepsonsive screen	yes
problem persist after restart	yes
camera has problems	yes
pictures taken is black	no
pictures work with other app	yes
phone got wet	yes
phone drop down	no
obvious signs of moisture in camera no.	no
photos has abnormal light	no
phone overheating	no
speakers or microphone stopped working	no
screen is blank	no
digitizer not accurate or very sensitive	yes

Στην παρακάτω εικόνα βλέπουμε την εκτέλεση του παραδείγματος «water damage».



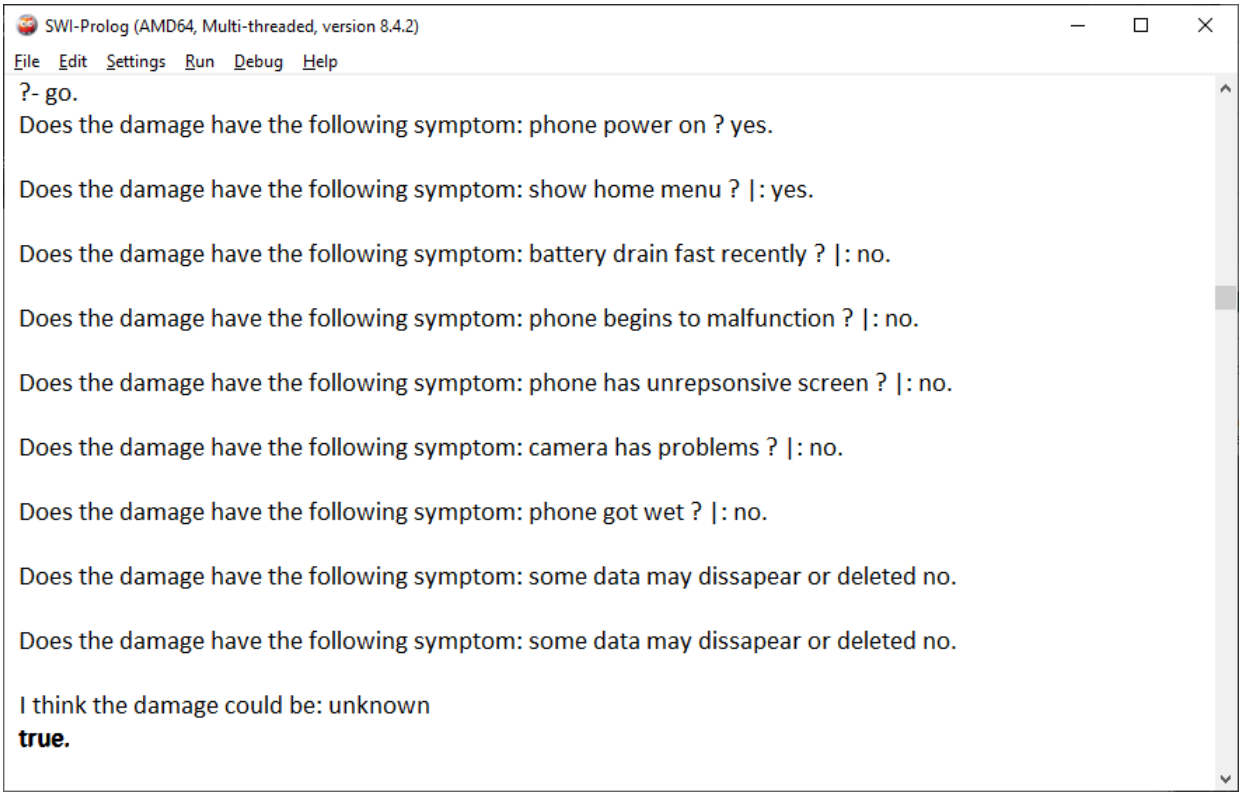
Εικόνα 52 Εκτέλεση υπόθεσης water damage

Παρατηρήσεις 3ου παραδείγματος:

1. Βλέπουμε ότι το πρόγραμμα μας είναι κοντά στο να βγάλει κάποιο συμπέρασμα αλλά εφόσον δεν ταιριάζουν όλα τα συμπτώματα προσπερνάει τις υποθέσεις μέχρι που τελικά καταλήγει στο συμπέρασμα του «water damage».
2. Στην υπόθεση «water damage» βλέπουμε ότι το πρόγραμμα δεν ρωτάει στον χρήστη την ερώτηση “phone got wet” γιατί την έχει ήδη ρωτήσει στην υπόθεση «camera damage».

3. Για να ισχύει η υπόθεση «water damage» πρέπει να ισχύει το ερώτημα “phone got wet” σε συνδυασμό με ένα από τα υπόλοιπα συμπτώματα της υπόθεσης. Έτσι όταν το πρόγραμμα πάρει θετική απάντηση σε ένα από τα ομαδοποιημένα συμπτώματα τότε η υπόθεση βγαίνει αληθής.

Στο 4ο παράδειγμα θα δούμε την περίπτωση που το πρόγραμμα δεν κατάφερε να εντοπίσει κάποια πιθανή βλάβη γιατί οι απαντήσεις του χρήστη ήταν ασαφής ή επειδή η βλάβη δεν έχει συμπεριληφθεί στο πρόγραμμα μας. Στην παρακάτω εικόνα 53 βλέπουμε την εκτέλεση του παραπάνω παραδείγματος.



Εικόνα 53 Εκτέλεση υπόθεσης unknown

Παρατήρηση 4ου παραδείγματος:

Το περιβάλλον SWI-Prolog δυσκολεύεται να εμφανίσει στο χρήστη τις μεγάλες ερωτήσεις αφού έχουν απαντηθεί όπως βλέπουμε στα δύο τελευταία συμπτώματα όπου το ένα αναφέρεται στην εσωτερική μνήμη και το άλλο στην μνήμη microSD.

ΣΥΜΠΕΡΑΣΜΑΤΑ

Οι διαγνωστικές εφαρμογές έχουν αποδειχθεί χρήσιμες στις μέρες μας γιατί βοηθούν τους χρήστες καθημερινά στην επίλυση προβλημάτων και στην εξοικονόμηση κόστους και χρόνου σε διάφορους τομείς. Η δημιουργία μιας τέτοιας εφαρμογής δεν αποτελεί εύκολη διαδικασία γιατί ο προγραμματιστής πρέπει πρώτα να έχει τεκμηριωμένα τα συμπτώματα βλαβών ή ασθενειών και να τα έχει κατανοήσει έτσι ώστε να μπορέσει να τα εισάγει σε ένα πρόγραμμα.

Σκοπός αυτής της πτυχιακής εργασίας ήταν η δημιουργία μιας διαγνωστικής εφαρμογής. Υλοποιήθηκε σε λογική γλώσσα προγραμματισμού και συγκεκριμένα στην Prolog. Στόχος της εφαρμογής είναι να γίνει η διάγνωση μιας πιθανής βλάβης σε ένα κινητό τηλέφωνο με την χρήση ερωτήσεων που καλείτε να απαντήσει ο χρήστης.

Μια δυσκολία που συναντήθηκε κατά την εκπόνηση αυτής της πτυχιακής εργασίας ήταν η έλλειψη διαθεσιμότητας επιστημονικών άρθρων διαγνωστικών εφαρμογών συγκεκριμένα σε κινητά τηλέφωνα, καθώς οι περισσότερες εφαρμογές είναι εμπορικής χρήσης.

Αξίζει να σημειωθεί, ότι παρόλες τις δυσκολίες και τα προβλήματα που συναντήθηκαν κατά την ανάπτυξη της εφαρμογής, η λειτουργικότητά της παραμένει ακέραια. Η τελική εφαρμογή που υλοποιήθηκε εξυπηρετεί το σκοπό της, όμως επιδέχεται και περαιτέρω βελτιώσεις.

Εν κατακλείδι, η μελλοντική επέκταση της εφαρμογής μας μπορεί να πραγματοποιηθεί με δύο τρόπους. Ο πρώτος τρόπος είναι να προστεθούν επιπλέον υποθέσεις βλαβών συμπεριλαμβανομένων και τα συμπτώματα τους και να εμπλουτιστούν οι ήδη υπάρχουσες υποθέσεις. Ενώ ο δεύτερος τρόπος είναι η εφαρμογή να υλοποιηθεί σε παραθυρικό περιβάλλον Visual Prolog έτσι ώστε να επιτευχθεί η ευκολότερη αξιοποίηση της εφαρμογής από το χρήστη.

ΒΙΒΛΙΟΓΡΑΦΙΑ

[1] Wikipedia, «Diagnostic Program,» 05 09 2022. [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Diagnostic_program. [Πρόσβαση 11 04 2022].

[2] «Catalysis Foundation,» [Ηλεκτρονικό]. Available: https://catalysisfoundation.org/about-us/the-role-of-diagnostics/. [Πρόσβαση 04 05 2022].

[3] J. Singla, «The Diagnosis of Some Lung Diseases in a Prolog Expert System,» *International Journal of Computer Applications,* τόμ. 78, αρ. 15, pp. 37-40, 2013.

[4] R. Aggarwal, S. Jain, Y. Jindal και N. Verma, «Approach towards Car Failure Diagnosis-An Expert System,» *International Journal of Computer Applications,* τόμ. 1, αρ. 23, pp. 61-64, 2010.

[5] S. Qiu, Z. Chen και Z. Liu, «Design and implementation of fault diagnosis expert system for missile circuit,» *Proceedings of the World Congress on Intelligent Control and Automation (WCICA),* αρ. Icsai, pp. 2977-2982, 2012.

[6] Wikipedia, «Prolog,» 10 02 2022. [Ηλεκτρονικό]. Available: https://el.wikipedia.org/wiki/Prolog. [Πρόσβαση 31 03 2022].

[7] Cleverism, 31 03 2022. [Ηλεκτρονικό]. Available: https://www.cleverism.com/skills-and-tools/prolog/. [Πρόσβαση 31 03 2022].

[8] Wikipedia, «Prolog,» 26 01 2022. [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Prolog. [Πρόσβαση 31 03 2022].

[9] «Ευφυή Συστήματα,» 01 15 2014. [Ηλεκτρονικό]. Available: https://people.iee.ihu.gr/~adamidis/intelsys.html. [Πρόσβαση 31 03 2022].

[10] I. Sakellariou, N. Vasileiadis, P. Kefalas και D. Stamatis, ΤΕΧΝΙΚΕΣ ΛΟΓΙΚΟΥ ΠΡΟΓΡΑΜΜΑΤΙΣΜΟΥ: Η γλώσσα Prolog, Kallipos, Open Academic Editions, 2015.

[11] A. Georgouli, ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ: Μια Εισαγωγική Προσέγγιση, Kallipos, Open

Academic Editions, 2015.

[12] Ε. Κ. Μαυρομιχάλη, «Εισαγωγή στη Γλώσσα Προγραμματισμού Prolog,» 2006. [Ηλεκτρονικό]. Available: https://icsdweb.aegean.gr/stamatatos/courses/Logic/Prolog/. [Πρόσβαση 2 4 2022].

[13] Wikipedia, «Cut (logic programming),» 16 11 2021. [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Cut_(logic_programming)#Types. [Πρόσβαση 7 4 2022].

[14] «SWI-Prolog's features,» [Ηλεκτρονικό]. Available: https://www.swi-prolog.org/features.html. [Πρόσβαση 2022 4 5].

[15] VisualProlog, «Visual Prolog,» [Ηλεκτρονικό]. Available: https://www.visual-prolog.com/. [Πρόσβαση 11 04 2022].

[16] FutureDial, «FutureDial,» [Ηλεκτρονικό]. Available: https://www.futuredial.com/. [Πρόσβαση 13 04 2022].

[17] PiceaSoft, «PiceaSoft,» [Ηλεκτρονικό]. Available: https://www.piceasoft.com/. [Πρόσβαση 13 04 2022].

[18] NSYS Group, «NSYS Group,» [Ηλεκτρονικό]. Available: https://nsysgroup.com/. [Πρόσβαση 13 04 2022].

[19] M360Soft, «M360Soft,» [Ηλεκτρονικό]. Available: https://m360soft.com/. [Πρόσβαση 13 04 2022].

[20] «Decision tree analysis,» 2022. [Ηλεκτρονικό]. Available: https://www.heavy.ai/technical-glossary/decision-tree-analysis. [Πρόσβαση 29 04 2022].

[21] «Διαγράμματα Δραστηριότητας,» 06 05 2017. [Ηλεκτρονικό]. Available: https://el.wikipedia.org/wiki/Διαγράμματα_δραστηριότητας. [Πρόσβαση 29 04 2022].

[22] L. Morgan, «Phones Hardware Problems: Causes, Symptoms, Fixes & Preventions,» 11 01 2017. [Ηλεκτρονικό]. Available: https://www.lemmymorgan.com/phones-hardware-problems-fixing/. [Πρόσβαση 05 05 2022].

[23] L. Morgan, «10 Most Common Mobile Phones Software Problems and How to Fix or

Prevent them,» 13 01 2017. [Ηλεκτρονικό]. Available: https://www.lemmymorgan.com/major-phones-software-problems/. [Πρόσβαση 05 05 2022].

[24] A. Miller, «Common Mobile Phone Problems and How to Fix Them,» 24 08 2021. [Ηλεκτρονικό]. Available: https://www.carlcare.com/global/tips-detail/common-mobile-phone-problems-and-how-to-fix/. [Πρόσβαση 05 05 2022].

[25] The Whiz Cells, «12 MOST COMMON CELL PHONE PROBLEMS AND THEIR SOLUTIONS,» 27 08 2018. [Ηλεκτρονικό]. Available: https://www.thewhizcells.com/12-most-common-cell-phone-problems-and-their-solutions/. [Πρόσβαση 05 05 2022].

ΠΑΡΑΡΤΗΜΑ ΚΩΔΙΚΑ

```
go :- hypothesize(Damage),
      write('I think the damage could be: '),
      write(Damage),
      nl,
      undo.


/* hypotheses to be tested */
hypothesize(damaged_charger)   :- damaged_charger, !.
hypothesize(damaged_battery)          :- damaged_battery, !.
hypothesize(power_button)             :- power_button, !.
hypothesize(operating_system)         :- operating_system, !.
hypothesize(battery_drain_app)   :- battery_drain_app, !.
hypothesize(software_glitch)          :- software_glitch, !.
hypothesize(stock_camera_app_problem):- stock_camera_app_problem, !.
hypothesize(general_app_camera_problem):- general_app_camera_problem, !.
hypothesize(camera_damage)            :- camera_damage, !.
hypothesize(camera_light_sensor):- camera_light_sensor, !.
hypothesize(water_damage)             :-water_damage, !.
hypothesize(speaker_damage)           :-speaker_damage, !.
hypothesize(storage_wear)             :-storage_wear, !.
hypothesize(corrupted_micro_sd) :-corrupted_micro_sd, !.
hypothesize('unknown').          /* no diagnose found */
```

```
/* Damage identification rules */

damaged_charger   :-not(powerup),
                    not(verify('phone charging ')),
                    verify('does phone charging with other charger ').

damaged_battery   :-not(powerup),
                     battery_drain,
                     verify('swollen battery ').

power_button              :-not(powerup),
                    not(verify('can you enter recovery mode ')).

operating_system   :-not(verify('show home menu ')),
                      verify('stucks at brand logo ').

battery_drain_app   :-battery_drain,
                      verify('device running background apps '),
                 not(verify('battery drain persist after closing all background apps ')).

software_glitch       :-(verify('phone begins to malfunction ');
                    verify('phone has unrepsonsive screen ')),!,
                    not(verify('problem persist after restart ')).
```

```
stock_camera_app_problem :-camera_problems,
                          verify('pictures taken is black ').
general_app_camera_problem     :-camera_problems,
                            not(verify('pictures work with other app ')).
camera_damage     :-camera_problems,
                  wet_phone ,
                  (verify('phone drop down ');
                  verify('obvious signs of moisture in camera lens ')).
camera_light_sensor          :-camera_problems,
                  verify('photos has abnormal light ').
water_damage                 :-wet_phone,
                  (verify('phone overheating ');
                  verify('speakers or microphone stopped working ');
                  verify('screen is blank ');
                  verify('digitizer not accurate or very sensitive ')).
speaker_damage    :-wet_phone,
                  verify('distortion in audio').
storage_wear      :-verify('some data may dissapear or deleted on internal memory
'),
                   (verify('warning about low storage memory ');
                   verify('phone random crash ');
                   verify('phone enter boot-loop ')).

corrupted_micro_sd:-verify('some data may dissapear or deleted on microSD '),
                  (verify('files not readable ');
                  verify('microSD is looks empty but it\'s not ')).
```

```
/* common rules */

powerup             :- verify('phone power on '),!.
battery_drain       :- verify('battery drain fast recently '),!.
camera_problems     :- verify('camera has problems '),!.
wet_phone           :- verify('phone got wet '),!.

/* how to ask questions */
ask(Question) :-
    write('Does the damage have the following symptom: '),
    write(Question),
    write('? '),
    read(Response),
    nl,
    ( (Response == yes ; Response == y)
      ->
       assert(yes(Question)) ;
       assert(no(Question)), fail).

:- dynamic yes/1,no/1.

/* how to verify something */
verify(S) :-
   (yes(S)
    ->
    true ;
```

```
    (no(S)
     ->
     fail ;
     ask(S))).

/* undo all yes/no assertions */
undo :- retract(yes(_)),fail.
undo :- retract(no(_)),fail.
undo.
```