ΑΛΕΞΑΝΔΡΕΙΟ ΤΕΧΝΟΛΟΓΙΚΟ ΕΚΠΑΙΔΕΥΤΙΚΟ ΙΔΡΥΜΑ
ΘΕΣΣΑΛΟΝΙΚΗΣ (Α.Τ.Ε.Ι.Θ.)
ΣΧΟΛΗ ΤΕΧΝΟΛΟΓΙΚΩΝ ΕΦΑΡΜΟΓΩΝ (ΣΤΕΦ)
ΤΜΗΜΑ ΜΗΧΑΝΙΚΩΝ ΑΥΤΟΜΑΤΙΣΜΟΥ Τ.Ε.

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

"Ανάπτυξη Λογισμικού για Διαχείριση Έξυπνων Αυτόνομων Οχημάτων"

ΚΑΤΙΚΑΡΙΔΗΣ ΔΗΜΗΤΡΙΟΣ
ΑΜ: "112968"
ΜΕΝΕΞΕΣ ΙΩΑΝΝΗΣ
ΑΜ: "112931"

ΕΠΙΒΛΕΠΩΝ ΚΑΘΗΓΗΤΗΣ: ΔΗΜΗΤΡΙΟΣ ΜΠΕΧΤΣΗΣ, ΚΑΘΗΓΗΤΗΣ ΕΦΑΡΜΟΓΩΝ

ΘΕΣΣΑΛΟΝΙΚΗ, ΣΕΠΤΕΜΒΡΙΟΣ 2017

Περίληψη - Abstract

Περίληψη

Το διαδίκτυο των αντικειμένων (Internet Of Things - IoTs) αναπτύσσεται ραγδαία μέσω της ανάπτυξης εξειδικευμένων υπηρεσιών, νέου εξοπλισμού και λογισμικού. Πρόσφορο έδαφος για την επέκταση του IoTs παρέχει η ευρεία αποδοχή των δικτύων κοινωνικής δικτύωσης που επέτρεψε την εξοικείωση με τη χρήση των δεδομένων και της πληροφορίας, οι ολοένα αυξανόμενες υπηρεσίες που παρέχονται σε επίπεδο υπολογιστικού νέφους (cloud computing) σε εταιρίες και ιδιώτες, τα εξειδικευμένα ασύρματα δίκτυα χαμηλής ισχύος και μεγάλης εμβέλειας για την ταυτόχρονη υποστήριξη χιλιάδων συσκευών, η δημιουργία και η μαζική χρήση open hardware αρχιτεκτονικών για επεξεργαστές χαμηλού κόστους. Σε αυτό το σύνθετο περιβάλλον προστίθεται η τάση για παροχή προϊόντων μεγάλης διαφοροποίησης από τη βιομηχανία προς τους τελικούς καταναλωτές, που προϋποθέτει ευέλικτες δομές παραγωγής που προσαρμόζονται άμεσα σε νέες τάσεις. Η τέταρτη φάση της βιομηχανικής επανάστασης (Industry 4.0) επαναπροσδιορίζει την εναρμόνιση της προσφοράς και της ζήτησης, τα διαθέσιμα αποθέματα, το χρόνο εξυπηρέτησης των πελατών, την ποιότητα των υπηρεσιών, την κατανάλωση πρώτων υλών και τις διαδικασίες κατεργασίας των προϊόντων, τη χρήση και τη συντήρηση των μηχανημάτων της παραγωγής αλλά και τον τρόπο και το είδος της εργασίας στην παραγωγή. Σε επίπεδο εφοδιαστικής αλυσίδας υπάρχει άμεση αλληλεπίδραση μεταξύ πελατών, εμπόρων λιανικής και χονδρικής, κατασκευαστών και προμηθευτών πρώτων υλών ώστε η παραγωγή να καθοδηγείται από τη ζήτηση και να παρέχεται η δυνατότητα ευέλικτων δομών παραγωγής.

Στην παρούσα εργασία μελετήθηκε η δημιουργία ενός δυναμικού βιομηχανικού περιβάλλοντος που είναι ικανό να προσαρμοστεί σε εναλλαγές στη ζήτηση και να διαμορφώνει δυναμικά τη γραμμή παραγωγής παρέχοντας προϊόντα μεγάλης διαφοροποίησης. Το σύνολο των οντοτήτων του βιομηχανικού περιβάλλοντος εισάγεται δυναμικά στο σύστημα και δηλώνει την παρουσία του, τις υπηρεσίες που προσφέρει και τις ιδιότητές του στην κεντρική υπηρεσία καταλόγου. Οι οντότητες λειτουργούν ως αυτόνομοι πράκτορες που επικοινωνούν μεταξύ τους μέσω των πρωτοκόλλων που έχουν αναπτυχθεί. Οι μεταφορές από και προς τα σημεία ενδιαφέροντος πραγματοποιούνται από πολλά έξυπνα αυτόνομα οχήματα με την αναζήτηση του συντομότερου μονοπατιού ενώ η όλη επικοινωνία ακολουθεί το κατανεμημένο πλαίσιο λειτουργίας των πρακτόρων λογισμικού. Από τη σκοπιά της εφοδιαστικής αλυσίδας η ανάπτυξη του λογισμικού επικεντρώθηκε σε επίπεδο διαχείρισης αποθήκης, αλλά μπορεί να επεκταθεί και σε επίπεδο παραγωγής με την προσθήκη των αντίστοιχων οντοτήτων. Ο χειριστής του συστήματος δημιουργεί την κάτοψη του βιομηχανικού χώρου, τοποθετεί τις

επιμέρους οντότητες και αναμένει την επερχόμενη ζήτηση ώστε να ξεκινήσει τη δυναμική μεταφορά των προϊόντων προς την έξοδο. Τα φορτία μπορούν να τοποθετηθούν δυναμικά σε οποιοδήποτε σημείο του χώρου, ζητούν μετακίνηση από τα οχήματα μέσω πρωτοκόλλων επικοινωνίας και επιλέγουν όχημα ανάλογα με τους κανόνες που έχουν τεθεί (μικρότερη απόσταση από το φορτίο, μεγαλύτερος χρόνος αναμονής οχήματος κ.α.). Το λογισμικό επιτρέπει την παρακολούθηση της λειτουργίας του συστήματος σε πραγματικό χρόνο.

Abstract

The Internet of Things is rapidly shaping the industrial era through innovative services, state of art equipment and the appropriate software. The social media's wide acceptance, enabled us to get familiar with the data and information use and innovative services, while cloud computing provided companies and individuals the ability to use low-powered and of large range specialized wireless services for simultaneous support of thousands devices and massive use of open hardware architectures for low cost processors. At this complex environment, there is a tendency by industry to provide high differentiation products to consumers, which presupposes flexible production structures that immediately adjust to new challenges. The fourth Industrial Revolution phase (Industry 4.0) redefines the alignment of supply and demand, available stocks, customer service time, services quality, raw material consumption and product processing, use and maintenance of production machinery. On supply chain level there is a direct interaction among customers, retailers, whole salers and raw material suppliers in order to build a demand driven supply chain network.

In this thesis, the creation of a dynamic industrial environment, capable of adjusting to demand challenges, was studied. Industrial entities are dynamically inserted into the system and declare their presence, while they offer their services at the central service directory. Entities operate as autonomous agents, communicating with each other through the developed protocols. Transportations between the points of interest are carried out, by many intelligent autonomous vehicles, while on the same time they discover the shortest path in a distributed operating framework. From the supply chain perspective, the application is focused on the warehouse management level, but it can be extended to production level, with the addition of the required entities. The user creates the facility layout, inserts the individual entities and awaits the upcoming demand in order to dynamically transport the product. Loads can be dynamically located at the facility layout, and place a movement request, using agent based communication protocols. The transportation protocol can use multiple rules (shortest distance from the loads, longer vehicle waiting time etc.) while the A * (a star) algorithm ensures the shortest distance selection. The software allows the monitoring and control of the system at real-time.

Ευχαριστίες

Στο σημείο αυτό θα θέλαμε να ευχαριστήσουμε όλους όσους συνέβαλαν στην ολοκλήρωση της διπλωματικής αυτής εργασίας και συγκεκριμένα:

- τον καθηγητή κ. Δημήτριο Μπεχτσή του Τμήματος Μηχανικών Αυτοματισμού, για την ουσιαστική καθοδήγησή τους ως επιβλέποντα της πτυχιακής αυτής εργασίας,
- τον καθηγητή κ. Δημήτριο Βλάχο του Τμήματος Μηχανολόγων Μηχανικών, της Πολυτεχνικής Σχολής ΑΠΘ, για τις συμβουλές του,
- την οικογένειά μας και τους φίλους μας για τη συμπαράσταση και την υπομονή τους κατά τη διάρκεια των φοιτητικών μας χρόνων.

Θεσσαλονίκη, Σεπτέμβριος 2017

Δημήτριος Κατικαρίδης

Ιωάννης Μενεξές

ΠΕΡΙΕΧΟΜΕΝΑ

ΠΙΝΑΚΑΣ ΕΙΚΟΝΩΝ

1. ΕΙΣΑΓΩΓΗ

Το διαδίκτυο των αντικειμένων (Internet of Things - IoTs) αναπτύσσεται ραγδαία μέσω της ανάπτυξης εξειδικευμένων υπηρεσιών, νέου εξοπλισμού και λογισμικού. Πρόσφορο έδαφος για την επέκταση του IoTs παρέχει η ευρεία αποδοχή των δικτύων κοινωνικής δικτύωσης που επέτρεψε την εξοικείωση με τη χρήση των δεδομένων και της πληροφορίας, οι ολοένα αυξανόμενες υπηρεσίες που παρέχονται σε επίπεδο υπολογιστικού νέφους (cloud computing) σε εταιρίες και ιδιώτες, τα εξειδικευμένα ασύρματα δίκτυα χαμηλής ισχύος και μεγάλης εμβέλειας για την ταυτόχρονη υποστήριξη χιλιάδων συσκευών, η δημιουργία και η μαζική χρήση open hardware αρχιτεκτονικών για επεξεργαστές χαμηλού κόστους. Σε αυτό το σύνθετο περιβάλλον προστίθεται η τάση για παροχή προϊόντων μεγάλης διαφοροποίησης από τη βιομηχανία προς τους τελικούς καταναλωτές, που προϋποθέτει ευέλικτες δομές παραγωγής που προσαρμόζονται άμεσα σε νέες τάσεις. Η τέταρτη φάση της βιομηχανικής επανάστασης (Industry 4.0) επαναπροσδιορίζει την εναρμόνιση της προσφοράς και της ζήτησης, τα διαθέσιμα αποθέματα, το χρόνο εξυπηρέτησης των πελατών, την ποιότητα των υπηρεσιών, την κατανάλωση πρώτων υλών και τις διαδικασίες κατεργασίας των προϊόντων, τη χρήση και τη συντήρηση των μηχανημάτων της παραγωγής αλλά και τον τρόπο και το είδος της εργασίας στην παραγωγή. Σε επίπεδο εφοδιαστικής αλυσίδας υπάρχει άμεση αλληλεπίδραση μεταξύ πελατών, εμπόρων λιανικής και χονδρικής, κατασκευαστών και προμηθευτών πρώτων υλών ώστε η παραγωγή να καθοδηγείται από τη ζήτηση και να παρέχεται η δυνατότητα ευέλικτων δομών παραγωγής.

Στην παρούσα εργασία μελετήθηκε η δημιουργία ενός δυναμικού βιομηχανικού περιβάλλοντος που είναι ικανό να προσαρμοστεί σε εναλλαγές στη ζήτηση και να διαμορφώνει δυναμικά τη γραμμή παραγωγής παρέχοντας προϊόντα μεγάλης διαφοροποίησης. Το σύνολο των οντοτήτων του βιομηχανικού περιβάλλοντος εισάγεται δυναμικά στο σύστημα και δηλώνει την παρουσία του, τις υπηρεσίες που προσφέρει και τις ιδιότητές του στην κεντρική υπηρεσία καταλόγου. Οι οντότητες λειτουργούν ως αυτόνομοι πράκτορες που επικοινωνούν μεταξύ τους μέσω των πρωτοκόλλων που έχουν αναπτυχθεί. Οι μεταφορές από και προς τα σημεία ενδιαφέροντος πραγματοποιούνται από πολλά έξυπνα αυτόνομα οχήματα με την αναζήτηση του συντομότερου μονοπατιού ενώ η όλη επικοινωνία ακολουθεί το κατανεμημένο πλαίσιο λειτουργίας των πρακτόρων λογισμικού. Από τη σκοπιά της εφοδιαστικής αλυσίδας η ανάπτυξη του λογισμικού επικεντρώθηκε σε επίπεδο διαχείρισης αποθήκης, αλλά μπορεί να επεκταθεί και σε επίπεδο παραγωγής με την προσθήκη των αντίστοιχων οντοτήτων. Ο χειριστής του συστήματος δημιουργεί την κάτοψη του βιομηχανικού χώρου, τοποθετεί τις επιμέρους οντότητες και αναμένει την επερχόμενη ζήτηση ώστε να ξεκινήσει τη δυναμική μεταφορά των προϊόντων προς την έξοδο. Τα φορτία μπορούν να τοποθετηθούν δυναμικά σε οποιοδήποτε σημείο

του χώρου, ζητούν μετακίνηση από τα οχήματα μέσω πρωτοκόλλων επικοινωνίας και επιλέγουν όχημα ανάλογα με τους κανόνες που έχουν τεθεί (μικρότερη απόσταση από το φορτίο, μεγαλύτερος χρόνος αναμονής οχήματος κ.α.). Το λογισμικό επιτρέπει την παρακολούθηση της λειτουργίας του συστήματος σε πραγματικό χρόνο.

1.1. Χαρακτηριστικά Αυτόνομων Οχημάτων στη βιομηχανία

Τα αυτόνομα ή αυτοοδηγούμενα οχήματα είναι ρομπότ τα οποία κινούνται μέσα σε ένα χώρο εργασίας, ακολουθώντας είτε δείκτες και γραμμές που βρίσκονται στο πάτωμα, είτε χρησιμοποιώντας κάμερες, μαγνήτες ή ανιχνευτές λέιζερ, που τα καθιστούν ικανά να αντιλαμβάνονται το περιβάλλον τους. Πιο συχνά φαίνονται να χρησιμοποιούνται σε εφαρμογές στη βιομηχανία, είτε μέσα σε αποθήκες είτε σε εγκαταστάσεις παραγωγής, για τη μετακίνηση υλικών εντός του χώρου εργασίας τους. Η χρήση και εφαρμογή τους διευρύνθηκε κατά τα τέλη του 20ού αιώνα.

Τα οχήματα, όπως προαναφέρθηκε, χρησιμοποιούν αισθητήρες προκειμένου να λάβουν πληροφορίες και ερεθίσματα, ούτως ώστε να μπορέσουν να κινηθούν στο χώρο. Παρακάτω θα αναλύσουμε τις μεθόδους που εφαρμόζονται για την οδήγησή τους.

1.1.1. Με καλωδίωση

Ένα καλώδιο τοποθετείται περίπου μία ίντσα κάτω από την επιφάνεια του πατώματος, μέσα σε σχισμή που έχει χαραχθεί σε αυτό, και προδιαγράφει τη διαδρομή που θα ακολουθήσει το αυτοοδηγούμενο όχημα. Το καλώδιο χρησιμοποιείται για τη μετάδοση ενός ραδιοφωνικού σήματος, το οποίο σήμα λαμβάνεται από το όχημα, μέσω ενός αισθητήρα που έχει τοποθετηθεί στο κάτω μέρος του, και περιέχει την πληροφορία που ελέγχει το σύστημα οδήγησης του οχήματος προκειμένου αυτό να ακολουθεί την επιθυμητή διαδρομή.

1.1.2. Με ταινία δρομολόγησης

Το όχημα ακολουθεί μία ταινία που βρίσκεται κολλημένη στο πάτωμα. Η ταινία αυτή μπορεί να είναι είτε μαγνητική είτε χρωματισμένη. Το κύριο πλεονέκτημα αυτής της μεθόδου δρομολόγησης εντοπίζεται στη δυνατότητα αφαίρεσης και επανατοποθέτησης της ταινίας σε ένα διαφορετικό σημείο, κάνοντας έτσι τη διαδρομή της διεργασίας πιο ευέλικτη. Οι ταινίες χρώματος είναι φθηνότερες αλλά περισσότερο επιρρεπείς στις φυσικές φθορές, συγκρινόμενες με τις μαγνητικές ταινίες, με αποτέλεσμα να στερούνται του πλεονεκτήματος τοποθέτησής τους σε χώρους αυξημένης κίνησης. Έπειτα, ένα ακόμη πλεονέκτημα των μαγνητικών ταινιών αποτελεί ή δυνατότητα παραμετροποίησης του σήματος που το όχημα λαμβάνει ή αποστέλλει, μεταβάλλοντας την πολικότητα της ταινίας.

1.1.3. Πλοήγηση με λέιζερ

Η πλοήγηση εδώ επιτυγχάνεται με την τοποθέτηση ταινίας αντανάκλασης επάνω σε τοίχους, στύλους είτε σε σταθερά μηχανήματα. Το όχημα είναι εξοπλισμένο με έναν αισθητήρα εκπομπής και λήψης λέιζερ σήματος, ο οποίος αισθητήρας λαμβάνει την επιστροφή του ανακλώμενου σήματος που αποστέλλεται από τον ίδιο, και έτσι πραγματοποιείται ο αυτόματος υπολογισμός της γωνίας, ή ακόμα, και της απόστασης του οχήματος από τους ανακλαστήρες. Στη συνέχεια, η πληροφορία αυτή συγκρίνεται με τον αποθηκευμένο, στην εσωτερική μνήμη του οχήματος, χάρτη που περιέχει τα σημεία ανάκλασης και, ως αποτέλεσμα, προκύπτει η σχετική θέση του οχήματος στο χώρο εργασίας, η οποία χρησιμοποιείται για το χειρισμό του συστήματος οδήγησης προκειμένου το αυτοοδηγούμενο όχημα να μένει εντός πορείας. Έτσι, λοιπόν, εκμεταλλευόμενο τη συνεχή ανανέωση της σχετικής του θέσης, μπορεί να μεταβεί στο εκάστοτε σημείο ενδιαφέροντος.

- Παλλόμενο λέιζερ

  Ένας τυπικός σαρωτής λέιζερ εκπέμπει ένα παλλόμενο φως λέιζερ, με συχνότητα 14,400 Hz, το οποίο δίνει τη μέγιστη δυνατή ανάλυση των 3.5 mrad (0.2°) ,στις 8 σαρώσεις ανά δευτερόλεπτο.
  Για να επιτευχθεί μια λειτουργική πλοήγηση, οι μετρήσεις πρέπει να παρεμβάλλονται με βάσει της έντασης του ανακλώμενου φωτός λέιζερ, ώστε να προσδιοριστεί το κέντρο του ανακλαστήρα.

- Διαμορφωμένο λέιζερ

  Η χρήση του μας δίνει καλύτερη εμβέλεια και ακρίβεια, σε αντίθεση με τα συστήματα παλλόμενου λέιζερ. Το όχημα εκπέμπει συνεχώς περιμετρικό κύμα λέιζερ πραγματοποιώντας συνεχόμενες σαρώσεις. Οι ανακλαστήρες δίνουν στο όχημα συνεχώς νέα δεδομένα, εφόσον βρίσκονται εντός οπτικού πεδίου σάρωσης του σαρωτή λέιζερ, παρέχοντας έτσι ακριβείς μετρήσεις σε κάθε σάρωση. Η εφαρμογή αυτής της μεθόδου, μπορεί να αποδώσει αποτελέσματα γωνιακής ανάλυσης της τάξεως του 0.1 mrad (0.006°), στις 8 σαρώσεις ανά δευτερόλεπτο.

1.1.4. Πλοήγηση με αισθητήρες αδράνειας και γυροσκοπίου (Inertial Navigation)

Κατά αυτήν τη μέθοδο πλοήγησης, ένα υπολογιστικό σύστημα ελέγχου, διευθύνει το όχημα ορίζοντας του συγκεκριμένους στόχους. Οι αναμεταδότες είναι τοποθετημένοι στο πάτωμα και τα

αυτοοδηγούμενα οχήματα τούς χρησιμοποιούν για να επιβεβαιώνουν ότι βρίσκονται εντός πορείας. Ένα γυροσκόπιο είναι ικανό να ανιχνεύει ακόμη και τις πιο ανεπαίσθητες μεταβολές στην κατεύθυνση του οχήματος, διορθώνοντάς το προκειμένου να το επαναφέρει στη διαδρομή του. Το περιθώριο σφάλματος, εφαρμόζοντας τη μέθοδο αυτή, είναι περίπου μία (1) ίντσα. Ο σύστημα πλοήγησης βάσει αδράνειας μπορεί να λειτουργήσει σχεδόν σε οποιοδήποτε περιβάλλον, συμπεριλαμβανομένων στενών διαδρόμων ή ακραίων θερμοκρασιών. Η εφαρμογή του, μπορεί να περιλαμβάνει τη χρήση μαγνητών, οποίοι είναι ενσωματωμένοι στο πάτωμα της εγκατάστασης, που μπορεί να διαβάσει και να ακολουθήσει το όχημα.

1.1.5. Οπτική πλοήγηση

Τα οχήματα μπορούν να εγκατασταθούν χωρίς τροποποιήσεις στο βιομηχανικό χώρο και λειτουργούν με κάμερες που καταγράφουν αντικείμενα καθ' όλη τη διάρκεια της διαδρομής, επιτρέποντάς τους να αναπαράγουν την παραπάνω διαδρομή χρησιμοποιώντας τα καταγεγραμμένα αντικείμενα για να πλοηγηθούν. Για να επιτευχθεί αυτό, χρησιμοποιούνται τεχνικές πιθανολογικής ογκομετρικής ανίχνευσης, τεχνολογία που ονομάζεται Evidence Grid και αναπτύχθηκε από τον Dr. Hans Moravec στο πανεπιστήμιο Carnegie Mellon. Η τεχνολογία Evidence Grid χρησιμοποιεί τις πιθανότητες κατοχής για κάθε σημείο στο χώρο, ούτως ώστε να αντισταθμίσει την αβεβαιότητα στην απόδοση των αισθητήρων και στο περιβάλλον. Οι κύριοι αισθητήρες πλοήγησης είναι ειδικά σχεδιασμένες στέρεο κάμερες. Τα οχήματα χρησιμοποιούν εικόνες 360 μοιρών και δημιουργούν ένα 3D χάρτη, ο οποίος τους επιτρέπει να ακολουθήσουν μια προκαθορισμένη διαδρομή χωρίς, είτε, εξωτερική παρέμβαση, είτε, την προσθήκη νέων χαρακτηριστικών στο χώρο κίνησης (σημεία ενδιαφέροντος για συστήματα αναγνώρισης θέσης)

1.1.6. Πλοήγηση με χρήση πληροφοριών από πολλές μεθόδους

Η μέθοδος αυτή περιλαμβάνει τη χρήση ποικίλων αισθητήρων, πολλούς από τους οποίους έχουμε ήδη αναφέρει, και αξιοποιεί τα δεδομένα που λαμβάνει από αυτούς μέσω τεχνικών εντοπισμού θέσης, όπως οι τεχνικές Monte-Carlo / Marcov, προκειμένου να γίνει σωστή εκτίμηση της τοποθεσίας του οχήματος καθώς, δυναμικά, πραγματοποιείται ο σχεδιασμός της συντομότερης επιτρεπόμενης διαδρομής προς το στόχο του. Η ευελιξία για παράδοση, κατά παραγγελία, σε οποιοδήποτε σημείο στο χώρο, αποτελεί σημαντικό πλεονέκτημα αυτών των συστημάτων. Είναι ικανά να διαχειριστούν σενάρια και περιπτώσεις αποτυχίας χωρίς να συμπαρασύρουν ολόκληρη τη λειτουργία παραγωγής, καθότι τα οχήματα μπορούν να σχεδιάσουν νέες διαδρομές αποφεύγοντας τις συσκευές που έχουν υποστεί βλάβη αποτελώντας, πλέον, εμπόδια.Ακόμη, η εγκατάστασή τους δεν απαιτεί πολύ χρόνο, γεγονός που δε θέτει το εργοστάσιο εκτός λειτουργίας για μεγάλο χρονικό διάστημα.

1.1.7. Γεωπλοήγηση

Κατά την πλοήγηση αυτή το όχημα αναγνωρίζει το περιβάλλον του προκειμένου να εντοπίσει τη θέση του. Το όχημα, εξοπλισμένο με τεχνολογία γεωπλοήγησης, ανιχνεύει ράφια, στήλες και τοίχους, εντός της αποθήκης, και χρησιμοποιώντας τα παραπάνω ευρήματα ως σταθερά σημεία αναφοράς, εντοπίζει την τοποθεσία του σε πραγματικό χρόνο και καθορίζει την πορεία του. Οι διαδρομές που μπορεί να ακολουθήσει είναι απείρως τροποποιήσιμες αφού δεν υπάρχουν περιορισμοί στις αποστάσεις που να καλύπτουν τον αριθμό των θέσεων παραλαβής ή εκφόρτωσης.

1.2. Αυτοοδηγούμενα οχήματα και καταλληλότητα

Τα αυτοματοποιημένα καθοδηγούμενα οχήματα (AGV) θεωρούνται συνήθως ως απλά μηχανήματα που εκτελούν απλές εργασίες αντί του ανθρώπινου δυναμικού. Την τελευταία δεκαετία έχουμε δει AGVs να ενσωματώνονται σε πολλές βιομηχανίες εκτός της διανομής και της μεταποίησης - όπως στη λιανική πώληση, στη στρατιωτική ή ακόμη και υγειονομική περίθαλψη. Με αυτόν τον αυξανόμενο ρυθμό χρήσης των AGVs, το λογικό ερώτημα που έρχεται στο μυαλό για πολλούς ιδιοκτήτες επιχειρήσεων και διαχειριστές επιχειρήσεων είναι το εξής: Πώς μπορώ να προσδιορίσω αν τα AGVs είναι κατάλληλα για τη βιομηχανία μου και για τις δραστηριότητές μου; Ποια είναι τα πλεονεκτήματα και τα μειονεκτήματα των AGV που μπορώ να σταθμίσω, για να αποφασίσω αν θα τα χρησιμοποιήσω; Η αλήθεια είναι πως τα AGV δεν είναι κατάλληλα για κάθε κλάδο ή σύνολο δραστηριοτήτων. Ας δούμε μερικά από τα σημαντικά οφέλη αλλά και μειονεκτήματα της χρήσης των AGV στη βιομηχανία.

1.2.1. Μειωμένο κόστος ανθρώπινου δυναμικού.

Η χρήση AGV έναντι ανθρώπου, μειώνει το εργατικό κόστος. Με την επένδυση ενός αρχικού κεφαλαίου για την απόκτηση του οχήματος, η εταιρία απαλλάσσεται από τα πάγια έξοδα που έχει ένας άνθρωπος-εργάτης, όπως για παράδειγμα ο μισθός του, που δεν θα είναι ποτέ μηδενικός, υγειονομική περίθαλψη, φορολογία, άδειες διακοπών και λοιπά.

1.2.2. Αυξημένη ασφάλεια των εργαζομένων.

Τα αυτοοδηγούμενα οχήματα είναι προγραμματισμένα ώστε να εκτελούν τις ενέργειές τους πάντοτε βάσει προκαθορισμένων κανόνων και με ασφάλεια. Ο παράγοντας «ασφάλεια», άλλωστε, είναι εφικτός χάρη στους αισθητήρες που διαθέτει το κάθε όχημα που του επιτρέπουν να αποφεύγει εμπόδια, τα οποία ενδεχομένως να παρουσιαστούν μέσα στο χώρο εργασίας που πρέπει να κινηθεί το όχημα, ή ακόμη και ανθρώπους που βρίσκονται στο ίδιο περιβάλλον. Στον αντίποδα όμως, τα οχήματα που απαιτούν οδήγηση από άνθρωπο, δεν κατέχουν κάποιο αυτόματο σύστημα ασφαλείας, θέτοντας τον χειριστή τους υπεύθυνο για τη λήψη αποφάσεων, όπου αυτό είναι αναγκαίο. Εξαιτίας, συνεπώς, του υπαρκτού ανθρώπινου παράγοντα, που μπορεί να είναι είτε η κούραση είτε η λανθασμένη αντίληψη των καταστάσεων, οι πιθανότητες λήψης μιας λάθος απόφασης αυξάνονται, αυξάνοντας μαζί τους και το ενδεχόμενο πρόκλησης ατυχήματος. Επιπροσθέτως, τα αυτοοδηγούμενα οχήματα μπορούν να «εργαστούν» σε αντίξοες συνθήκες όπου ο άνθρωπος δεν θα μπορούσε να ανταπεξέλθει, όπως οι πολύ χαμηλές ή υψηλές θερμοκρασίες, η αυξημένη υγρασία / ξηρασία, θόρυβος κλπ.. Εν τέλει, όλη αυτή η πρόληψη για ασφάλεια, οδηγεί σε μηδενικά κόστη ζημιών που μπορεί να προκύψουν, ή μάλιστα και σε ενδεχόμενη αύξηση απόδοσης και ευελιξίας των διεργασιών.

1.2.3. Αύξηση της ακρίβειας και επιτυχίας των στόχων που θέτει η επιχείρηση.

Σε αντίθεση με τον άνθρωπο, η μηχανή δεν θα κάνει ποτέ λάθος μέσα σε μία γραμμή παραγωγής. Επίσης, η μηχανή δεν θα αλλοιώσει την αντίληψή της λόγω κούρασης αλλά, αντίθετα, είναι ικανή να δουλεύει 24 ώρες την ημέρα, για 7 μέρες την εβδομάδα. Έτσι ο άνθρωπος μπορεί να επενδύσει ώρες σε άλλους τομείς όπως την έρευνα και την ανάπτυξη.

1.2.4. Επεκτασιμότητα.

Η αύξηση του πλήθους διεργασιών μιας βιομηχανίας και του απαιτούμενου δυναμικού, είναι μεγέθη ανάλογα. Η αύξηση του πρώτου έχει ως συνέπεια να αυξάνει το δεύτερο. Έτσι λοιπόν μπορούμε να προσθέτουμε όλο και περισσότερα οχήματα μέσα σε μια διεργασία, σταδιακά και σύμφωνα με το φόρτο εργασίας που αυξάνεται, χωρίς να χρειάζεται να αγοράσουμε όλα τα οχήματα εξ' αρχής, δαπανώντας μεγάλο αρχικό κεφάλαιο. Τελικά θα καταλήξουμε να έχουμε μια εντελώς αυτόνομη και αυτόματη διεργασία με χρήση όλο και μικρότερου ανθρώπινου δυναμικού.

1.2.5. Μείωση ρύπων.

Ένα αυτοοδηγούμενο όχημα είναι αποδεδειγμένο πως εκπέμπει λιγότερους ρύπους από ένα όχημα που το διαχειρίζεται άνθρωπος, διότι τα τελευταία είναι, κατά κόρον, πετρελαιοκίνητα, έναντι των αυτοοδηγούμενων οχημάτων, η επιλογή των οποίων συνήθως γίνεται έτσι ώστε να είναι ηλεκτροκίνητα, κινούμενα από μπαταρία.

1.2.6. Θέσεις εργασίας ατόμων με εξειδικευμένα προσόντα.

Όπως κάθε υπολογιστικό σύστημα, έτσι και τα αυτοοδηγούμενα οχήματα απαιτούν προγραμματισμό και συντήρηση. Για να πραγματοποιηθούν αυτά χρειάζονται άνθρωποι με εξειδικευμένες γνώσεις. Αυτό έχει ως αποτέλεσμα τη δημιουργία νέων θέσεων εργασίας με απαιτούμενες γνώσεις στον πληροφοριακό και ηλεκτρονικό τομέα.
Φυσικά, εκτός από τα πλεονεκτήματα, υπάρχουν και αρκετά σημεία όπου τα αυτοοδηγούμενα οχήματα κρίνονται ακατάλληλα.

1.2.7. Υψηλό κόστος αρχικής επένδυσης

Πιο πάνω αναφέραμε πως μία βιομηχανία έχει λιγότερα έξοδα εάν έχει στην κατοχή της κάποια αυτοοδηγούμενα οχήματα. Το κύριο πρόβλημα είναι το αρχικό κεφάλαιο επένδυσης για την αγορά των πρώτων AGVs. Μακροπρόθεσμα λοιπόν τα αυτοοδηγούμενα οχήματα μειώνουν τα έξοδα μιας εταιρίας, αλλά στο άμεσο μέλλον υπάρχει μία αβεβαιότητα σχετικά με την τελική απόδοση της βιομηχανίας. Ακόμη, ένα τέτοιο όχημα απαιτεί συντήρηση ή οποία κοστίζει χρήματα αλλά και χρόνο, γεγονός που συνεπάγεται την έλλειψη ενός οχήματος από τη γραμμή παραγωγής και εν συνεχεία, οδηγεί στην καθυστέρησητης της γραμμής παραγωγής μιας βιομηχανίας.

1.2.8. Κατάλληλα κυρίως για επαναλαμβανόμενες διεργασίες

Τα αυτοοδηγούμενα οχήματα είναι κατάλληλα κυρίως για διεργασίες οι οποίες είναι περιοδικές. Τα ρομπότ αρχικοποιούνται και εκτελούν την ίδια διαδικασία συνεχώς. Εάν μία διεργασία δεν

είναι επαναλαμβανόμενη, ενδεχομένως, να είναι ευκολότερη και ταχύτερη για έναν εργάτη που οδηγεί ένα όχημα

1.3. Χαρακτηριστικά εκπομπών αυτοοδηγούμενων οχημάτων

Εκτός από την κατηγοριοποίηση με βάση τους αισθητήρες, μπορούμε να κατατάξουμε τα αυτοοδηγούμενα οχήματα και βάσει της εκπομπής των ρύπων. Παρακάτω ακολουθεί ένα ενδεικτικό παράδειγμα από στοιχεία πραγματικών οχημάτων.

Οι τιμές που εμφανίζονται παρακάτω, αποτελούν δεδομένα της δημοσιευμένης εργασίας του Fuc et al., (2014) και χρησιμοποιεί την ανάλυση για οχήματα χωρητικότητας τριών τόνων που διανύουν την συνολική απόσταση του ενός χιλιομέτρου. Οι ρύποι βάσει του ευρωπαϊκού προτύπου αποτελούνται κυρίως από πέντε κατηγορίες χημικών. Το διοξείδιο του άνθρακα (CO2), το μονοξείδιο του άνθρακα (CO), οξείδια του αζώτου (NOx), συνολικός υδρογονάνθρακας (THC) και ισοδύναμο υπερθέρμανσης του πλανήτη (Global Warming equivalent). Οι μετρήσεις έχουν πραγματοποιηθεί εξετάζοντας τα σενάρια κατά τα οποία το όχημα είναι είτε επιβαρυμένο από κάποιο φορτίο είτε οχι, προκειμένου να φανεί η επιρροή του φορτίου πάνω σε ένα όχημα.

1.3.1. Κατηγορία LPG (Liquefied petroleum gas)

Με φορτίο

1. CO2: 2959.57 γραμμάρια
2. CO: 27.04 γραμμάρια
3. NOx: 19.63 γραμμάρια
4. THC: 3.06 γραμμάρια
5. Global warming: 3.58 κιλογραμμάρια

Χωρίς φορτίο

1. CO2: 1935.16 γραμμάρια
2. CO: 13.36 γραμμάρια
3. NOx: 13.90 γραμμάρια
4. THC: 1.51 γραμμάρια
5. Global warming eq: 2.33 κιλογραμμάρια

1.3.2. Κατηγορία DSL (Diesel)

Με φορτίο

1. CO2: 2130.11 γραμμάρια
2. CO: 7.28 γραμμάρια
3. NOx: 20.16 γραμμάρια
4. THC: 1.77 γραμμάρια
5. Global warming: 2.49 κιλογραμμάρια

Χωρίς φορτίο

1. CO2: 1510.83 γραμμάρια
2. CO: 3.84 γραμμάρια
1. NOx: 14.33 γραμμάρια
2. THC: 1.08 γραμμάρια
3. Global warming eq: 1.2 κιλογραμμάρια

1.3.3. Κατηγορία ELE (Electric)

Με φορτίο

1. CO2: 0 γραμμάρια
2. CO: 0 γραμμάρια
3. NOx: 0 γραμμάρια
4. THC: 0 γραμμάρια
5. Global warming: 0.67 κιλογραμμάρια

Χωρίς φορτίο

1. CO2: 0 γραμμάρια
2. CO: 0 γραμμάρια
3. NOx: 0 γραμμάρια
4. THC: 0 γραμμάρια
5. Global warming eq: 0.64 κιλογραμμάρια

Στην περίπτωση της κατηγορίας των ηλεκτροκίνητων οχημάτων υπάρχουν έμμεσοι ρύποι τους οποίους δεν υπολογίζουμε.

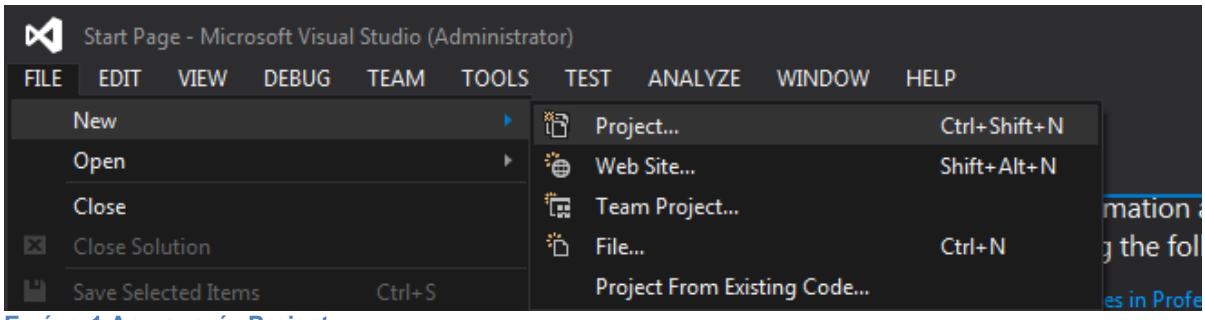
2. ΠΕΡΙΒΑΛΛΟΝ ΑΝΑΠΤΥΞΗΣ ΤΗΣ ΕΦΑΡΜΟΓΗΣ

2.1. Εισαγωγή

Για τις ανάγκες της εργασίας, χρησιμοποιήθηκε το περιβάλλον ανάπτυξης «Microsoft Visual Studio 2013», της εταιρείας Microsoft, και η γλώσσα στην οποία πραγματοποιήθηκε η ανάπτυξη της εφαρμογής είναι η Visual C#. Ο κύριος λόγος που προτιμήθηκε το συγκεκριμένο περιβάλλον ανάπτυξης είναι επειδή, σε συνδυασμό με τη Visual C#, προσφέρει στο χρήστη τη δυνατότητα να δημιουργήσει παραθυρικές εφαρμογές, χωρίς να απαιτεί εξειδικευμένες γνώσεις. Και αυτό γιατί η συγκεκριμένη γλώσσα είναι φιλική προς αρχάριους χρήστες και, ταυτόχρονα, σημαντικά ευέλικτη προς τη κατεύθυνση προγραμματισμού παραθυρικών εφαρμογών. Η ομοιότητά της, επίσης, με τη γλώσσα προγραμματισμού C++, την καθιστά προσιτή σε κάποιον που θα ήθελε να ασχοληθεί με τη Visual C#, γνωρίζοντας ήδη τη C++, καθώς πέραν των όποιων ομοιοτήτων που θα συναντήσει στη δομή των δύο γλωσσών, η Microsoft έχει δώσει πολύ μεγάλη προσοχή στον τρόπο που έχει οργανώσει την τεκμηρίωση, και για τις 2 γλώσσες, παρέχοντας στους χρήστες αναλυτική υποστήριξη και βοήθεια για καθεμία από αυτές. Ας δούμε όμως πώς μπορεί κάποιος να δημιουργήσει μια απλή εφαρμογή, όπως για παράδειγμα μια εφαρμογή που θα εμφανίζει το μήνυμα «Hello world!», μέσα από το περιβάλλον του Visual Studio 2013 χρησιμοποιώντας τη γλώσσα Visual C#.

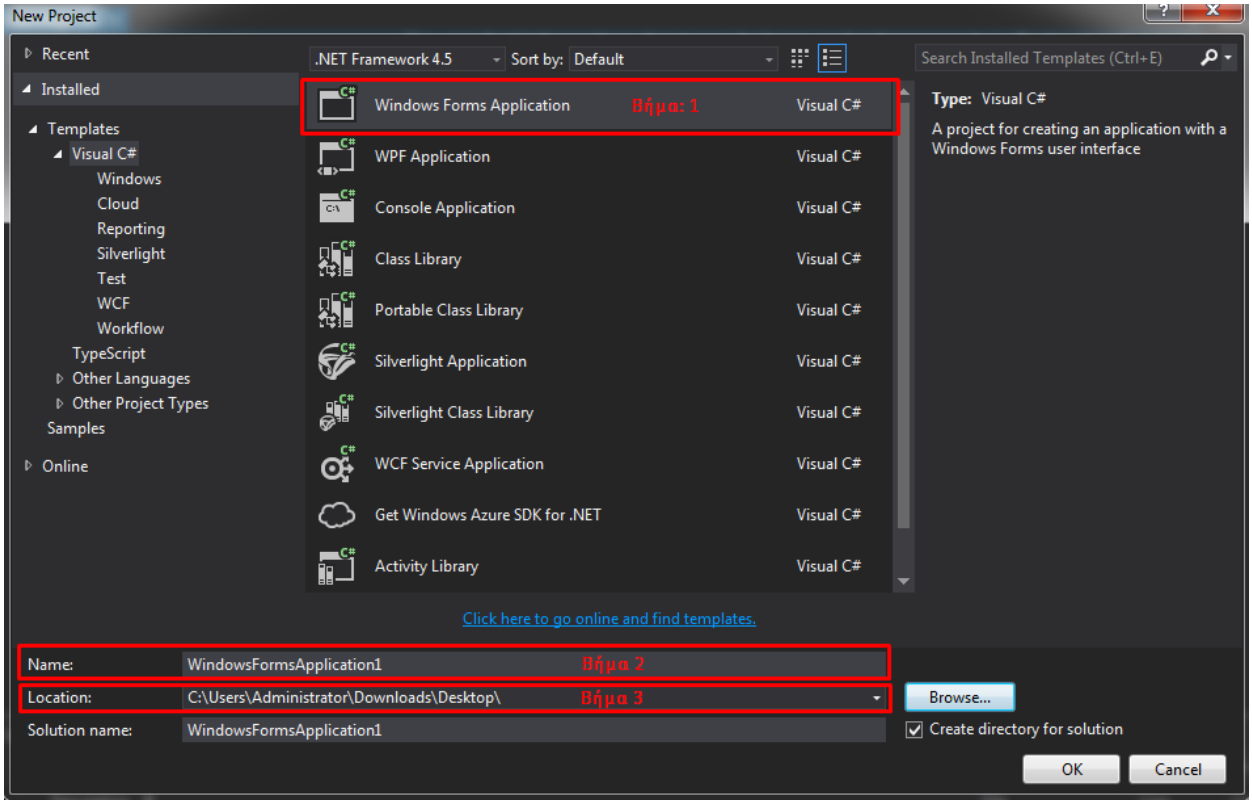
2.2. Εισαγωγή στο Visual Studio και στη Visual C#

Ανοίγοντας το Visual Studio 2013, οδηγούμαστε, από το μενού στο πάνω μέρος της εφαρμογής, στην επιλογή File -> New -> Project.



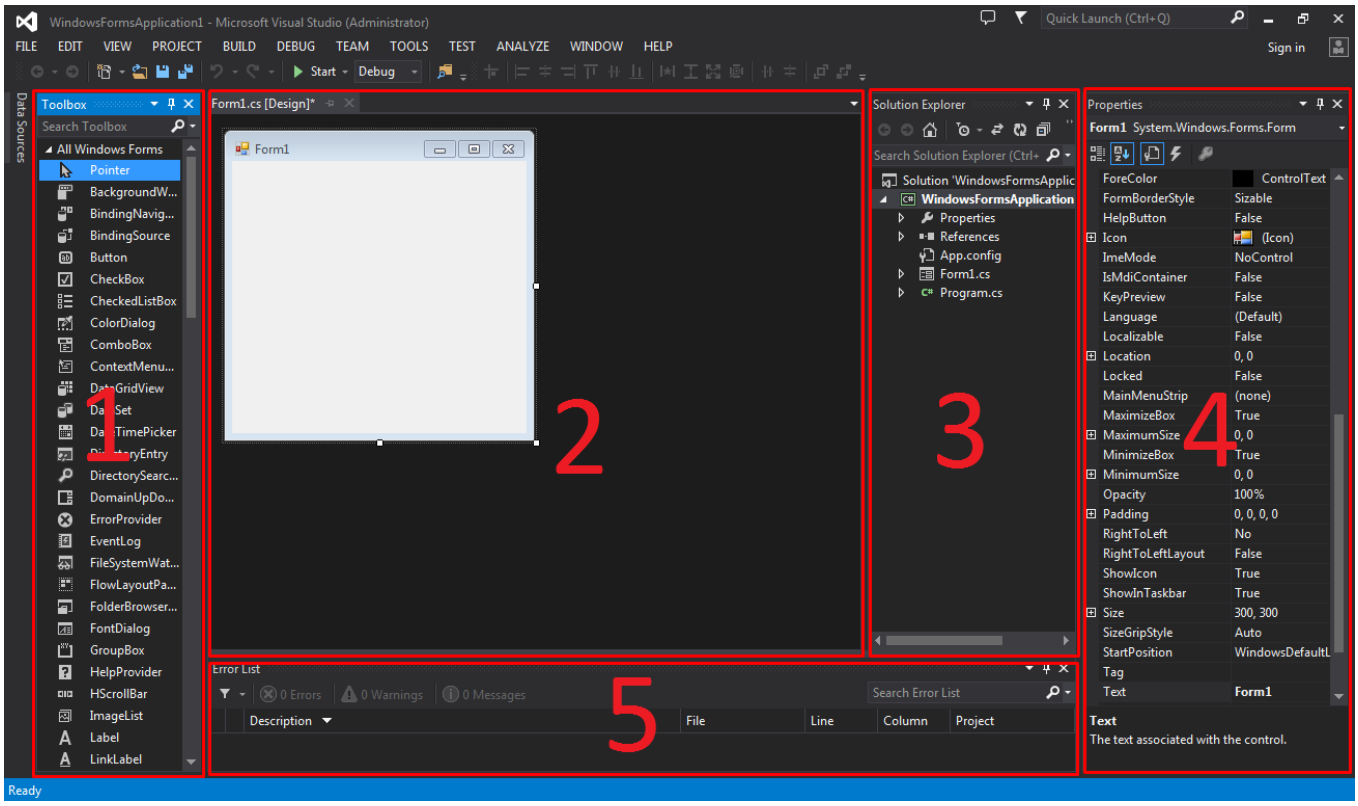
Εικόνα 1 Δημιουργία Project

Από το παράθυρο που μας εμφάνισε, επιλέγουμε την επιλογή «Windows Form Application» και, αν επιθυμούμε, μπορούμε να ορίσουμε το όνομα του Project καθώς και τη διαδρομή στην οποία αυτό θα αποθηκευτεί. Επιλέγουμε «OK»



Εικόνα 2 Ολοκλήρωση δημιουργίας Project

Σε αυτό το σημείο, η δημιουργία του Project έχει ολοκληρωθεί και ξεκινάει το κομμάτι της ανάπτυξης της εφαρμογής που ο χρήστης έχει σκοπό να φτιάξει. Έχοντας, λοιπόν, επιλέξει «OK» στο προηγούμενο βήμα (2), τώρα θα πρέπει να έχουμε μπροστά μας την παρακάτω εικόνα:

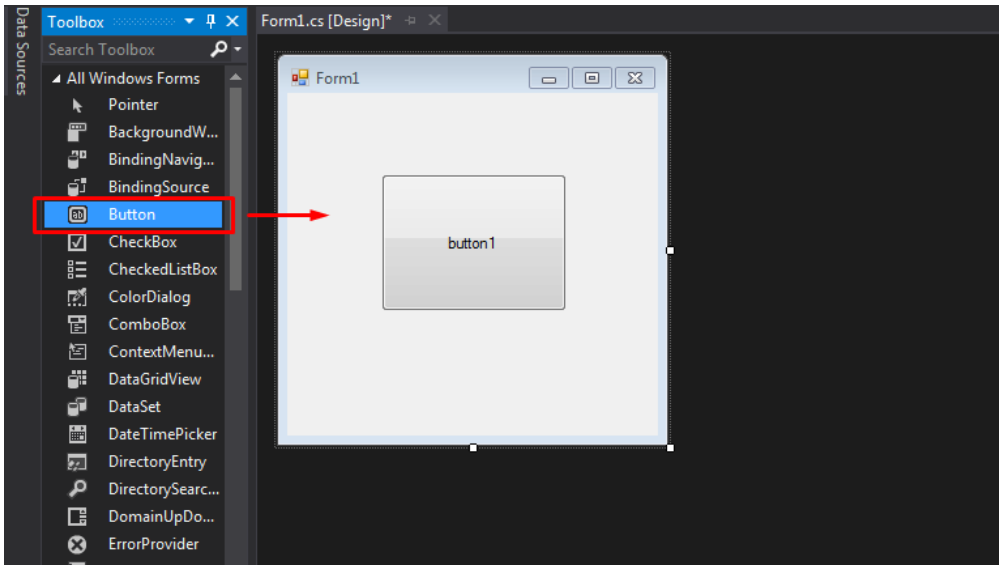


Εικόνα 3 Visual Studio παράθυρα

Ας εξηγήσουμε τί είναι το κάθε ένα από τα πέντε αριθμημένα μέρη της εικόνας που βλέπουμε

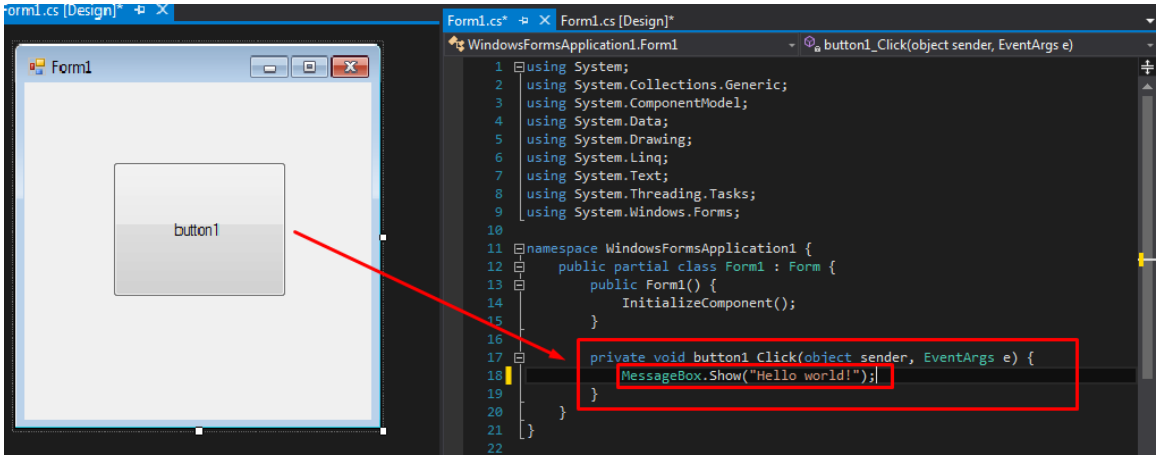
1. Παράθυρο όπου περιέχονται τα αντικείμενα που ο χρήστης μπορεί να εισάγει στη φόρμα.
2. Παράθυρο όπου βρίσκεται η φόρμα την οποία επεξεργάζεται ο χρήστης
3. Παράθυρο από όπου μπορεί να γίνει πλοήγηση στα αρχεία που απαρτίζουν το project (Solution Explorer).
4. Παράθυρο που περιέχει τις ιδιότητες από το εκάστοτε επιλεγμένο, από το χρήστη, αντικείμενο.
5. Εδώ εμφανίζονται τυχόν λάθη ή προειδοποιήσεις καθώς επίσης και στοιχεία που βοηθούν στον εντοπισμό και διόρθωση τους όπως π.χ η περιγραφή του μηνύματος, το αρχείο ή η σειρά που εντοπίστηκε κάποιο λάθος.

Από το παράθυρο αντικειμένων στο αριστερό μέρος της οθόνης, επιλέγουμε το αντικείμενο «Button» και το μετακινούμε (drag and drop) επάνω στη φόρμα (στο παράθυρο με αριθμό 2 που περιγράψαμε παραπάνω)



Εικόνα 4 Εισαγωγή button σε φόρμα

Κάνοντας διπλό κλικ επάνω στο αντικείμενο του Button, που πλέον βρίσκεται μέσα στη φόρμα, θα εμφανιστεί ένα νέο παράθυρο όπου βρίσκεται ο κώδικας που χρησιμοποιεί η συγκεκριμένη φόρμα. Αντιγράφουμε το «MessageBox.Show("Hello world!");» όπως φαίνεται στην παρακάτω εικόνα:

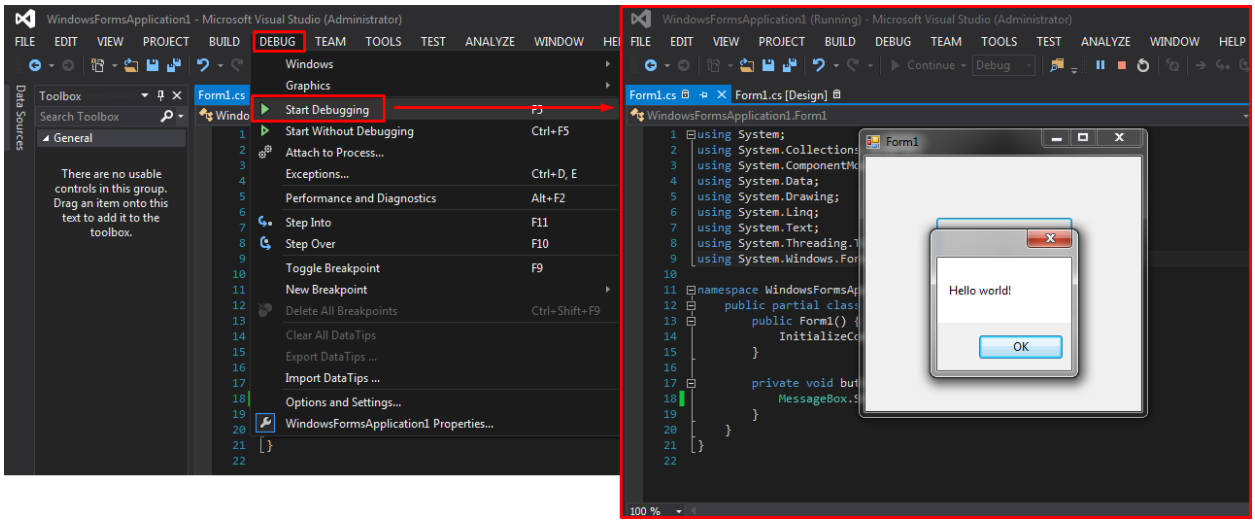


Εικόνα 5 Δημιουργία ενός Click Event για button

Να σημειωθεί πως με το διπλό κλικ που μόλις κάναμε, δημιουργήθηκε αυτομάτως το κομμάτι κώδικα που θα εκτελεστεί όταν ο χρήστης, κατά την εκτέλεση της εφαρμογής, πατήσει το Button.

Για να «τρέξουμε» την εφαρμογή που μόλις δημιουργήσαμε, επιλέγουμε από το επάνω μέρος του προγράμματος την επιλογή "Debug" -> "Start Debugging". Κατά την εκτέλεση της εφαρμογής που

φτιάξαμε, πατώντας επάνω στο Button, θα εμφανίζεται το μήνυμα «Hello world!»

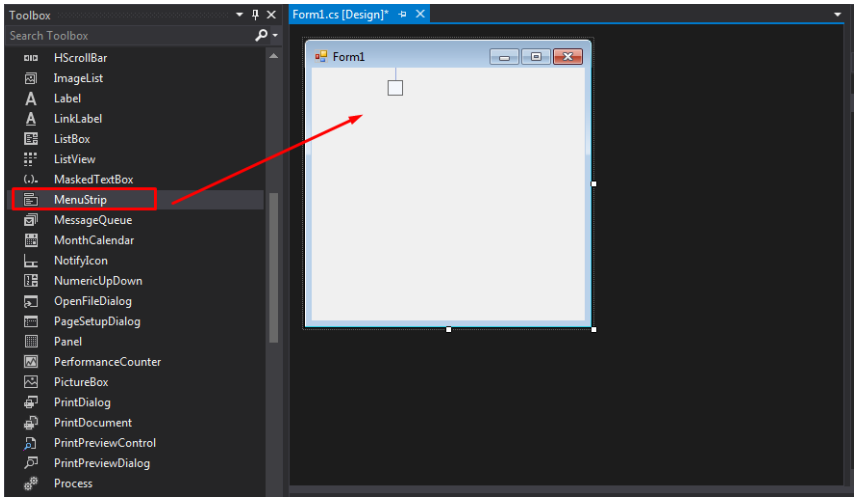


Εικόνα 6 Compile και εκτέλεση της εφαρμογής

2.3. Δημιουργία διαδραστικής φόρμας και μενού

Με παρόμοιο τρόπο, όπως στο πρώτο βήμα της προηγούμενης ενότητας, ξεκινάμε να φτιάξουμε το καινούριο Project ακολουθώντας τα βήματα 1 & 2 που περιγράφηκαν παραπάνω (File -> New -> Project…)

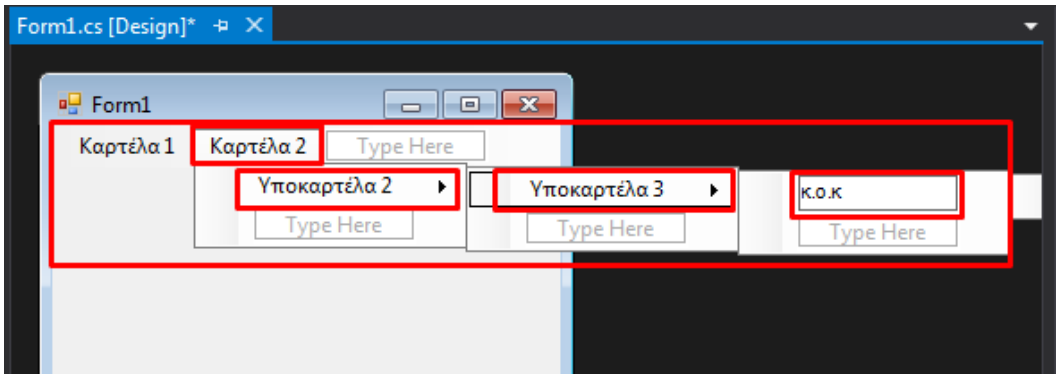
Από την εργαλειοθήκη, στο αριστερό μέρος της οθόνης, επιλέγουμε το αντικείμενο «MenuStrip» και το τοποθετούμε στην άδεια φορμα, όπως φαίνεται παρακάτω:



Εικόνα 7 Εισαγωγή MenuStrip σε φόρμα

Επόμενο βήμα είναι να ονοματίσουμε τα επιμέρους μενού από τα οποία θα αποτελείται το αντικείμενο «MenuStrip». Επιλέγοντας το μενού από τη φόρμα, δημιουργούμε τα διαδοχικά μενού εισάγοντας τα

ονόματα της κάθε καρτέλας:

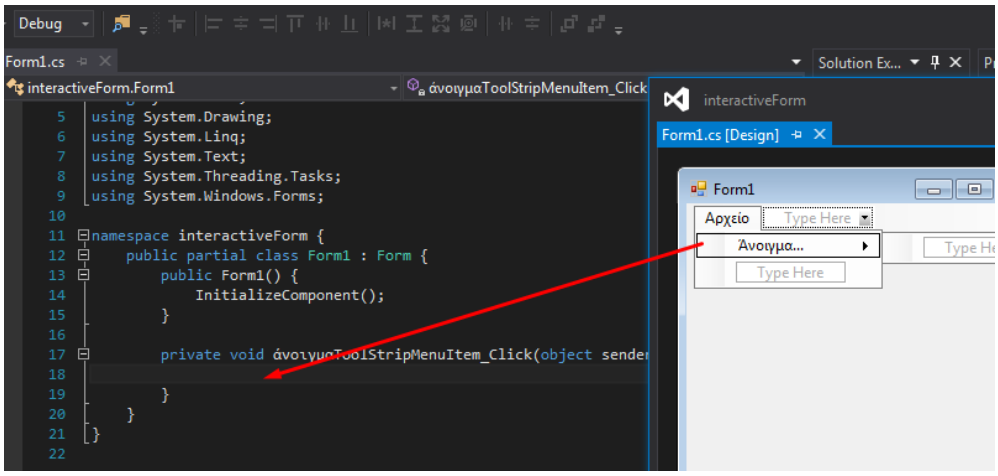


Εικόνα 8 Δημιουργία υπομενού σε Menustrip

Παρατηρούμε πως ο χρήστης μπορεί να εισάγει πολλαπλές καρτέλες που είτε είναι ανεξάρτητες μεταξύ τους (Καρτέλα 1 – Καρτέλα 2), είτε ανήκουν στην προηγούμενη (Καρτέλα 2 -> Υποκαρτέλα 2 -> κλπ κλπ).

Σημείωση: Από το παράθυρο Ιδιοτήτων, είναι εφικτή η αλλαγή τόσο του προγραμματιστικού ονόματος του εκάστοτε αντικειμένου όσο και του ονόματος με το οποίο που εμφανίζεται στη φόρμα.

Κάνοντας διπλό κλικ στο μενού που επιθυμούμε, μεταφερόμαστε σε ένα καινούριο παράθυρο το οποίο περιέχει τον κώδικα της κύριας φόρμας και παρατηρούμε πως έχει δημιουργηθεί ένα τμήμα κώδικα. Αυτό σημαίνει πως με το παραπάνω διπλό κλικ, ενεργοποιήσαμε τη συνάρτηση που θα καλείται κάθε φορά που κάνουμε κλικ πάνω στην υποκαρτέλα.

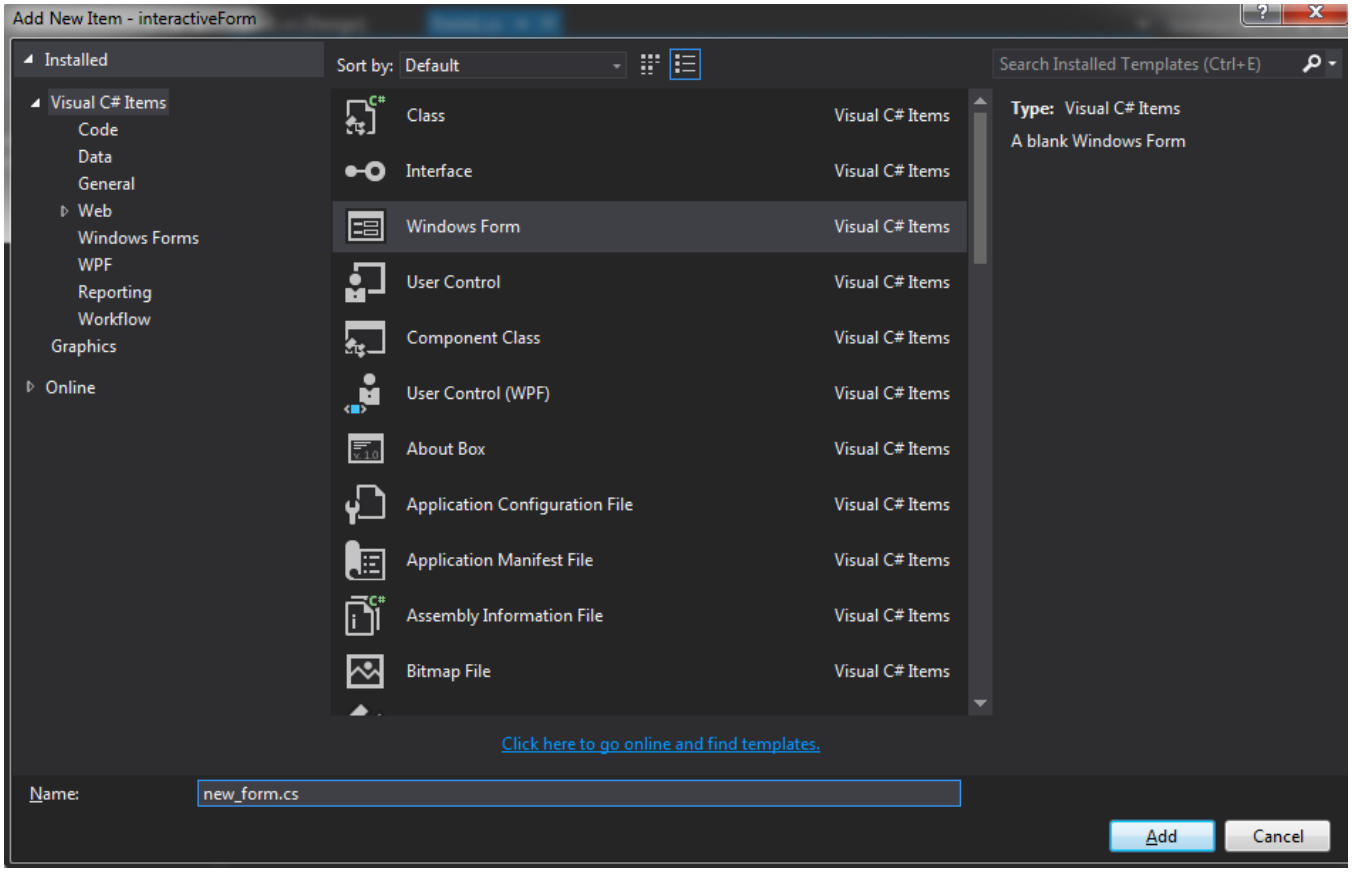


Εικόνα 9 Δημιουργία Click Event για υπομενού MenuStrip

Η συγκεκριμένη εφαρμογή αποτελεί παράδειγμα κατά το οποίο παρουσιάζεται το άνοιγμα μιας δεύτερης φόρμας, όταν πραγματοποιείται το πάτημα της υποκαρτέλας «Άνοιγμα...» που ανήκει στην

καρτέλα «Αρχείο».

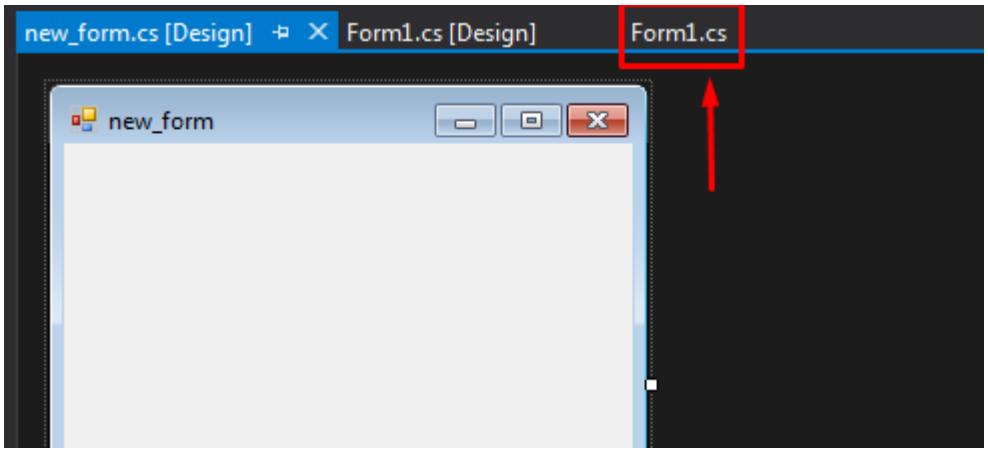
Στο σημείο αυτό λοιπόν, θα προσθέσουμε τη δεύτερη αυτή, νέα, φόρμα η οποία θα εμφανίζεται σύμφωνα με την παραπάνω διαδικασία. Επιλέγουμε από το μενού του Visual Studio, την επιλογή PROJECT και στη συνέχεια επιλέγουμε «Add Windows Form...». Στο νέο παράθυρο που θα εμφανιστεί Επιλεγουμε «Windows Form», ονομάζουμε τη νέα φόρμα όπως εμείς επιθυμούμε, και πατάμε το πλήκτρο «Add».



Εικόνα 10 Δημιουργία και εισαγωγή δεύτερου αρχείου φόρμας σε εφαρμογή

Το πρόγραμμα θα μας οδηγήσει πίσω στο project μας, όπου πλέον εμφανίζεται η νέα κενή φόρμα, με το όνομα που δώσαμε (new_form.cs) να αναγράφεται στη κύρια μπάρα του παραθύρου. Από τη στιγμή που δημιουργήσαμε τη νέα φόρμα, απομένει να εκτελέσουμε μερικές εντολές ώστε η τελευταία να εμφανίζεται κατά το πάτημα της υποκαρτέλας που βρίσκεται στην πρώτη φόρμα.

Μετακινούμαστε στην καρτέλα που περιέχει τον κώδικα της κύριας φόρμας κάνοντας «κλικ» στην καρτέλα «Form1.cs»

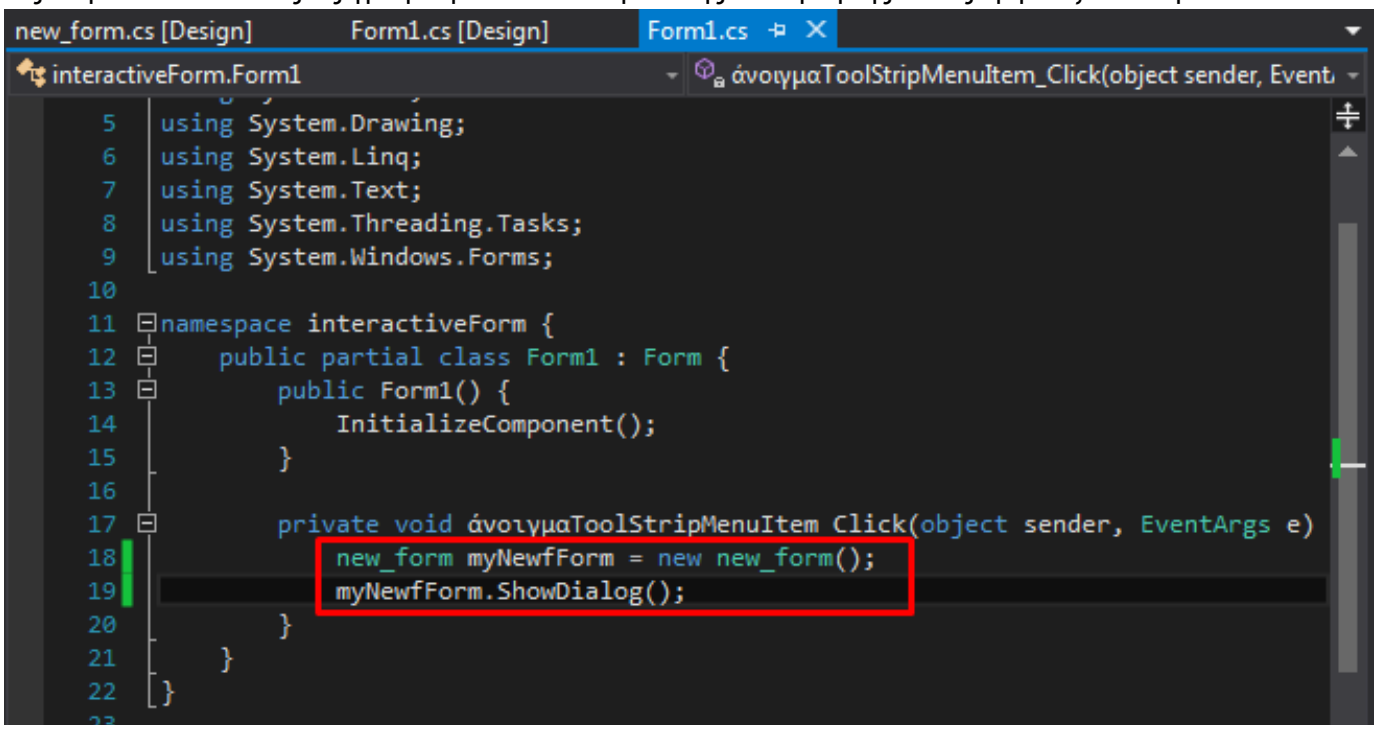


Εικόνα 11 Πλοήγηση στις καρτέλες αρχείων ενός Project

Οι εντολές που θα πρέπει να εκτελέσουμε ώστε να εμφανίζεται η φόρμα (new_form.cs), που δημιουργήσαμε, κάθε φορά που πατάμε στην επιλογή «Άνοιγμα...», είναι οι εξής δύο:

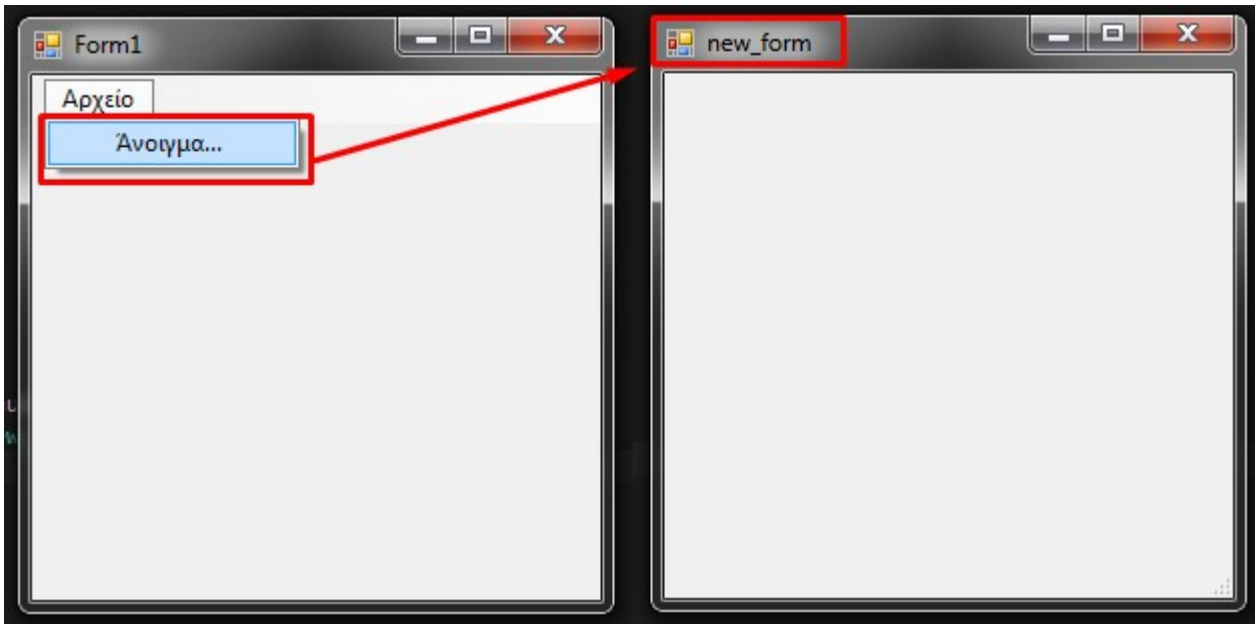
```
New_form myNewForm = new new_form();
myNewForm.ShowDialog();
```

Τις παραπάνω εντολές τις γράφουμε στο εσωτερικό της συνάρτησης όπως εμφανίζεται παρακάτω:



Εικόνα 12 Εισαγωγή κώδικα για τη δυναμική δημιουργία και εμφάνιση καινούριας φόρμας

Έχοντας πραγματοποιήσει το προηγούμενο βήμα, μπορούμε πλέον να «τρέξουμε» την εφαρμογή μας (από το επάνω μέρος του προγράμματος -> “Debug” -> “Start Debugging”). Το αποτέλεσμα θα πρέπει να είναι όμοιο με αυτό που εμφανίζεται στην εικόνα:



Εικόνα 13 Αποτέλεσμα εκτέλεσης κώδικα για την δημιουργία και εμφάνιση καινούριας φόρμας

3. ΠΩΣ ΑΝΑΠΤΥΧΘΗΚΕ Η ΕΦΑΡΜΟΓΗ

Έχοντας λοιπόν παρουσιάσει τις βασικές αρχές του Visual Studio 2013 αλλά και της γλώσσας Visual C#, μπορούμε να προχωρήσουμε στην ανάλυση της δομής και στις βασικές αρχές της εφαρμογής. Η πολυπλοκότητα του κώδικα, μας οδήγησε στο διαχωρισμό του σε επιμέρους τμήματα και ομαδοποιήσεις, που το καθένα εξυπηρετεί διαφορετικό σκοπό.

3.1. Classes

Η κύρια class ονομάζεται “Vehicle.cs” και σε αυτήν περιέχονται τα εξής χαρακτηριστικά και οι ιδιότητες του αντικειμένου (object):

- το εικονίδιο και το μέγεθός του
- το σημείο εκκίνησής του
- το ID του
- η κατάστασή του (απασχολημένο / διαθέσιμο)
- η τρέχουσα τοποθεσία του
- η τοποθεσία του φορτίου προς παραλαβή
- η διαδρομή που θα ακολουθήσει και όλα τα ενδιάμεσα σημεία
- οι απαραίτητες συναρτήσεις δόμησης (constructor)

Έπειτα, ορίσαμε μια 2η class με όνομα “constants.cs”, στην οποία περιέχονται κάποιες σταθερές τιμές που οριοθετούν τις παραμέτρους σχεδίασης. Η επιλογή των συγκεκριμένων τιμών που ορίσαμε, έγινε με σκοπό την επίδειξη.

Η κύρια φόρμα της εφαρμογής ονομάζεται “main_form.cs” και περιλαμβάνει τις θεμελιώδεις λειτουργίες και κανόνες, στοιχεία πάνω στα οποία βασίζεται η προσομοίωση και η σχεδίαση του παραθυρικού περιβάλλοντος. Για την οργάνωση του κώδικα, η παραπάνω φόρμα, έχει διαμεριστεί στα παρακάτω επιμέρους αρχεία κειμένου, τα οποία ανήκουν στη σκοπιά της main_form.cs:

- variables.cs
- functions.cs
- timers.cs

3.1.1. Variables.cs

Το αρχείο περιέχει όλες τις global μεταβλητές που χρησιμοποιούνται στο πρόγραμμα. Στο εσωτερικό του, πραγματοποιείται η αρχικοποίηση συγκεκριμένων μεταβλητών και τάξεων (classes), που σκοπό έχει την αποφυγή πιθανών προγραμματιστικών προβλημάτων (memory leaks, crashes) κατά την εκτέλεση της εφαρμογής. Ακόμη, ορίζονται και αρχικοποιούνται κάποιες ειδικού τύπου μεταβλητές (enum) οι οποίες εξασφαλίζουν την ομαλή λειτουργία κάποιων αντικειμένων του προγράμματος.

3.1.2. Functions.cs

Το αρχείο περιλαμβάνει όλες τις βασικές συναρτήσεις που εξυπηρετούν τη σωστή λειτουργία των events της κυρίως φόρμας. Ειδικότερα, περιέχονται συναρτήσεις που:

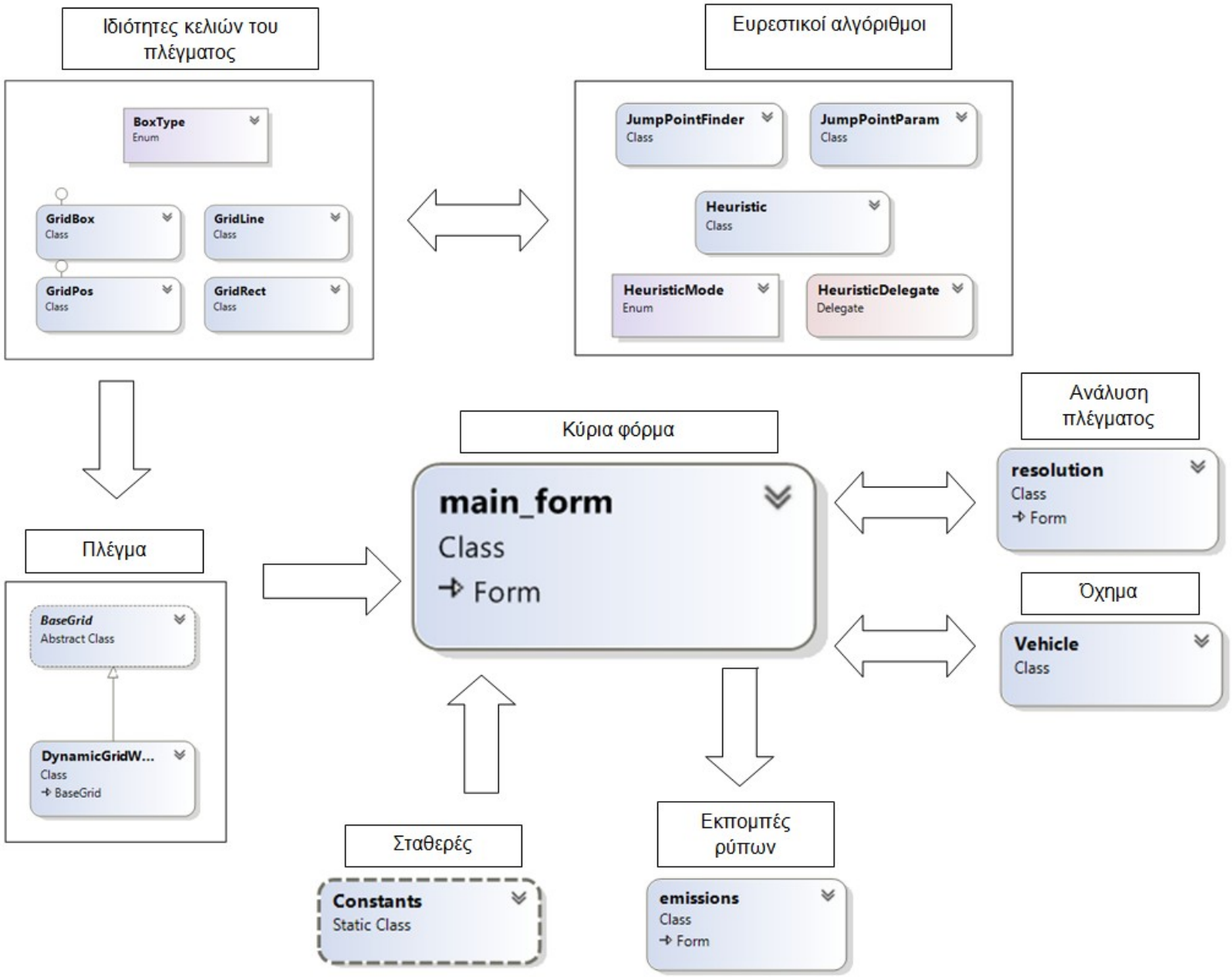
1. πραγματοποιούν την εύρεση της βέλτιστης διαδρομής για κάθε όχημα
2. υπολογίζουν και αποθηκεύουν τα ενδιάμεσα σημεία
3. αξιοποιούν τα δεδομένα μέσω μαθηματικών συναρτήσεων
4. πραγματοποιούν τη βηματική απεικόνιση της προσομοίωσης,
5. παρέχουν στο χρήστη πληροφορίες σχετικά με την κατάσταση της διεργασίας, κατά τη διάρκεια εκτέλεσής της

Επειδή η εφαρμογή υποστηρίζει την ταυτόχρονη κίνηση μέχρι και 5 οχημάτων, η ανάπτυξη βοηθητικών συναρτήσεων κρίθηκε αναγκαία. Συναρτήσεις τέτοιου τύπου, λοιπόν, αφορούν κανόνες προτεραιότητας και ιεραρχίας ανάμεσα στα οχήματα, όπως επίσης και πρόβλεψη σεναρίων κατά τα οποία το εκάστοτε όχημα αδυνατεί να φέρει εις πέρας την αποστολή που του έχει ανατεθεί.

Τέλος, περιέχονται οι απαραίτητες συναρτήσεις που εξυπηρετούν δευτερεύουσες λειτουργίες, όπως η πλοήγηση του χρήστη μέσω πλήκτρων συντόμευσης, η επανεκκίνηση του προγράμματος και η εισαγωγή/εξαγωγή προσχεδιασμένου, σε ειδικού τύπου αρχείο, layout εργασίας.

3.1.3. Timers.cs

Το συγκεκριμένο αρχείο περιέχει τα events των timers, οι οποίοι χειρίζονται τις συναρτήσεις που είναι “υπεύθυνες” για τη βηματική απεικόνιση της προσομοίωσης και τα απαραίτητα, για την πραγματοποίηση αυτής, δεδομένα. Το πλήθος των events βρίσκεται σε πλήρη αντιστοιχία με το πλήθος των αυτοοδηγούμενων οχημάτων που υποστηρίζεται από την εφαρμογή.



Εικόνα 14 Δομή εφαρμογής – σχηματική απεικόνιση

3.2. Επικοινωνία κλάσεων

Το πρώτο και πιο χαμηλό επίπεδο της εφαρμογής είναι οι ιδιότητες των κελιών που αποτελούν το πλέγμα. Οπτικά, ο χρήστης αντιλαμβάνεται το πλέγμα ως μία ενιαία οντότητα την οποία θεωρεί πως έχει δημιουργηθεί από κάθετες και οριζόντιες ευθείες. Στην πραγματικότητα, όμως, το πλέγμα αποτελεί μια αυτόνομη και ξεχωριστή ύπαρξη, ανεξάρτητη της κύριας φόρμας, η οποία ενσωματώνει πολλά επιμέρους αντικείμενα, τα κελιά (GridBox). Προγραμματιστικά λοιπόν, για να επιτευχθεί αυτό, έπρεπε να δημιουργηθεί ένα νέο επίπεδο (layer), το οποίο «αιωρείται» πάνω από την κύρια φόρμα, και πάνω σε αυτό να σχεδιάζεται το βασικό πλέγμα (BaseGrid). Το πλέγμα αυτό, αποτελείται από διαδοχικά κάθετα και οριζόντια κελιά, σε πλήρη αντίθεση με τα «παραδοσιακά» πλέγματα που διαχωρίζουν το χώρο με γραμμές. Κάθε ένα από τα παραπάνω κελιά, θεωρείται ένα μοναδικό αντικείμενο (class object). Γίνεται, λοιπόν, εύκολα αντιληπτό ότι από τη σκοπιά του προγραμματισμού, το πλέγμα αποτελεί έναν πίνακα πινάκων(m_rectangles[ ][ ]) που περιέχει στοιχεία τύπου GridBox. Αυτό παρέχει το πλεονέκτημα της ευελιξίας καθώς είναι εύκολο να δοθούν ξεχωριστές ιδιότητες σε κάθε κελί του πλέγματος.

```
m_rectangles[widthTrav][heightTrav] =
new GridBox(
                widthTrav * Constants.__BlockSide,
                heightTrav * Constants.__BlockSide + Constants.__TopBarOffset,
BoxType.Normal
            );
```

3.2.1. Ιδιότητες κελιών του πλέγματος

Όπως παρατηρούμε και στην παραπάνω εικόνα, αυτή η ομάδα κλάσεων περιέχει ακόμη τέσσερα (4) classes πέραν του GridBox.

Το BoxType είναι μία enumeration ιδιότητα της τάξης GridBox η οποία προσδιορίζει τον τύπο του κελιού που προκαλεί την κλήση του. Οι τύποι κελιών που υποστηρίζονται είναι οι εξής πέντε (5)

1. Start, Κελί εκκίνησης των οχημάτων
2. End, Κελί τερματισμού
3. Wall, Κελί εμποδίου
4. Normal, Κελί ελεύθερης κίνησης
5. Load, Κελί φορτίου

Το GridRect είναι η ιδιότητα του κελιού ως γραφικό αντικείμενο (Graphics.Rectangle() class). Αυτό εξυπηρετεί στην επίλυση προβλημάτων που αφορούν σχεδιαστικά ζητήματα, όπως τον τρόπο με τον οποίο σχεδιάζεται η κάθε ευθεία της διαδρομής.

Το class GridLine περιέχει, ως ιδιότητες, τέσσερις (4) ακέραιους αριθμούς που δηλώνουν δύο (2) ζεύγη συντεταγμένων. Πιο αναλυτικά, ενσωματώνει τις ιδιότητες “fromX-fromY” που αφορούν τις συντεταγμένες ενός αρχικού σημείου Α, και τις ιδιότητες “toX – toY” οι οποίες δηλώνουν ένα τελικό σημείο Β. Τα δύο (2) παραπάνω ζεύγη δηλώνονται σε αυτό το class και έπειτα χρησιμοποιούνται για το σχεδιασμό της ευθείας, που ενώνει τα παραπάνω ζεύγη συντεταγμένων, με διεύθυνση από το σημείο Α στο Β.

Οι συντεταγμένες, που αποθηκεύονται, και στη συνέχεια αξιοποιούνται κατά τους υπολογισμούς της βέλτιστης διαδρομής, είναι τύπου GridPos. Το class αυτό χρησιμοποιείται για τη δήλωση ζευγών συντεταγμένων, με μορφή συμβατή με τις συναρτήσεις που ανταλλάσουν δεδομένα για την εμφάνιση της τελικής διαδρομής.

3.2.2. Ευρετικοί αλγόριθμοι

Το προηγούμενο πακέτο κλάσεων συνεργάζεται με τους αλγορίθμους εύρεσης της βέλτιστης διαδρομής, στέλνοντας στη συνάρτηση “Reset” του class “JumpPointParam” τις, τύπου GridPos, τιμές του αρχικού σημείου Α και του τελικού σημείου Β. Οι τιμές αυτές ανάλογα με την εκάστοτε περίπτωση θα αποτελούν είτε ζεύγη συντεταγμένων «Οχήματος – Φορτίου», είτε «Φορτίου – Εξόδου» είτε ακόμη «Οχήματος – Εξόδου».

- jumpParam.Reset(StartPos[pos_index], loadPos[0]);
- jumpParam.Reset(loadPos[0], endPos);
- jumpParam.Reset(StartPos[pos_index], endPos);

Λαμβάνοντας, βέβαια, υπόψητη διαδικασία που ακολουθείται, κατά την οποία ένα φορτίο ζητάει από τα οχήματα να το παραλάβουν, έπρεπε να προβλεφθεί η περίπτωση που το αιτούμενο παραλαβής φορτίο είναι «παγιδευμένο». Για την ανάγκη αποτροπής του σεναρίου που το φορτίο αυτό καλεί κάποιο όχημα για να το παραλάβει, αναπτύχθηκε μία συνάρτηση η οποία επεξεργάζεται τη Λίστα που περιέχει όλα τα φορτία, ένα προς ένα, και ελέγχει ποια από αυτά είναι προσβάσιμα και ποια όχι. Τα φορτία, για τα οποία δεν είναι δυνατό να υπολογιστεί διαδρομή, μαρκάρονται ως μη διαθέσιμα, τους δίνεται η τιμή «4» ως δείκτης, και αφαιρούνται από την παραπάνω Λίστα φορτίων. Τη συνάρτηση που πραγματοποιεί την παραπάνω διαδικασία, μπορούμε να δούμε και στο παρακάτω σχήμα:

```
private void checkForTrappedLoads(List<GridPos> pos) {
    int list_index = 0;
    bool removed;

    //if the 1st AGV  cannot reach a Load, then that Load is
    //removed from the loadPos and not considered as available - marked as "4"
    do
    {
        removed = false;
        searchGrid.SetWalkableAt(new GridPos(pos[list_index].x, pos[list_index].y), true);
        jumpParam.Reset(StartPos[0], pos[list_index]);
        if (JumpPointFinder.FindPath(jumpParam, paper).Count == 0)
        {
            searchGrid.SetWalkableAt(new GridPos(pos[list_index].x, pos[list_index].y), false);
            isLoad[pos[list_index].x, pos[list_index].y] = 4;
            pos.Remove(pos[list_index]);
            removed = true;
        }
        else
        {
            isLoad[pos[list_index].x, pos[list_index].y] = 1;
        }

        if (!removed)
            list_index++;
    } while (list_index < pos.Count);

}
```

Εικόνα 15 Συνάρτηση υπολογισμού μη παγιδευμένων φορτίων

3.2.3. Κύρια φόρμα

Στη συνέχεια, το αποτέλεσμα, που επιστρέφει η συνάρτηση “jumpParam.Reset()”, που κλήθηκε παραπάνω, δίνεται ως όρισμα στη συνάρτηση “FindPath” που ανήκει στο class “JumpPointFinder”. Η κλήση της “FindPath” θα επιστρέψει τα σημεία καμπής (JumpPoints), τα οποία με τη σειρά τους αποθηκεύονται σε μια Λίστα, τύπου GridPos.

```
List<GridPos> JumpPointsList = new List<GridPos>();

JumpPointsList = JumpPointFinder.FindPath(jumpParam, paper);
```

Η διαδικασία αυτή είναι επαναλαμβανόμενη και συνεχίζει μέχρι να πραγματοποιηθεί ο υπολογισμός των σημείων καμπής της διαδρομής του κάθε οχήματος. Μόλις ολοκληρωθεί αυτή η διαδικασία, το αποτέλεσμα των παραπάνω υπολογισμών θα αποθηκευτεί σε μία αντίστοιχη, τύπου GridPos, Λίστα που θα ανήκει σε κάθε όχημα, ξεχωριστά.

```
for (int j = 0; j < JumpPointsList.Count; j++) {
AGVs[i].JumpPoints.Add(JumpPointsList[j]);
}
```

Έπειτα, αναπτύχθηκε η συνάρτηση “DrawPoints” η οποία μετατρέπει τη Λίστα σημείων καμπής σε πίνακα βημάτων που περιέχει όλα τα ενδιάμεσα σημεία της διαδρομής.

```
for (int k = 0; k < Constants.__WidthBlocks; k++) {

    for (int l = 0; l < Constants.__HeightBlocks; l++) {

        if (m_rectangles[k][l].boxRec.Contains(_p)) {
            //+9 is the width/2 - handling boxes from their centre
            int sideX = m_rectangles[k][l].boxRec.X + ((Constants.__BlockSide / 2) - 1);
            int sideY = m_rectangles[k][l].boxRec.Y + ((Constants.__BlockSide / 2) - 1);
            currentLinePoints[i].X = sideX;
            currentLinePoints[i].Y = sideY;

            if (dotsToolStripMenuItem.Checked)...

            using (Font stepFont = new Font("Tahoma", 8, FontStyle.Bold))...
            calibrated = true;

        }

    }

}

if (calibrated) {
    AGVs[agv_index].Steps[AGVs[agv_index].StepsCounter].X = currentLinePoints[i].X;
    AGVs[agv_index].Steps[AGVs[agv_index].StepsCounter].Y = currentLinePoints[i].Y;
    AGVs[agv_index].StepsCounter++;
}
```

Εικόνα 16 Εισαγωγή σε πίνακα όλων των σημείων που ανήκουν στη βέλτιστη διαδρομή του κάθε οχήματος

Ο παραπάνω πίνακας ανήκει, όπως και η Λίστα, στο class του οχήματος και είναι απαραίτητος για τη διαδικασία της γραφικής προσομοίωσης.

3.2.4. Όχημα

```
//****************************************
//AGV Steps
internal class AGVSteps {
    public double X { get; set; }
    public double Y { get; set; }
}

private AGVSteps[] steps;
public AGVSteps[] Steps {
    get { return this.steps; }
}
//========================================
```

Εικόνα 17 Επιμέρους ιδιότητες του class οχήματος.

Το class του οχήματος, περιέχει επίσης και κάποια ακόμα στοιχεία που αποτελούν ιδιότητές του και τού δίνουν υπόσταση. Στοιχεία τέτοια είναι η τρέχουσα τοποθεσία και κατάστασή του, τα οποία χρησιμοποιούνται για την παρακολούθησή του μέσω του Monitoring Panel καθώς επίσης και όλα τα γραφιστικά αντικείμενά του, απαραίτητα για τη διαδικασία της προσομοίωσης.

3.2.5. Πλέγμα

Τα κελιά μαζί με τις ιδιότητές τους, σχηματίζουν το πλέγμα το οποίο βλέπει ο χρήστης. Οι αλγόριθμοι αναζήτησης, αλλά και οι οντότητες του GridPos και GridBox, αποστέλνουν δεδομένα στο class DynamicGridWPool. Το class αυτό περιέχει όλα τα έγκυρα σημεία της βέλτιστης διαδρομής, τα οποία στη συνέχεια αποστέλει στο BaseGrid. Τελικά, το ολοκληρωμένο πλέγμα, με τις αντίστοιχες ευθείες βέλτιστης διαδρομής, εμφανίζονται στην κύρια φόρμα.

3.2.6. Σταθερές

Για να εκτελεστούν με επιτυχία όλα τα παραπάνω χρειαζόμαστε τις προκαθορισμένες σταθερές τιμές, τις οποίες ο χρήστης αρχικοποιεί ανάλογα με τις προσωπικές του προτιμήσεις. Οι τιμές αυτές βρίσκονται μέσα στο class Constants και περιέχει τις παρακάτω μεταβλητές:

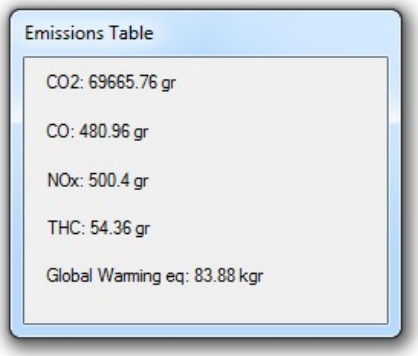
1. public const int __MaximumSteps, Ο μέγιστος αριθμός βημάτων ανά γραμμή
2. public const int __TopBarOffset, Η μετατόπιση του πλέγματος από τη μπάρα του μενού
3. public const int __BottomBarOffset, Η μετατόπιση του πλέγματος από την κάτω μπάρα της φόρμας

4. public static int __MaximumAGVs, Ο μέγιστος αριθμός των οχημάτων που υποστηρίζει η εφαρμογή
5. public static int __WidthBlocks, Το πλήθος κελιών σε μήκος
6. public static int __HeightBlocks, Το πλήθος κελιών σε ύψος
7. public static int __BlockSide, Το μέγεθος, σε pixel, του κάθε κελιού
8. public static int __ResolutionMultiplier, Ο πολλαπλασιαστής της ανάλυσης

Τέλος, υπάρχουν δύο ακόμη φόρμες.

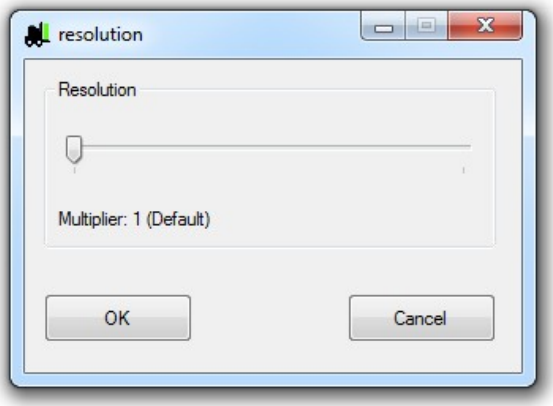
3.2.7. Εκπομπές ρύπων

Η φόρμα των εκπομπών παρουσιάζεται κατά την εκκίνηση της προσομοίωσης και ανανεώνεται καθώς εξελίσσεται η διεργασία, με την κλήση της συνάρτησης “update_emissions”. Τα στοιχεία των εκπομπών αλλάζουν και τελικά ο χρήστης αποκτά μία εικόνα σχετικά με τις συνολικές εκπομπές.



Εικόνα 18 Φόρμα εμφάνισης συνολικών εκπομπών ρύπων

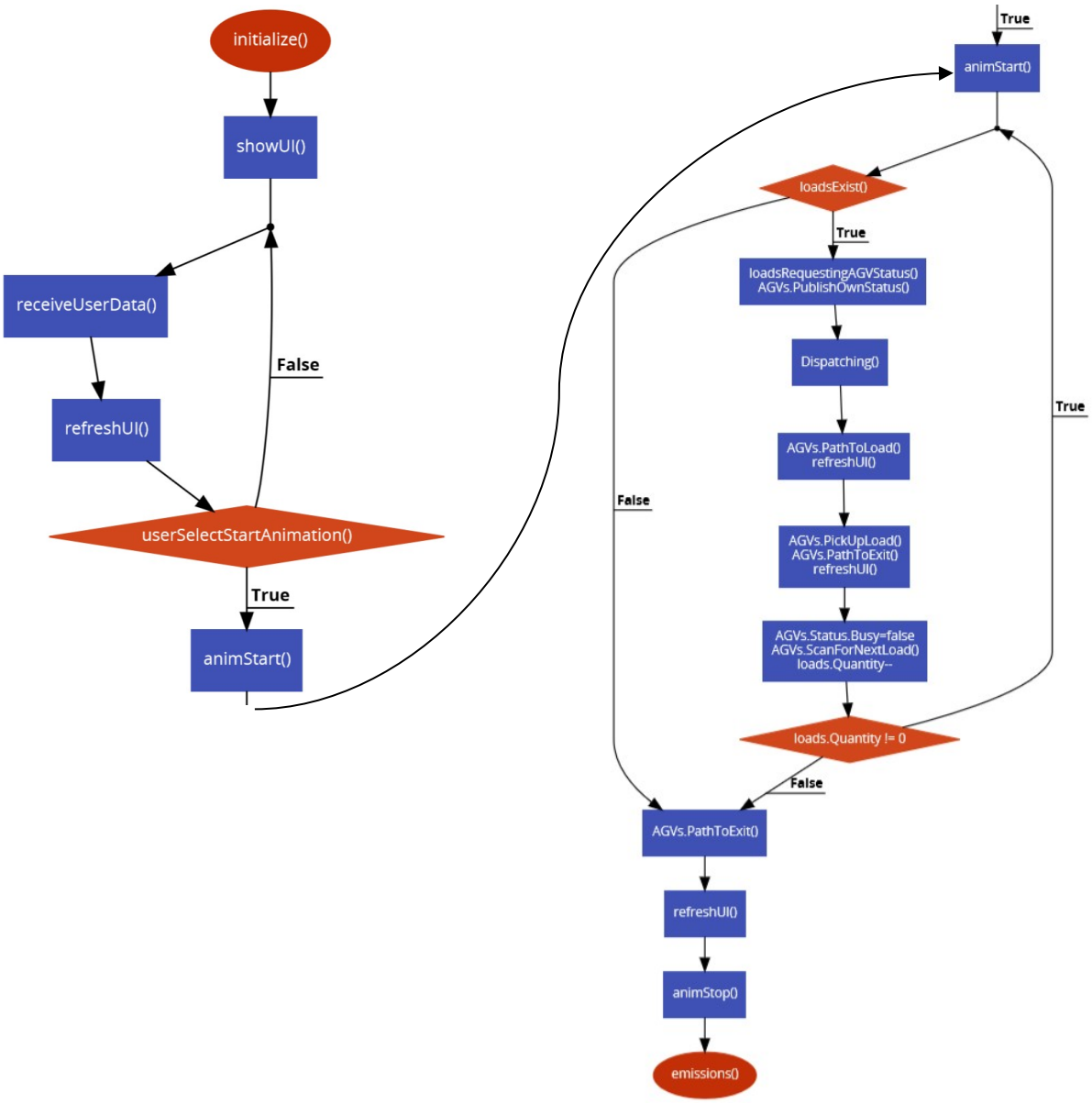
3.2.8. Ανάλυση πλέγματος



Ο χρήστης έχει την επιλογή να παραμετροποιήσει την ανάλυση του πλέγματος μέσω του χειριστηρίου που παρέχει η παρακάτω φόρμα. Μετακινώντας το συρώμενο δείκτη της μπάρας που εμφανίζεται στο κέντρο του αντίστοιχου παραθύρου μπορεί να αυξήσει την ανάλυση του Grid ή να την επαναφέρει στην αρχική της τιμή.

Εικόνα 19 Φόρμα παραμετροποίησης της ανάλυσης πλέγματος

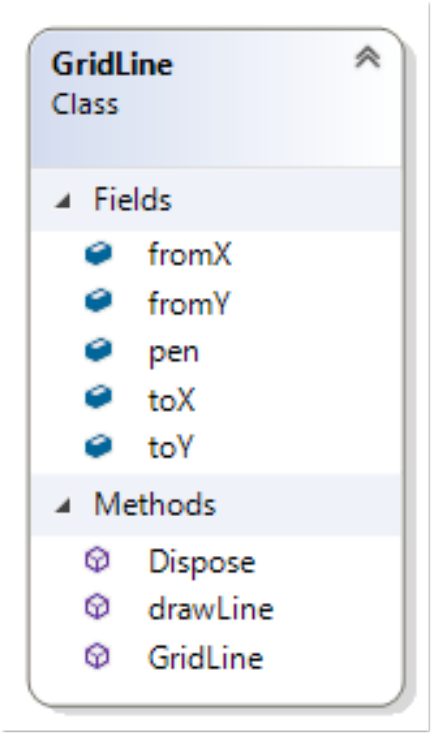
Η ανάλυση του τρόπου λειτουργίας της εφαρμογής, που έγινε παραπάνω, μπορεί να εξηγηθεί και βηματικά υπό τη μορφή διαγράμματος ροής, κάνοντας τη διαδικασία κατανόησής της αλλά και της αρχιτεκτονικής της, αισθητά πιο εύκολη.



Εικόνα 20 Διάγραμμα ροής της εφαρμογής, υπό μορφή ψευτοκώδικα

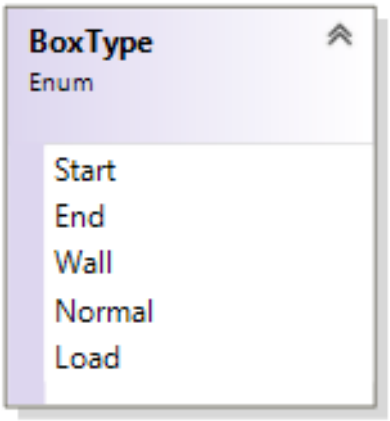
3.3. Περιεχόμενο κλάσεων

Το Visual Studio έχει τη δυνατότητα να εμφανίζει οπτικά, με τη μορφή διαγραμμάτων, τις κλάσεις, με τρόπο που γίνεται ευκολότερα κατανοητή η δομή τους. Παρακάτω, θα αναφερθούμε σε πολύ συγκεκριμένες κλάσεις, που συμμετέχουν στη διαδικασία που περιγράψαμε προηγουμένως, παραθέτοντας εικόνες που συγκεντρώνουν το περιεχόμενο της καθεμίας. Στο πεδίο “Fields” ανήκουν οι μεταβλητές του εκάστοτε class ενώ στο πεδίο “Methods” εμφανίζονται οι συναρτήσεις. Για λόγους απλούστευσης, εμφανίζουμε μόνο τα στοιχεία που δημιουργήσαμε εμείς (μεταβλητές & συναρτήσεις), καθώς οι ιδιότητες των αντικειμένων που χρησιμοποιεί η εφαρμογή, μπορούν να θεωρηθούν ως μεταβλητές και, αντίστοιχα, τα γεγονότα (events) θεωρούνται συναρτήσεις. Αυτό έχει ως αποτέλεσμα τα διαγράμματα να γίνονται πολύπλοκα και να μην είναι ξεκάθαρο το πλήθος των συναρτήσεων που αναπτύχθηκαν και τον μεταβλητών που χρησιμοποιήθηκαν, κατά την δόμηση της εφαρμογής.



Εικόνα 21 Δομή της κλάσης GridLine

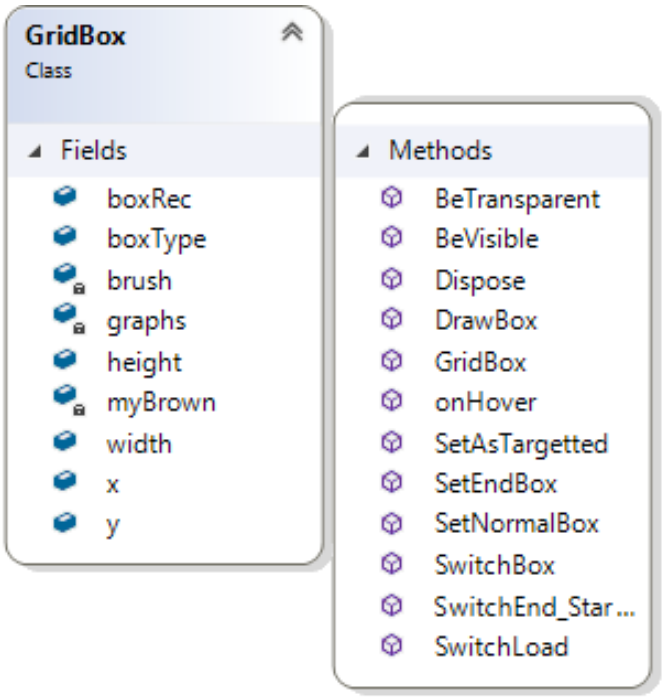
Η κλάση GridLine συμμετέχει στην χάραξη της ευθείας ανάμεσα σε δύο (2) σημεία. Περιέχει ιδιότητες που δηλώνουν ζεύγη συντεταγμένων, τόσο του αρχικού σημείου (“fromX – fromY”) όσο και του τελικού σημείου (“toX – toY”). Η ιδιότητα “pen” εξυπηρετεί τη σχεδίαση της γραμμής που ενώνει τα παραπάνω σημεία, μέσω της συνάρτησης “drawLine( )”.



Εικόνα 22 Δομή της κλάσης BoxType

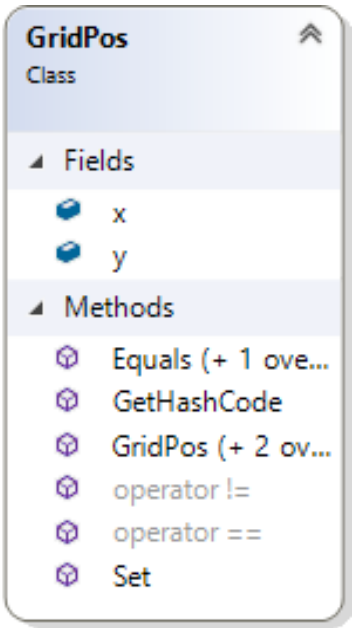
Στην περιγραφή του class GridBox, παρακάτω, θα δούμε πως ο τύπος του κάθε κελιού προσδιορίζεται από την ενσωματωμένη ιδιότητα “BoxType”. Στο σχήμα, λοιπόν, παρατηρούμε την ανάπτυξη του συγκεκριμένου μέλους της κλάσης “GridBox” που περιέχει τους πέντε (5) τύπους κελιών που υποστηρίζει η εφαρμογή.

Η κλάση GridBox δίνει υπόσταση σε κάθε κελί του πλέγματος, καθώς περιέχει τη βασική ιδιότητα “boxRec” μέσω της οποίας γίνεται η δήλωση του κάθε κελιού, ως αντικείμενο, και περιλαμβάνει τις διαστάσεις του αλλά και τις συντεταγμένες του επάνω στο πλέγμα. Τον τύπο του κάθε κελιού προσδιορίζει η ιδιότητα που περιγράψαμε αμέσως προηγουμένως, “boxType”.



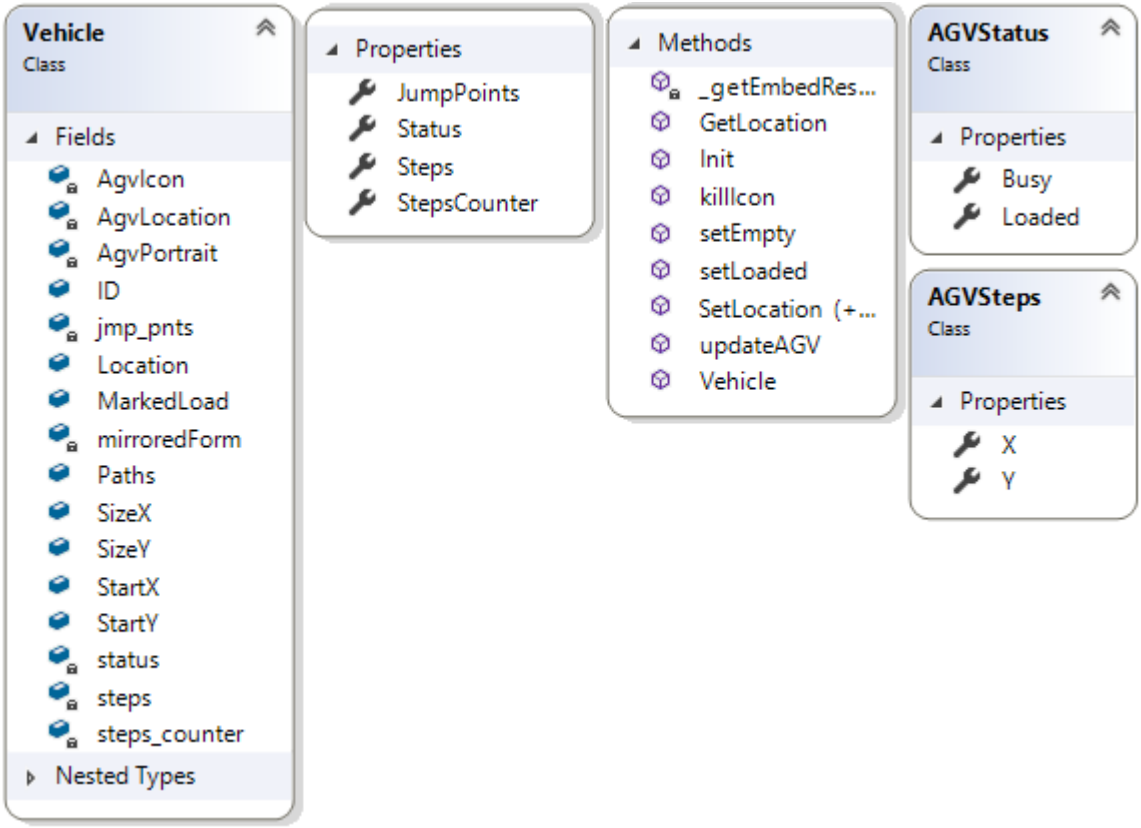
Εικόνα 23 Δομή της κλάσης GridBox

Η συνάρτηση “SetAsTargetted( )” είναι υπεύθυνη για την αλλαγή χρώματος ενός κελιού-φορτίου που έχει θέσει τον εαυτό του προς παραλαβή από κάποιο όχημα. Περιέχονται επίσης λοιπές συναρτήσεις εναλλαγής τύπου κελιών από “Normal” σε “Wall”, και αντίστροφα, από “Φορτίο” σε “Normal”, και αντίστροφα, αλλά και συναρτήσεις χρωματισμού του κελιού πάνω από το οποίο έχει τοποθετήσει ο χρήστης τοποντίκι.



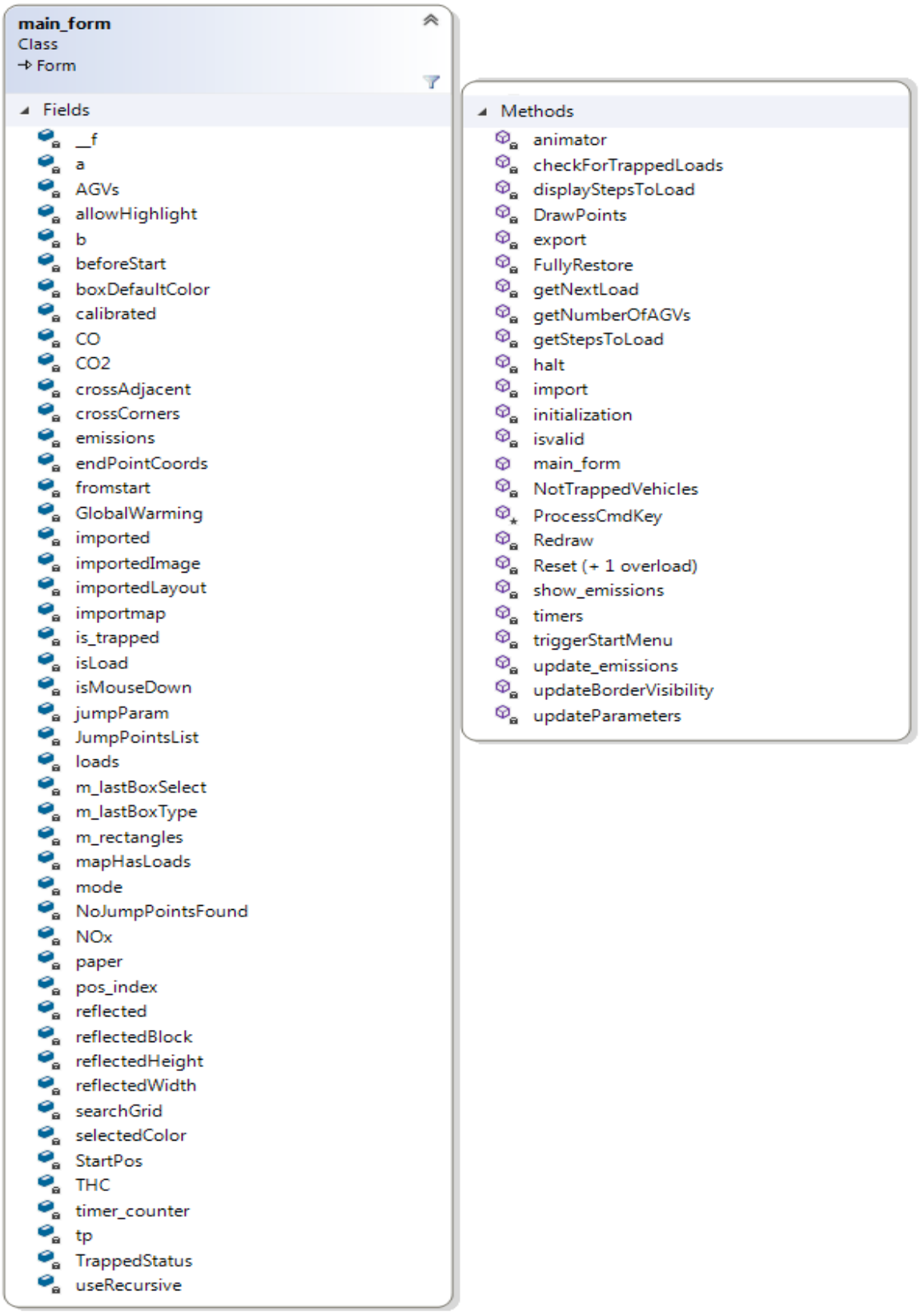
Εικόνα 24 Δομή της κλάσης GridPos

Όπως έχει ήδη αναφερθεί για το Class GridPos, η δομή του είναι όμοια με το class “Point” που παρέχει η Visual C#. Και οι δύο κλάσεις αποτελούνται από τις δύο (2) ιδιότητες “x-y” και διαμορφώνουν ένα ζεύγος συντεταγμένων. Ο λόγος που συντάχθηκε και προτιμήθηκε το συγκεκριμένο class, οφείλεται στην ανάγκη συμβατότητας με τις υπόλοιπες κλάσεις της εφαρμογής, με τις οποίες συνεργάζεται.



Εικόνα 25 Δομή της κλάσης Vehicle (όχημα)

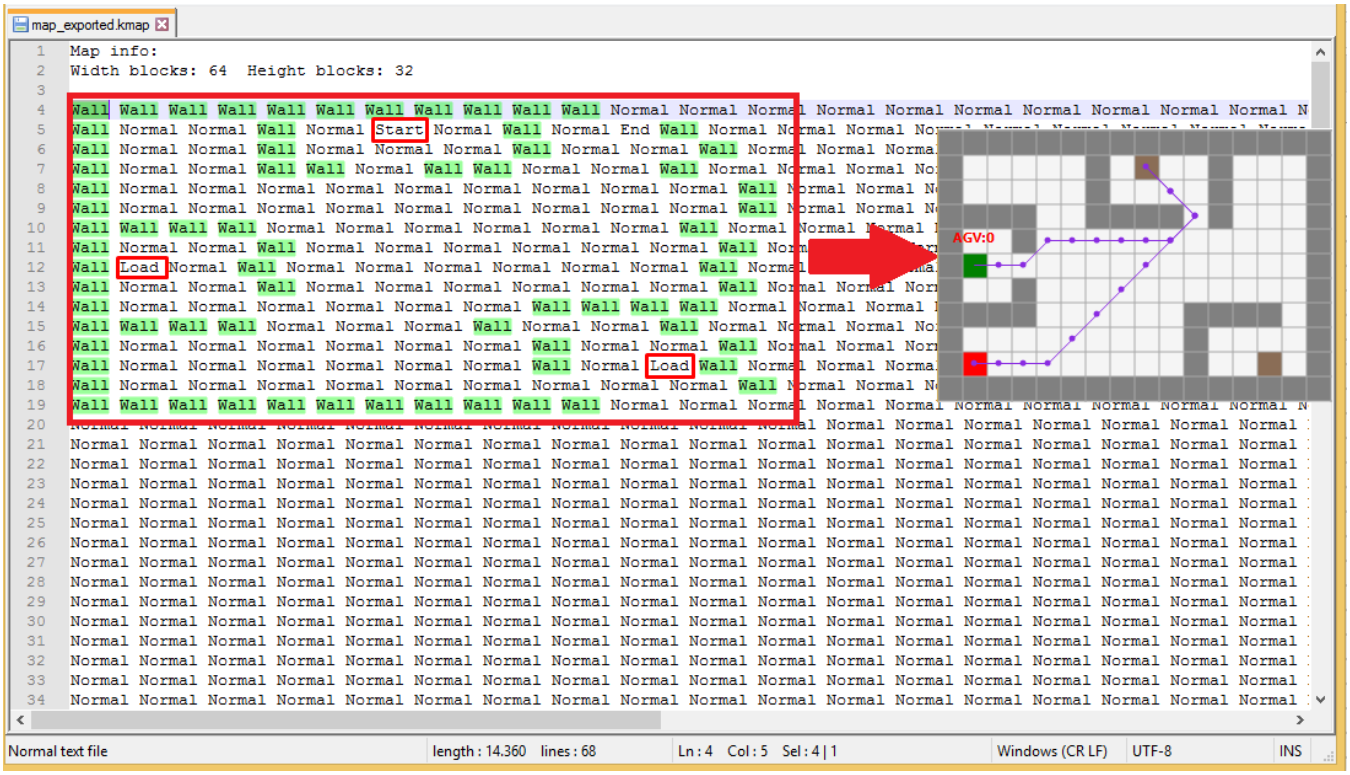
Η κατάσταση του οχήματος (AGVStatus) όπως και τα βήματα αυτού (AGVSteps), αποτελούν τις τελευταίες, έχοντας κάθε μία τις δικές τους ιδιότητες. Οι επιμέρους, αυτές, ιδιότητες είναι η κατάσταση “Busy” και “Loaded”, τύπου “bool”, για το εσωτερικό class “AGVStatus”. Αντίστοιχα, για την εσωτερική κλάση “AGVSteps”, ιδιότητες αποτελούν οι συντεταγμένες X και Y, τύπου “double”. Πέραν της παραπάνω ιδιαιτερότητας, η κλάση του οχήματος ακολουθεί τη δομή των προηγούμενων παραδειγμάτων, έχοντας τις δικές της ιδιότητες, μεταβλητές και συναρτήσεις. Οι “setLoaded( )” και “setEmpty( )” χειρίζονται το εικονίδιο που εμφανίζεται κατά την προσομοίωση. Για κάθε μία από τις δύο καταστάσεις του οχήματος (φορτωμένο ή μη), έχει δημιουργηθεί ένα αντίστοιχο εικονίδιο του οποίου η διαδρομή καταχωρείται, μέσω των δύο αυτών συναρτήσεων, στο αντικείμενο “AgvIcon”, που προηγουμένως έχουμε δηλώσει ως “PictureBox”. Κατά αυτόν τον τρόπο είναι εφικτό να αλλάζει το εικονίδιο του οχήματος, ανάλογα με την κατάστασή του, με την κλήση της αντίστοιχης συνάρτησης. Έπειτα, ο μηχανισμός μέσω του οποίου αλλάζει η τοποθεσία του οχήματος επάνω στο πλέγμα, όσο βρίσκεται σε εξέλιξη η προσομοίωση, εξαρτάται από τη συνάρτηση “SetLocation( )”. Η τελευταία, δέχεται ως όρισμα ένα ζεύγος συντεταγμένων είτε υπό τη μορφή μεταβλητής τύπου “Point” είτε ως δύο ξεχωριστούς ακεραίους.



Εικόνα 26 Δομή της κύριας φόρμας

Εν ολίγοις, η συνάρτηση αυτή είναι υπεύθυνη για τη διαδικασία της προσομοίωσης την οποία βλέπει ο χρήστης στην κύρια φόρμα. Παράλληλα, στο παρασκήνιο εκτελούνται διαρκώς νέοι υπολογισμοί εύρεσης διαδρομών για τα οχήματα, που πραγματοποιούνται μέσω της συνάρτησης “getNextLoad( )”, καθώς επίσης και έλεγχοι για την αποφυγή συγκρούσεων ανάμεσα στα οχήματα που κινούνται, καλώντας την συνάρτηση “halt( )”. Οι εκπομπές ρύπων εμφανίζονται σε ένα νέο παράθυρο μέσω της συνάρτησης “show_emissions( )” και ανανεώνονται με την κλήση της “update_emissions( )”, καθ’ όλη τη διάρκεια της προσομοίωσης. Η παρακολούθηση της εν εξελίξει διεργασίας γίνεται, όπως έχει ήδη προαναφερθεί, από το Monitor Panel στο οποίο εμφανίζονται τα δεδομένα, που υπολογίζει η “getStepsToLoad( )”, χρησιμοποιώντας τη συνάρτηση “displayStepsToLoad( )”.

Τέλος, οι λειτουργίες εισαγωγής και εξαγωγής χάρτη, υλοποιούνται από τις συναρτήσεις “import( )” και “export( )” αντίστοιχα. Όπως έχουμε προαναφέρει, στην ανάπτυξη του class “GridBox”, η ιδιότητα “boxType”, που περιέχεται σε αυτό, καθορίζει τον τύπο του κάθε κελιού. Η “export( )”, κατά συνέπεια, προσπελάζει το πλέγμα, κελί προς κελί, ελέγχει τον τύπο του καθενός και εξάγει αυτήν την πληροφορία σε ένα αρχείο τύπου “.kmap”. Κατά την ολοκλήρωση της διαδικασίας εξαγωγής, ο χάρτης έχει πλέον εξαχθεί υπό τη μορφή ψευτο-κειμένου που αποτελείται μόνο από λέξεις που δηλώνουν την ιδιότητα του κάθε “GridBox”.



Εικόνα 27 Μορφή εξαγωγής του σχεδιασμένου χάρτη

Αντίστοιχη λογική ακολουθεί και η συνάρτηση εισαγωγής, “import( )”. Κατά την εκτέλεσή της, αρχικά, γίνεται ανάγνωση των πληροφοριών που περιέχονται στην επικεφαλίδα του αρχείου, προκειμένου να καθοριστεί το μέγεθος του πλέγματος σύμφωνα με το οποίο χτίστηκε ο χάρτης. Η συγκεκριμένη πληροφορία είναι απαραίτητη ούτως ώστε να δηλωθεί ο ένας πίνακας, τύπου “BoxType”, του οποίου το μέγεθος πρέπει να ταυτίζεται με το μέγεθος του πλέγματος.

```
importmap = new BoxType[width_blocks, height_blocks];
```

Έπειτα ξεκινάει η διαδικασία ανάγνωσης, κατά την οποία η κάθε γραμμή διαβάζεται ξεχωριστά και αποθηκεύεται σε μία μεταβλητή. Στη συνέχεια, η παραπάνω μεταβλητή διαχωρίζεται, σύμφωνα με ένα διαχωριστή που εμείς έχουμε ορίσει, έτσι ώστε να εξαλειφθούν τα κενά που αυτή περιέχει και, εν τέλει, να αποκομιστούν οι ιδιότητες-λέξεις του κάθε κελιού που με τη σειρά τους καταχωρούνται σε ένα πίνακα “words[ ]” , τύπου “string”. Έχοντας λοιπόν, στην επιθυμητή μορφή, το περιεχόμενο της γραμμής που διαβάστηκε παραπάνω, συγκρίνουμε το κάθε στοιχείο του πίνακα “words[ ]” με τους πιθανούς τύπους κελιών και το αποθηκεύουμε σε ένα νέο πίνακα “importmap[,]”, τύπου “BoxType”. Στο τέλος της διαδικασίας ανάγνωσης του αρχείου-χάρτη, το περιεχόμενο αυτού έχει αποθηκευτεί στον παραπάνω πίνακα, από τον οποίο και θα εισαχθεί στο πλέγμα.

```
importmap = new BoxType[width_blocks, height_blocks];

words = reader.ReadLine().Split(delim);

int starts_counter = 0;
for (int z = 0; z < importmap.GetLength(0); z++) {
    int i = 0;
    foreach (string _s in words) {
        if (i < importmap.GetLength(1)) {
            if (_s == "Start")... else if (_s == "End")
                importmap[z, i] = BoxType.End;
            else if (_s == "Normal")
                importmap[z, i] = BoxType.Normal;
            else if (_s == "Wall")
                importmap[z, i] = BoxType.Wall;
            else if (_s == "Load")
                importmap[z, i] = BoxType.Load;
            i++;
        }
    }
    if (z == importmap.GetLength(0) - 1) { } else
        words = reader.ReadLine().Split(delim);
}
reader.Close();
```

(local variable) StreamReader reader

Εικόνα 28 Τμήμα κώδικα για την εισαγωγή χάρτη που έχει εξαχθεί μέσω της εφαρμογής

3.4. Forms

Αφού αναπτύξαμε το “παρασκήνιο” της εφαρμογής, στη συνέχεια ακολουθεί το “προσκήνιο” αυτής, εξετάζοντας τα επιμέρους παράθυρα και μενού.

3.4.1. main_form.cs

Η κυρίως φόρμα, όπως και ονομάζεται, αποτελεί το κεντρικό παράθυρο στο οποίο έχει πρόσβαση ο χρήστης κατά την εκτέλεση της εφαρμογής.

Στο παράθυρο αυτό βρίσκονται:

- η μπάρα επιλογών προτιμήσεων και παραμετροποίησης της εφαρμογής
- το πάνελ ρυθμίσεων της διαδικασίας προσομοίωσης
- το στατικό πλέγμα απεικόνισης της κάτοψης χώρου εργασίας (facility layout)

3.4.2. emissions.cs

Η δευτερεύουσα αυτή φόρμα, είναι διαθέσιμη στο χρήστη από τη στιγμή έναρξης της προσομοίωσης. Σκοπός αυτής είναι να παρέχει στο χρήστη πληροφορίες, σχετικές με τις εκπομπές ρύπων των οχημάτων που συμμετέχουν στη διεργασία, που στόχο έχουν να προσφέρουν στο χρήστη τη δυνατότητα λήψης αποφάσεων για πιθανή αντίστοιχη διεργασία σε πραγματικό περιβάλλον.

3.4.3. resolution.cs

Η φόρμα δίνει τη δυνατότητα στο χρήστη να μεταβάλλει την ανάλυση του πλέγματος. Οι δυνατές επιλογές που έχει ο χρήστης είναι η προεπιλεγμένη ανάλυση και η διπλάσιά της.

3.4.4. about.cs

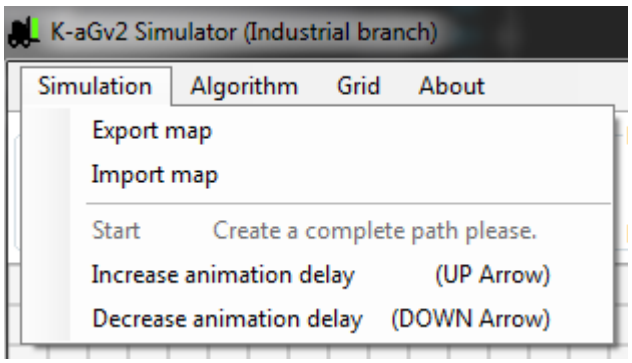
Το παράθυρο αυτό αποτελεί το καθιερωμένο παράθυρο “σχετικά με την εφαρμογή”, που περιέχει τα ονόματα των συνεισφερόντων καθώς επίσης και το ίδρυμα στο οποίο ανήκουν.

3.5. Menu

Η μπάρα επιλογών προτιμήσεων και παραμετροποίησης της εφαρμογής, που προαναφέρθηκε παραπάνω, περιλαμβάνει τα εξής υπομενού:

3.5.1. Simulation

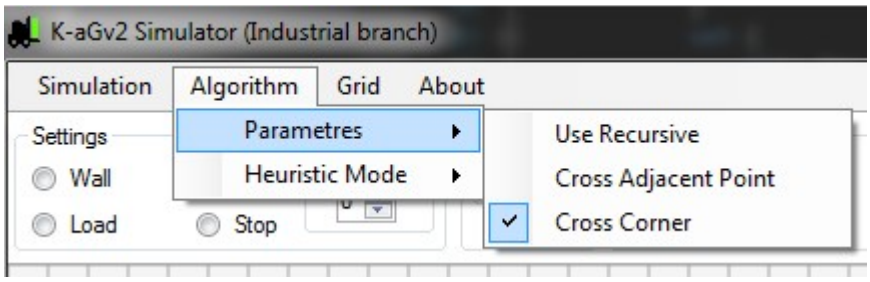
- Import map (εισαγωγή αρχείου χάρτη)
- Start (έναρξη προσομοίωσης)
- Increase animation delay (μείωση της ταχύτητας απεικόνισης)
- Decrease animation delay (αύξηση της ταχύτητας απεικόνισης)



Εικόνα 29 Simulation – Μενού και λειτουργίες

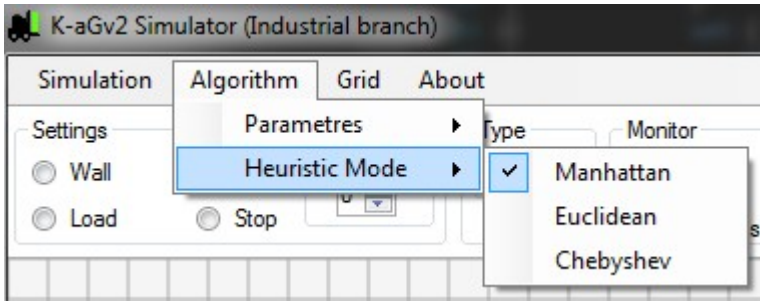
3.5.2. Algorithm

- Parameters (παράμετροι)
  - Use Recursive
  - Cross Adjacent Point
  - Cross Corner



Εικόνα 30 Parametres – Παράμετροι του Αλγορίθμου

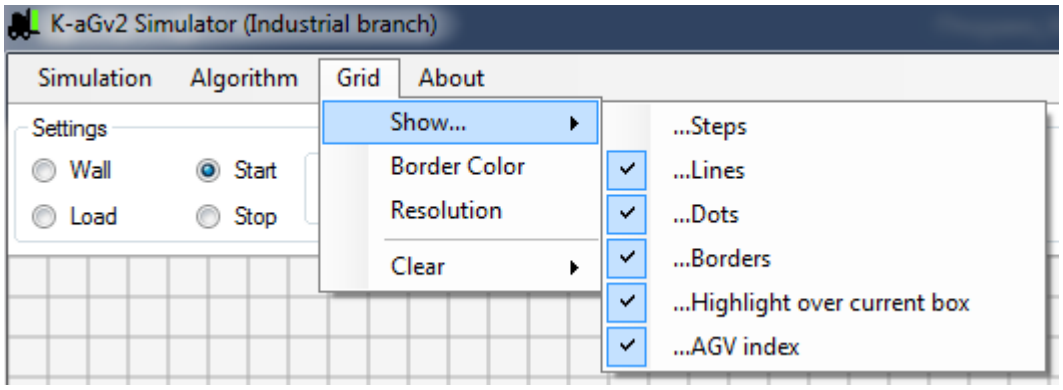
- Heuristic mode (ευρετική λειτουργία)
  - Manhattan
  - Euclidean
  - Chebyshev



Εικόνα 31 HeuristicMode – Τρόποι εύρεσης διαδρομής

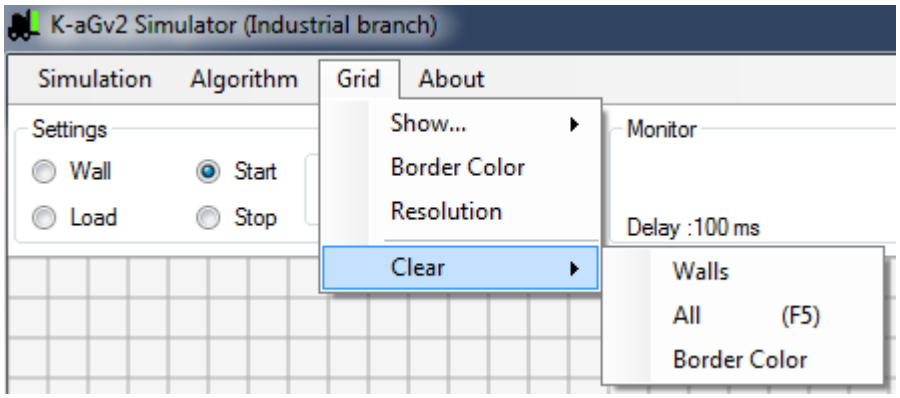
3.5.3. Grid

- Show
  - Steps
  - Lines
  - Dots
  - Borders
  - Highlight over current box
  - Index
- Border Color
- Resolution



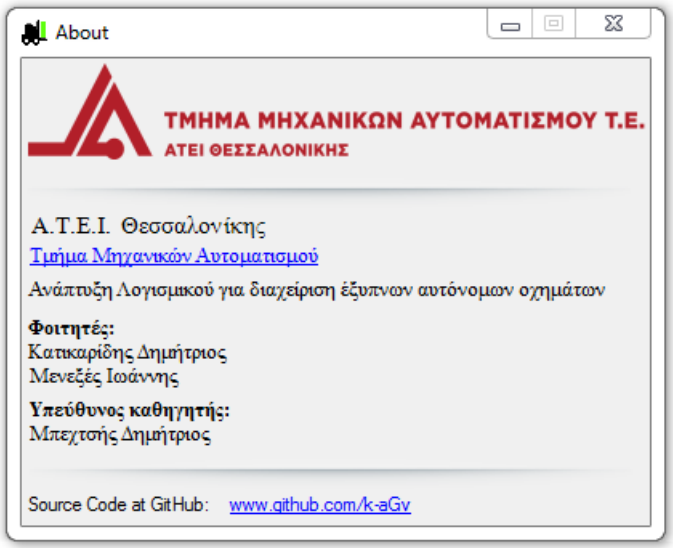
Εικόνα 32 Grid – Εμφάνιση στοιχείων και πληροφοριών στο πλέγμα

- Clear
  - Walls
  - All
  - Border Color



Εικόνα 33 Grid – Επιλογές καθαρισμού του πλέγματος

3.5.4. About



Εικόνα 34 Σχετικά με την εφαρμογή

3.6. Συναρτήσεις εύρεσης διαδρομής

Το πρώτο βήμα ήταν να βρεθεί αλγόριθμος γραμμένος στη γλώσσα C# για την εύρεση της βέλτιστης διαδρομής ανάμεσα σε 2 σημεία. Έπειτα από έρευνα, καταλήξαμε στο, βασισμένο σε A*, αλγόριθμο του Woong Gyu La.

Ο συγκεκριμένος αλγόριθμος, εμφανίζει τη βέλτιστη διαδρομή αποθηκεύοντας τα ενδιάμεσα σημεία καμπής χωρίς να συγκρατεί όλα τα υπόλοιπα ενδιάμεσα σημεία που απαρτίζουν την υπολογισμένη διαδρομή.

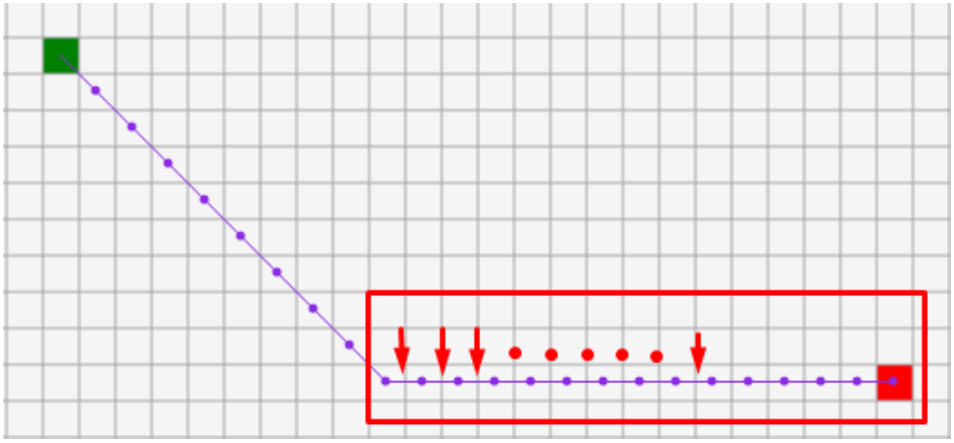
Για τη διαδικασία του υπολογισμού, ακολουθούνται τα εξής βήματα:

1. Αναγνώριση του αρχικού και τελικού σημείου (GridBox.cs) επάνω στο grid (BaseGrid.cs)
2. Εκτέλεση του A* αλγορίθμου, του οποίου η υλοποίηση συμβαίνει μέσα στο class “JumpPointFinder.cs”
3. Αποθήκευση των Nodes στα “DynamicGridWPool.cs” και “NodePool.cs”
4. Διαχωρισμός των Nodes που αποτελούν τη βέλτιστη διαδρομή, και αποθήκευση των σημείων καμπής σε List τύπου GridPos (η List που περιέχει τα Nodes, επιστρέφεται στο “JumpPointFinder.cs”)
5. Η συνάρτηση JumpPointFinder.FindPath() επιστρέφει στο κυρίως πρόγραμμα την παραπάνω List.

Το πρώτο πρόβλημα που κλήθηκε να αντιμετωπιστεί, ήταν η εύρεση των παραπάνω ενδιάμεσων σημείων, των οποίων η μελλοντική χρήση θα ήταν αναγκαία για την προσομοίωση της κίνησης των

οχημάτων.
Ο υπολογισμός αυτός, γίνεται με τον παρακάτω τρόπο:

Γνωρίζοντας τις συντεταγμένες του αρχικού σημείου και τις συντεταγμένες του επόμενου σημείου καμπής, και εφόσον τα δύο αυτά σημεία είναι συνευθειακά, ανεξαρτήτου διεύθυνσης, είναι δυνατό να χωρίσουμε την ευθεία αυτή σε επιμέρους ημιευθείες, ίσου μήκους. Για να επιτευχθεί όμως αυτό, έπρεπε πρώτα να υπολογιστεί η απόσταση μεταξύ του αρχικού σημείου και του σημείου καμπής, ο οποίος υπολογισμός πραγματοποιήθηκε μέσω της συνάρτησης “kagvFunctions.kFunctions. GetLength()” που επιστρέφει την ευκλείδεια απόσταση σε αριθμό pixels.



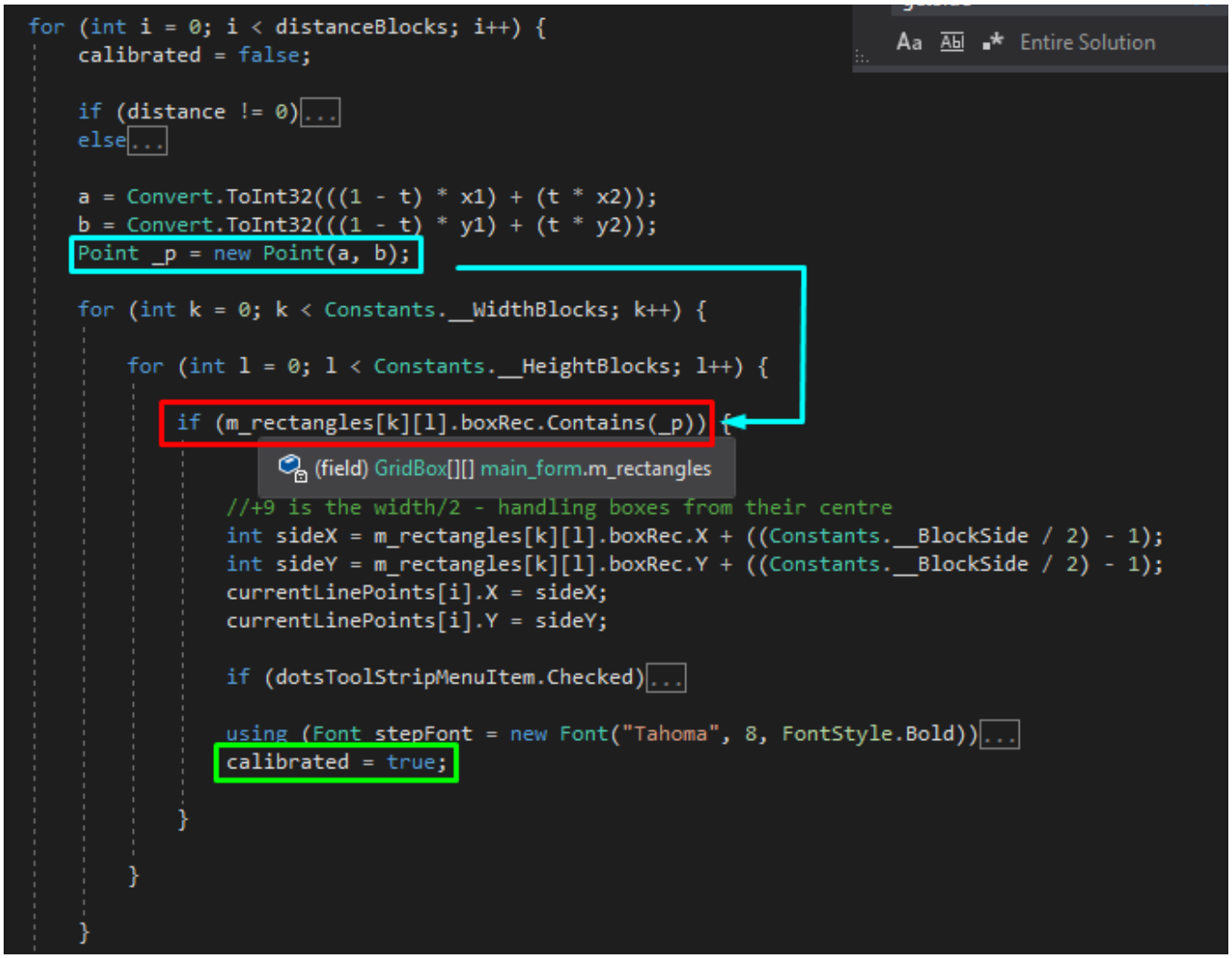
Εικόνα 35 Προσέγγιση προβλήματος εύρεσης των ενδιάμεσων ημιευθειών

Επόμενο βήμα ήταν να αναγνωριστεί η διεύθυνση της παραπάνω ευθείας. Δεδομένου ότι τα πιθανά σενάρια είναι δύο (διαγώνιος ή μη ευθεία), έπρεπε να γίνουν οι αντίστοιχες μαθηματικές πράξεις.

Στην περίπτωση της διαγωνίου ευθείας, ο υπολογισμός των ημιευθειών απαιτεί την εύρεση της υποτείνουσας του GridBox. Γνωρίζοντας πως το κάθε GridBox είναι τετράγωνο, με διαστάσεις που έχουμε ορίσει εμείς, είναι εφικτό να υπολογίσουμε την υποτείνουσα του, εφαρμόζοντας το Πυθαγόρειο Θεώρημα. Η ανάγκη του παραπάνω υπολογισμού, απαιτούσε την δημιουργία μιας ξεχωριστής συνάρτησης. Η συνάρτηση που αναπτύχθηκε ονομάζεται “kagvFunctions.kFunctions.getSide()”, δέχεται ως ορίσματα τις διαστάσεις του GridBox και επιστρέφει την υποτείνουσά του. Έχοντας πλέον στη διάθεσή μας τα παραπάνω δεδομένα, της Ευκλείδειας απόστασης μεταξύ δύο (2) σημείων και του μήκους της υποτείνουσας του GridBox, συμπληρώνουμε όλες τις απαραίτητες μεταβλητές ώστε να λυθεί η εξίσωση για την εύρεση του πλήθους των GridBox που απαρτίζουν την παραπάνω διαγώνιο. Η διαίρεση, τελικά, της Ευκλείδειας απόστασης και της υποτείνουσας, δίνει ως αποτέλεσμα τον αριθμό των κελιών που ισούται με το πλήθος των ημιευθειών από τις οποίες αποτελείται η εκάστοτε διαγώνιος.

Στη δεύτερη περίπτωση, της μη διαγωνίου ευθείας, η διαδικασία που ακολουθείται είναι αισθητά πιο απλουστευμένη από την παραπάνω. Οι μεταβλητές που χρειαζόμαστε, πλέον, είναι η Ευκλείδεια απόσταση και μόνο το μήκος της μίας πλευράς του GridBox. Διαιρώντας τα δύο αυτά δεδομένα, προκύπτει το πλήθος των κελιών που ανήκουν στην παραπάνω, είτε οριζόντια είτε κατακόρυφη, ευθεία.

Στη συνέχεια, ακολουθείται μια επαναλαμβανόμενη διαδικασία κατά την οποία υπολογίζεται η αρχή και το τέλος κάθε ημιευθείας, με τρόπο τέτοιο ώστε κάθε επόμενη ημιευθεία να έχει ως αρχικό σημείο το ζεύγος συντεταγμένων που αποτελούν το τελικό σημείο της προηγούμενης. Το αποτέλεσμα όμως των μαθηματικών υπολογισμών, μέσω των οποίων προέκυψε η τιμή της Ευκλείδειας απόστασης, καθιστά την τελευταία, εκ των πραγμάτων, δεκαδικής μορφής. Αυτό μεταφράζεται σε μη δυνατή μετατροπή της, από δεκαδική μορφή σε πλήθος pixels. Η μετατροπή αυτή κρίθηκε απαραίτητη, διότι ήταν επιθυμητό η ημιευθείες να έχουν το αρχικό τους σημείο στο κέντρο του κάθε GridBox, γεγονός που προκαλούσε πρόβλημα μη ταύτισης των δύο (2) αυτών παραμέτρων. Αυτό είχε ως αποτέλεσμα η ευθεία που απαρτίζεται από τις επιμέρους υπολογιζόμενες ημιευθείες να μη βρίσκεται σε πλήρη συμφωνία με την ευθεία που εμείς επιχειρούμε να διασπάσουμε. Τη λύση σε αυτό το πρόβλημα τη βρήκαμε αναπτύσσοντας μια συγκεκριμένη ρουτίνα, κατά την οποία ελέγχουμε εάν το υπολογιζόμενο ζεύγος συντεταγμένων περιέχεται στην ιδιότητα “boxRec” του πίνακα “m_rectangles[ ] [ ]”, ο οποίος πίνακας είναι τύπου GridBox.



Εικόνα 36 Τμήμα κώδικα για την διόρθωση απόκλισης ανάμεσα στις υπολογιζόμενες και την πραγματικέςημιευθείες

Σε κάθε αληθή παραπάνω περίπτωση, είναι εφικτό να τοποθετήσουμε το αντίστοιχο ζεύγος συντεταγμένων στο κέντρο του εκάστοτε κελιού, εξαλείφοντας την παραμικρή πιθανότητα μαθηματικής ανακρίβειας.

4. Λογισμικό και εφαρμογή

4.1. Υποστηριζόμενοι τύποι κελιών

Έχουμε ήδη αναφέρει πως κάθε στοιχείο του πίνακα m_rectangles[ ] [ ] αντιπροσωπεύει και ένα αντίστοιχο κελί στο πλέγμα. Όλα τα κελιά είναι αντικείμενα της κλάσης “GridBox” και ο τύπος τους προσδιορίζεται από την ιδιότητα “BoxType”, η οποία μπορεί να πάρει πέντε διαφορετικές τιμές.

4.1.1. Start/Stop

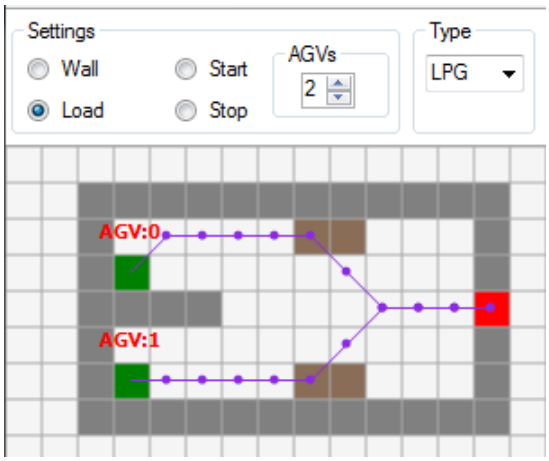
Το λογισμικό υποστηρίζει αυτοοδηγούμενα οχήματα τα οποία ο χρήστης μπορεί να εισάγει στο πλέγμα, ενεργοποιώντας την επιλογή “Start” από το ειδικό μενού ρυθμίσεων στο επάνω-αριστερό μέρος της κύριας φόρμας και ορίζοντας τον αριθμό αυτών από το αριθμητικό χειριστήριο με τίτλο «AGVs». Έχοντας εκτελέσει τα παραπάνω βήματα, μπορεί πλέον ο χρήστης να αρχικοποιήσει τις θέσεις των αυτοοδηγούμενων οχημάτων χρησιμοποιώντας το ποντίκι. Ο τελικός προορισμός είναι μοναδικός για όλα τα οχήματα και ορίζεται από το χρήστη, επιλέγοντας “Stop” από το ειδικό μενού και στη συνέχεια τοποθετώντας το επάνω στο πλέγμα.

4.1.2. Loads

Ο χρήστης μπορεί να αναθέσει αποστολές στα οχήματα τοποθετώντας φορτία προς μετακίνηση, επάνω στο πλέγμα. Η διαδικασία δημιουργίας των φορτίων είναι παρόμοια με την εισαγωγή των αυτοοδηγούμενων οχημάτων με διαφορά ότι δεν περιορίζει το χρήστη στο μέγιστο αριθμό φορτίων που μπορεί να εισάγει στην εφαρμογή. Ενεργοποιώντας την επιλογή “load” από το ειδικό μενού, ο χρήστης τοποθετεί τα φορτία σε όποια σημεία ορίσει ο ίδιος ενώ ακόμη, μπορεί να τα αφαιρέσει κάνοντας “click” στο φορτίο που επιθυμεί να αφαιρέσει.

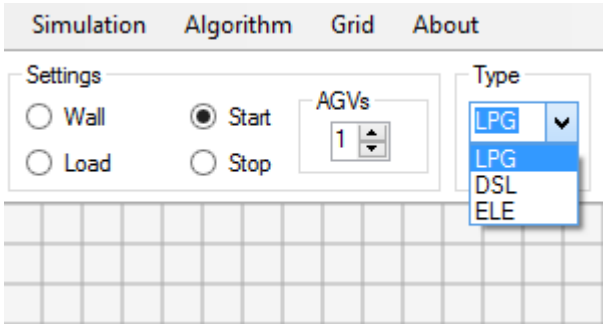
4.1.3. Walls/Normal

Ο χώρος μέσα στον οποίο θα εκτελεστεί η διεργασία που ορίζει ο χρήστης, αρχικά αποτελείται μόνο από άδεια κελιά (“Normal”) και τον οριοθετεί επιλέγοντας “Walls” από το ειδικό μενού. Ο σχεδιασμός της κάτοψης του χώρου εργασίας πραγματοποιείται με όμοιο τρόπο, όπως και παραπάνω, με το χρήστη να επιλέγει, με κλικ του ποντικιού, ποιά κελιά του πλέγματος θα έχουν την ιδιότητα του “τοίχου”. Τέλος, η εφαρμογή προσφέρει στο χρήστη τη δυνατότητα να πραγματοποιήσει την παραπάνω διαδικασία, κρατώντας πατημένο το κλικ του ποντικιού και παράλληλα μετακινώντας το σε όλα τα επιθυμητά σημεία που θέλει να ορίσει ως “τοίχο” / εμπόδιο. Η εναλλαγή των κελιών από “Wall” σε “Normal” είναι αμφίδρομη και πραγματοποιείται κάνοντας κλικ στο αντίστοιχο αντικείμενο.



Εικόνα 37 Παράδειγμα σχεδιασμού ενός χώρου εργασίας

Ο χρήστης έχει τη δυνατότητα να επιλέξει τον τύπο καυσίμου που χρησιμοποιούν τα αυτοοδηγούμενα οχήματα μέσω της επιλογής “Type” που βρίσκεται στο ειδικό μενού.



Εικόνα 38 Τρόπος επιλογής τύπου οχήματος

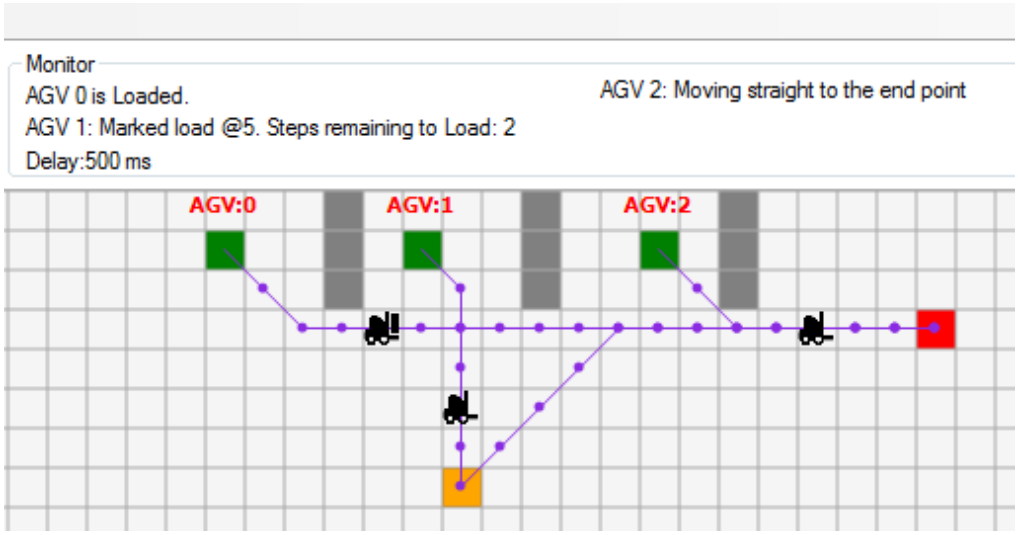
4.2. Λειτουργίες παρακολούθησης και μηχανισμοί ευελιξίας

Ο τρόπος ανάπτυξης της εφαρμογής την καθιστά σημαντικά ευέλικτη καθώς παρέχει στο χρήστη ένα πακέτο δυνατοτήτων και λειτουργιών που στοχεύουν τόσο στον εύκολο αλλά και αποτελεσματικό χειρισμό της. Ο χειριστής της είναι ικανός να δέχεται ανάδραση από την εφαρμογή με μηχανισμούς τέτοιους που έχουν ως αποτέλεσμα μια συνολική εμπειρία πιο ευχάριστη και φιλική προς τον ίδιο.

4.2.1. Παράθυρο εποπτείας της διεργασίας

Ο χρήστης, κατά τη διάρκεια της προσομοίωσης, έχει πρόσβαση σε πληροφορίες σχετικές με τη διεργασία, που τον ενημερώνουν για την τρέχουσα κατάσταση των αυτοοδηγούμενων οχημάτων καθώς επίσης και την απόστασή τους από το φορτίο προς παραλαβή. Ακόμη, μπορεί να πληροφορηθεί

και γιατο χρόνο περιόδου των χρονιστών που χειρίζονται τα οχήματα, έτσι ώστε να παρέμβει στο ρυθμό με τον οποίο πραγματοποιείται η προσομοίωση.



Εικόνα 39 Μενού παρακολούθησης της εν εξελίξει διεργασίας

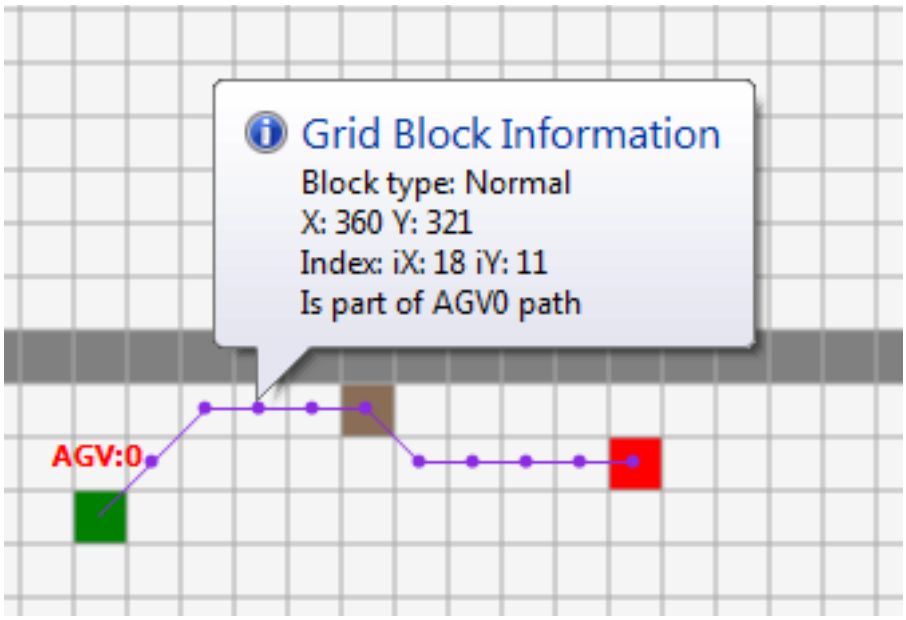
```
877
878         private void displayStepsToLoad(int counter, int agv_index) {
879             int stepstoload;
880             string agvinfo;
881
882             if (getStepsToLoad(agv_index) == -1)
883                 agvinfo = "AGV " + (agv_index) + ": Moving straight to the end point";
884             else {
885                 stepstoload = (getStepsToLoad(agv_index) - counter);
886                 agvinfo = "AGV " + (agv_index )
887                     + ": Marked load @" + getStepsToLoad(agv_index)
888                     + ". Steps remaining to Load: " + stepstoload;
889                 if (stepstoload < 0)
890                     agvinfo = "AGV " + (agv_index ) + " is Loaded.";
891             }
892
893             gb_monitor.Controls.Find(
894                 "agv" + (agv_index + 1) + "steps_LB",
895                  true)[0].Text = agvinfo;
896
897         }
898
```

Εικόνα 40 Τμήμα κώδικα για την υλοποίηση του παραθύρου παρακολούθησης

4.2.2. Tooltip ποντικιού

Έπειτα, ένα ακόμη πακέτο πληροφοριών περιέχεται στο αναδυόμενο μενού, που εμφανίζεται κατά το πάτημα του δεξιού πλήκτρου του ποντικιού επάνω στο πλέγμα. Το εν λόγω μενού εμφανίζει δεδομένα για το κελί πάνω από το οποίο ο χρήστης εκτέλεσε το πάτημα του δεξιού «κλικ», όπως ο τύπος του και

η τοποθεσία του, υπολογιζόμενη σε pixels ή σύμφωνα με τη διάταξή του στο πλέγμα, καθώς επίσης πληροφορεί αν το συγκεκριμένο κελί αποτελεί μέρος της διαδρομής κάποιου οχήματος.



Εικόνα 41 Παράθυρο πληροφοριών σχετικές με τα κελιά του πλέγματος (Mouse Tooltip)

4.2.3. Μηχανισμοί εισαγωγής / εξαγωγής χώρου εργασίας

Χρησιμοποιώντας τις επιλογές εισαγωγής και εξαγωγής χάρτη, ο χρήστης βρίσκεται σε θέση είτε να αποθηκεύσει το χάρτη που έχει σχεδιάσει, είτε να εισάγει έναν τον οποίο έχει προσχεδιάσει στο παρελθόν. Το παραγόμενο αρχείο έχει την επέκταση (*.kmap) και η διαμόρφωσή του είναι αναγνωρίσιμη μόνο από το συγκεκριμένο λογισμικό. Στόχος των δύο αυτών λειτουργιών είναι η εξοικονόμηση χρόνου και κόπου στην περίπτωση που ο χρήστης επιθυμεί να επαναλάβει μια προσομοίωση, για ένα χώρο εργασίας που παλαιότερα είχε σχεδιάσει.

4.2.4. Χειρισμός των χρονιστών της εφαρμογής

Παραπάνω αναφέραμε πως ο χρήστης είναι ενήμερος για το ρυθμό εκτέλεσης της προσομοίωσης και ικανός να παρέμβει σε αυτόν μεταβάλλοντάς τον. Είναι εφικτή, λοιπόν, η αυξομείωση της ταχύτητας αναπαράστασης της προσομοίωσης είτε χρησιμοποιώντας τις αντίστοιχες επιλογές που βρίσκονται στην πρώτη καρτέλα του κύριου μενού, είτε μέσω συντομεύσεων (Up/Down). Επίσης, ανάλογες συντομεύσεις είναι διαθέσιμες για την εκκίνηση της προσομοίωσης (Space) αλλά και την εκκαθάριση του πλέγματος εργασίας (F5).

4.2.5. Παραμετροποίηση ευρετικών αλγορίθμων

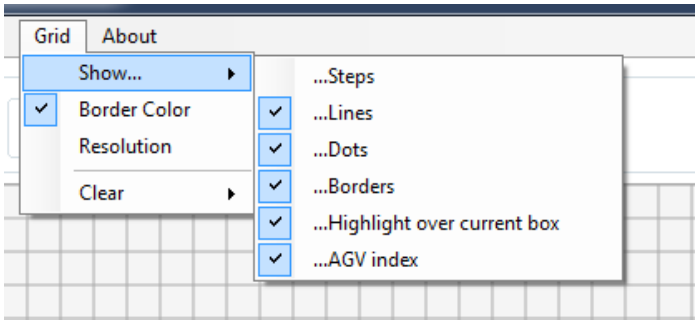
Τέλος, αξίζει να σημειωθεί η δυνατότητα παραμετροποίησης της ευρετικής λειτουργίας του αλγορίθμου, που υπολογίζει τη βέλτιστη διαδρομή. Οι παράμετροι ρυθμίσεων ορίζουν το μοντέλο (Ευκλείδια απόσταση, Chebyshev, Manhattan) σύμφωνα με το οποίο ο αλγόριθμος, θα πραγματοποιεί τις απαιτούμενες μαθηματικές πράξεις. Ακόμη, ο αλγόριθμος μπορεί να ρυθμιστεί έτσι ώστε να ικανοποιεί μία ευέλικτη διαδρομή πάνω στο πλέγμα (Cross adjacent point, cross corner). Τέλος, ο χρήστης δύναται να επιλέξει τον τρόπο με τον οποίο θα υπολογίζεται, προγραμματιστικά, η βέλτιστη διαδρομή (recursive function / loop function) καταναλώνοντας την αντίστοιχη υπολογιστική ισχύ.

4.3. Παραμετροποίηση γραφικών στοιχείων του πλέγματος

Πέραν των υπολογιστικών παραμετροποιήσεων, η εφαρμογή προσφέρει επιλογές ρύθμισης των εικονικών στοιχείων της. Οι τελευταίες, έχουν τοποθετηθεί στο μενού «Grid» και μέσω αυτών ο χρήστης έχει τη δυνατότητα να διαμορφώσει το γραφικό περιβάλλον όπως επιθυμεί. Πιο συγκεκριμένα, οι παραμετροποιήσεις αφορούν την εμφάνιση της:

- αρίθμησης βημάτων κάθε διαδρομής
- χάραξης της πορείας του κάθε οχήματος
- σημείωσης των βημάτων κάθε διαδρομής
- απόχρωσης του πλέγματος, αν αυτή έχει αλλάξει
- επισκίασης του πλαισίου που βρίσκεται το ποντίκι
- αρίθμησης των οχημάτων

Στην ίδια κατηγορία ρυθμίσεων ανήκουν οι επιλογές αλλαγής χρώματος των ορίων του πλέγματος, της ανάλυσής του και επαναφοράς της κατάστασης αυτού, στην αρχική του μορφή.



Εικόνα 42 BorderColor – Χρωματισμός του ορίου πλέγματος

4.4. Πρωτόκολλα επικοινωνίας φορτίων - οχημάτων

Η πολυπλοκότητα που προέκυψε ώστε η διεργασία να διεξάγεται ομαλά, απαιτούσε την ανάπτυξη ενός πρωτοκόλλου επικοινωνίας των οντοτήτων που βρίσκονται στο χώρο. Στόχος είναι η άμεση λήψη αποφάσεων, μέσω συγκεκριμένων κριτηρίων, που αφορούν τον τρόπο διεξαγωγής των διαδικασιών (planning).

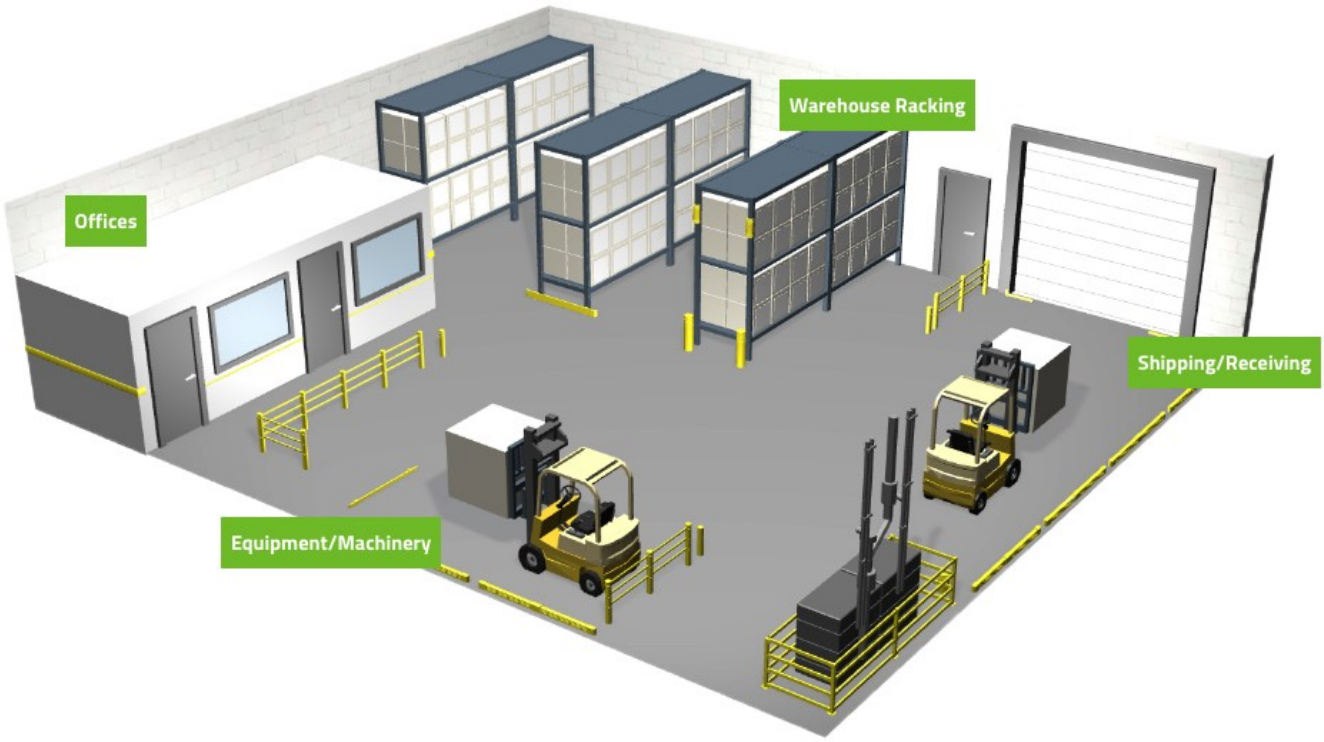
Το εκάστοτε φορτίο εκδίδει ένα αίτημα μεταφοράς, προς τα όλα τα οχήματα του χώρου. Τα οχήματα, με τη σειρά τους, κοινοποιούν στο παραπάνω φορτίο, την κατάσταση στην οποία βρίσκονται και στη συνέχεια το φορτίο επιλέγει, βάσει συγκεκριμένων κανόνων και έπειτα από ελέγχους που έχουν προηγηθεί, το όχημα που θα αναλάβει τη μεταφορά του. Κατόπιν της παραπάνω επιλογής, το όχημα ενημερώνεται για τη νέα αποστολή του, και γνωστοποιεί, στις άλλες οντότητες, ότι δεν είναι πλέον διαθέσιμο να εκπληρώσει κάποιο άλλο αίτημα. Τελικά το όχημα μετακινείται, ακολουθώντας τη βέλτιστη διαδρομή, προς το φορτίο και εν συνεχεία το μεταφέρει στον τελικό προορισμό. Κατά την άφιξή του, παραδίδει το φορτίο του στο σημείο εκφόρτωσης και δηλώνει ξανά, στις υπόλοιπες οντότητες, την κατάσταση διαθεσιμότητάς του. Μόλις γίνει αυτό, επαναλαμβάνεται η παραπάνω διαδικασία, όπως αυτή περιγράφηκε, και ο κύκλος διεργασιών συνεχίζεται έως ότου εξαντληθούν τα διαθέσιμα φορτία.

```
checkForTrappedLoads(loadPos);

for (int widthTrav = 0; widthTrav < Constants.__WidthBlocks; widthTrav++)
    for (int heightTrav = 0; heightTrav < Constants.__HeightBlocks; heightTrav++)
        if (m_rectangles[widthTrav][heightTrav].boxType == BoxType.Load && isLoad[widthTrav, heightTrav] == 1) {
            isLoad[widthTrav, heightTrav] = 3;
            AGVs[whichAGV].MarkedLoad = new Point(widthTrav, heightTrav);

            endPos.x = widthTrav;
            endPos.y = heightTrav;
            loads--;

            //Mark all loads as unwalkable,except the targetted ones
            for (int k = 0; k < Constants.__WidthBlocks; k++)
                for (int l = 0; l < Constants.__HeightBlocks; l++)
                    if (m_rectangles[k][l].boxType == BoxType.Load && isLoad[k, l] != 3)
                        searchGrid.SetWalkableAt(new GridPos(k, l), false);

            widthTrav = Constants.__WidthBlocks;
            heightTrav = Constants.__HeightBlocks;
        }
```

Εικόνα 43 Τμήμα κώδικα ανάθεσης αποστολής για παραλαβή φορτίου

5. Χρήση της εφαρμογής

Στο προηγούμενο κεφάλαιο αναλύσαμε τον τρόπο λειτουργίας της εφαρμογής και όλο το παρασκήνιο αυτής. Στο κεφάλαιο αυτό θα υλοποιήσουμε μια διεργασία από μία πραγματική αποθήκη, έστω της παρακάτω εικόνας, σχεδιάζοντας την κάτοψή της στην εφαρμογή.

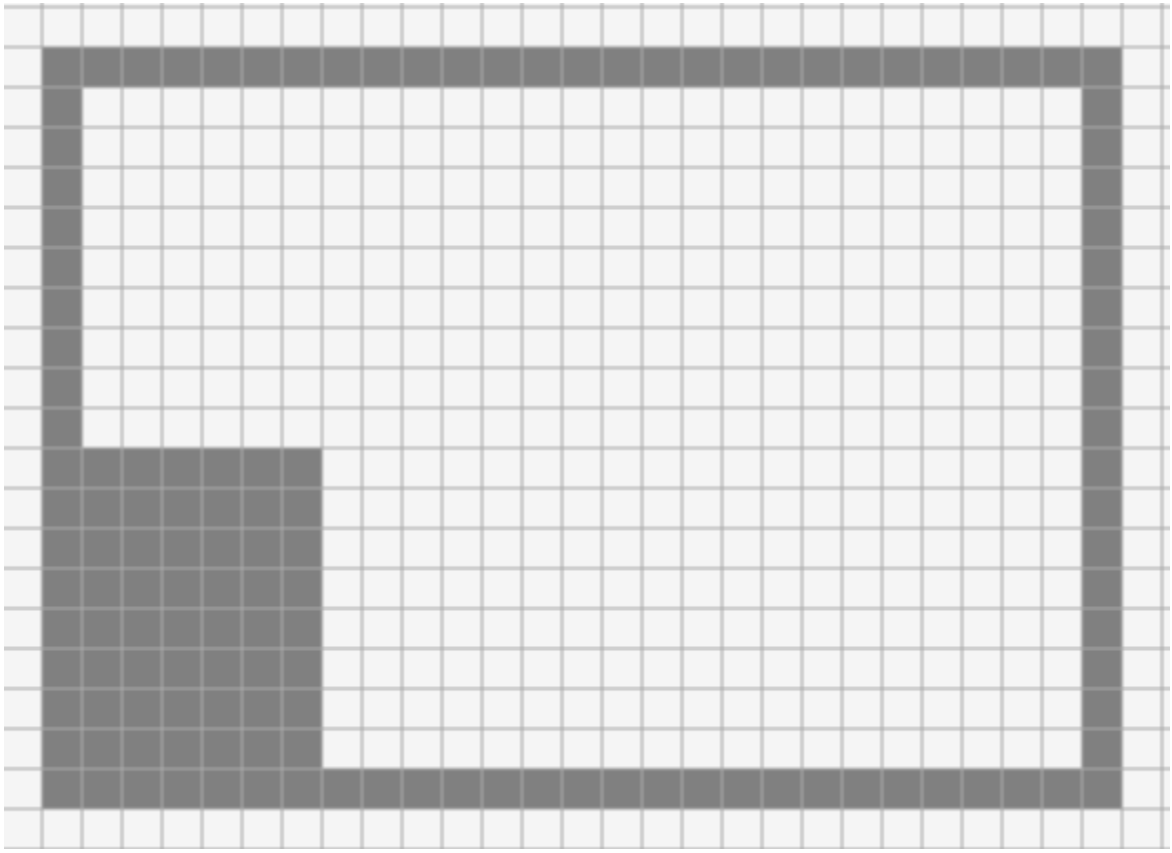


Εικόνα 44 Πρότυπο για το σχεδιασμό κάτοψης στην εφαρμογή

5.1. Προετοιμασία του περιβάλλοντος

5.1.1. Οριοθέτηση του χώρου εργασίας

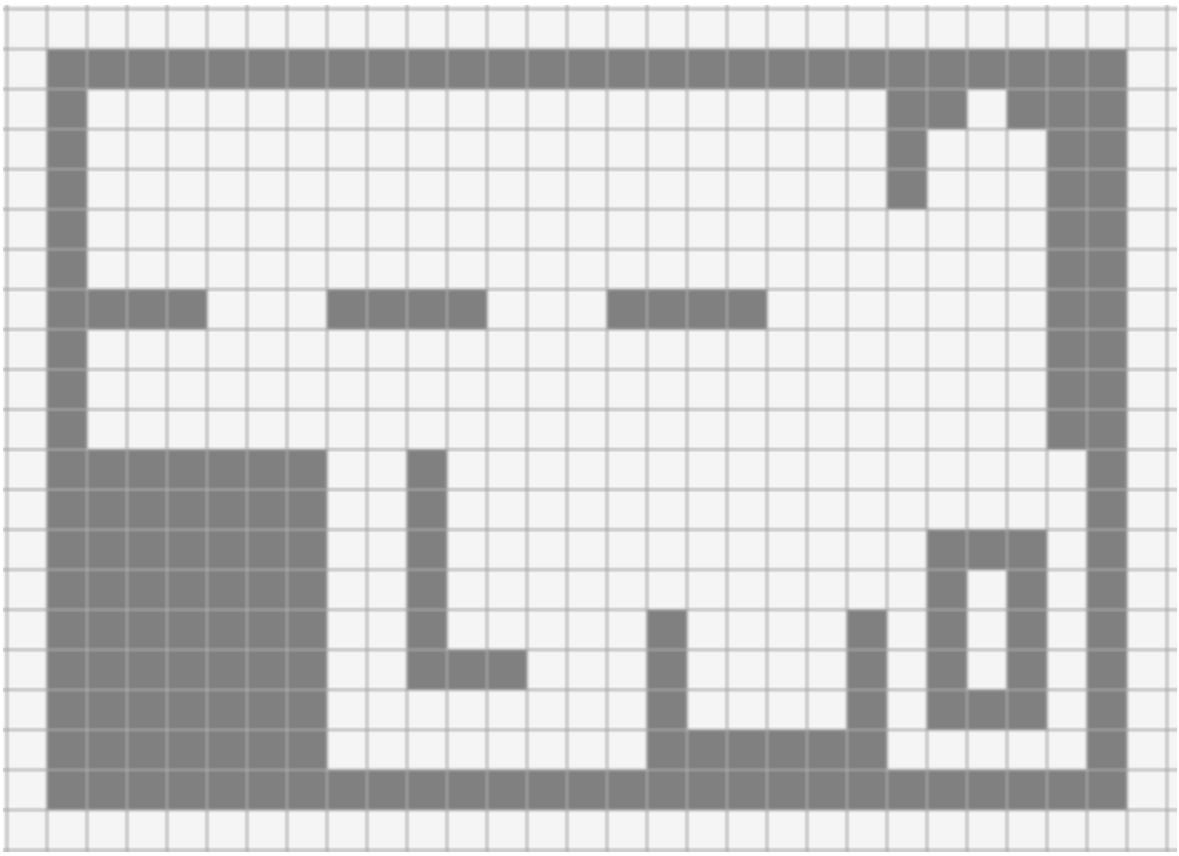
Ανοίγουμε λοιπόν την εφαρμογή και επιλέγουμε το radio button με όνομα “Walls”. Στη συνέχεια σχεδιάζουμε την περίμετρο της αποθήκης και τα γραφεία (Offices), κάνοντας κλικ στα κελιά που θέλουμε να ορίσουμε ως τοίχο.



Εικόνα 45 Οριοθέτηση του χώρου εργασίας

5.1.2. Προσθήκη εσωτερικών εμποδίων

Στη συνέχεια, τοποθετούμε τα κάγκελα και τα προστατευτικά κράσπεδα όπως την εικόνα.



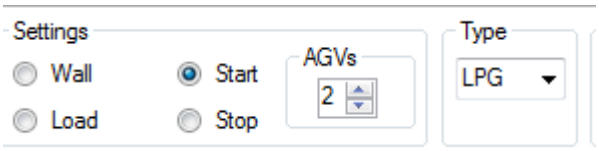
Εικόνα 46 Προσθήκη εσωτερικών εμποδίων στο χώρο εργασίας

Πλέον απομένουν τα 2 οχήματα, τα φορτία και το τελικό σημείο εκφόρτωσης. Ξεκινώντας από τα φορτία, επιλέγουμε το “Start” ώστε να τοποθετήσουμε τις αρχικές θέσεις των οχημάτων.

Επιπλέον επιλέγουμε από το αριθμητικό μενού, τον αριθμό ‘2’ ώστε το λογισμικό να γνωρίζει πόσα οχήματα θα τοποθετηθούν, και μετά επιλέγουμε από το δίπλα μενού, τον τύπο οχήματος που θα τοποθετήσουμε.

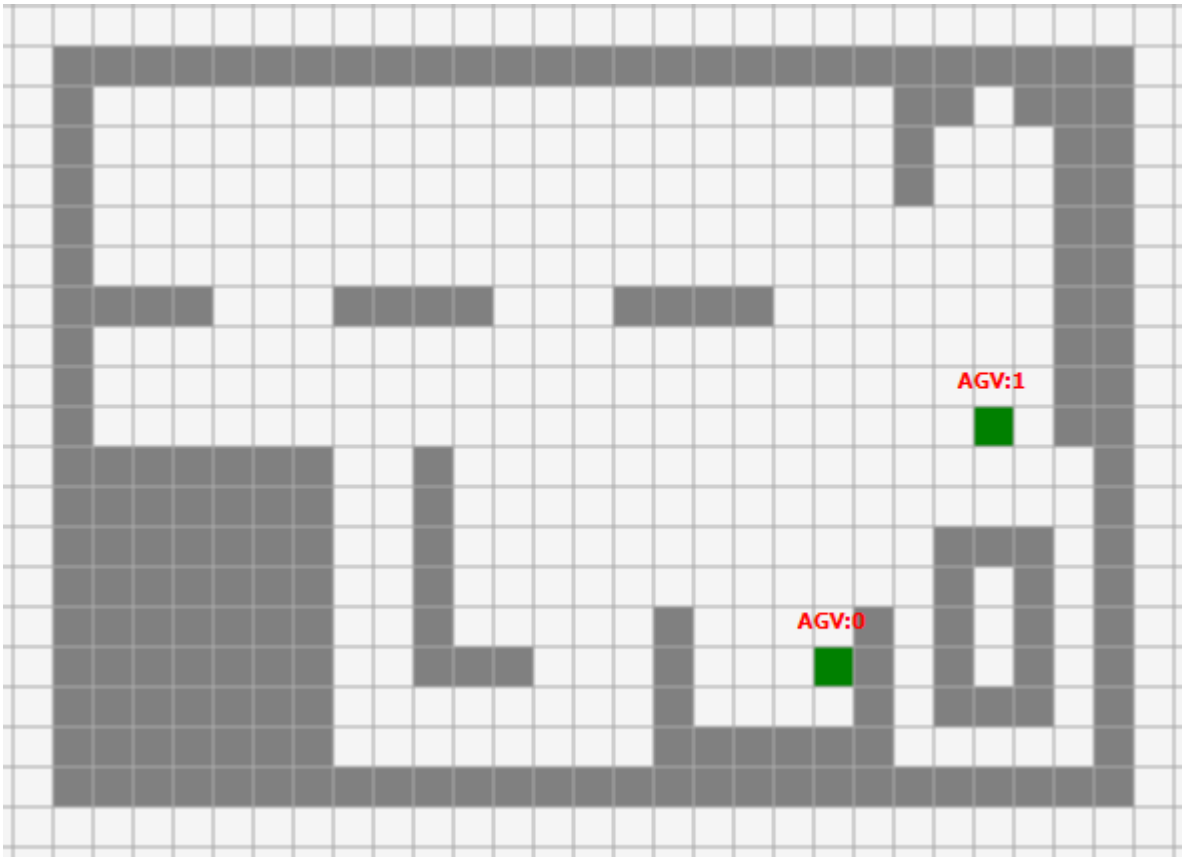
5.1.3. Εισαγωγή οχημάτων στο χώρο εργασίας

Έστω, για το παράδειγμά μας, ότι το όχημα θα είναι LPG.



Εικόνα 47 Επιλογή τύπου οχήματος

Κάνουμε κλικ στα επιθυμητά κελιά όπου θα βρίσκονται οι αρχικές θέσεις των οχημάτων μας.

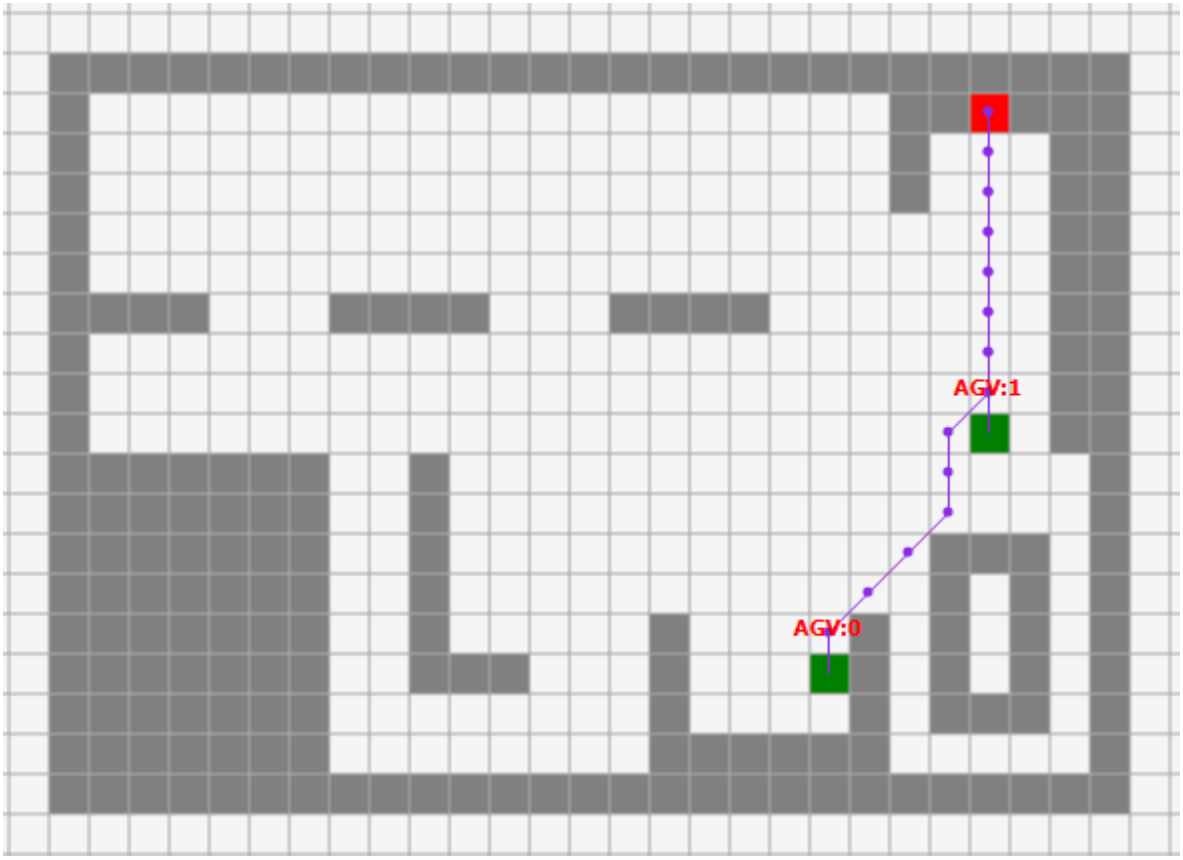


Εικόνα 48 Εισαγωγή των αρχικών θέσεων των οχημάτων

1.1.1.

5.1.4. Εισαγωγή τελικού σημείου στο χώρο εργασίας

Μετά επιλέγουμε το radio button με όνομα “Stop”, το οποίο αντιπροσωπεύει το μοναδικό τελικό σημείο εκφόρτωσης όλων των φορτίων, και κάνουμε κλικ στο σημείο που θέλουμε να τοποθετηθεί.

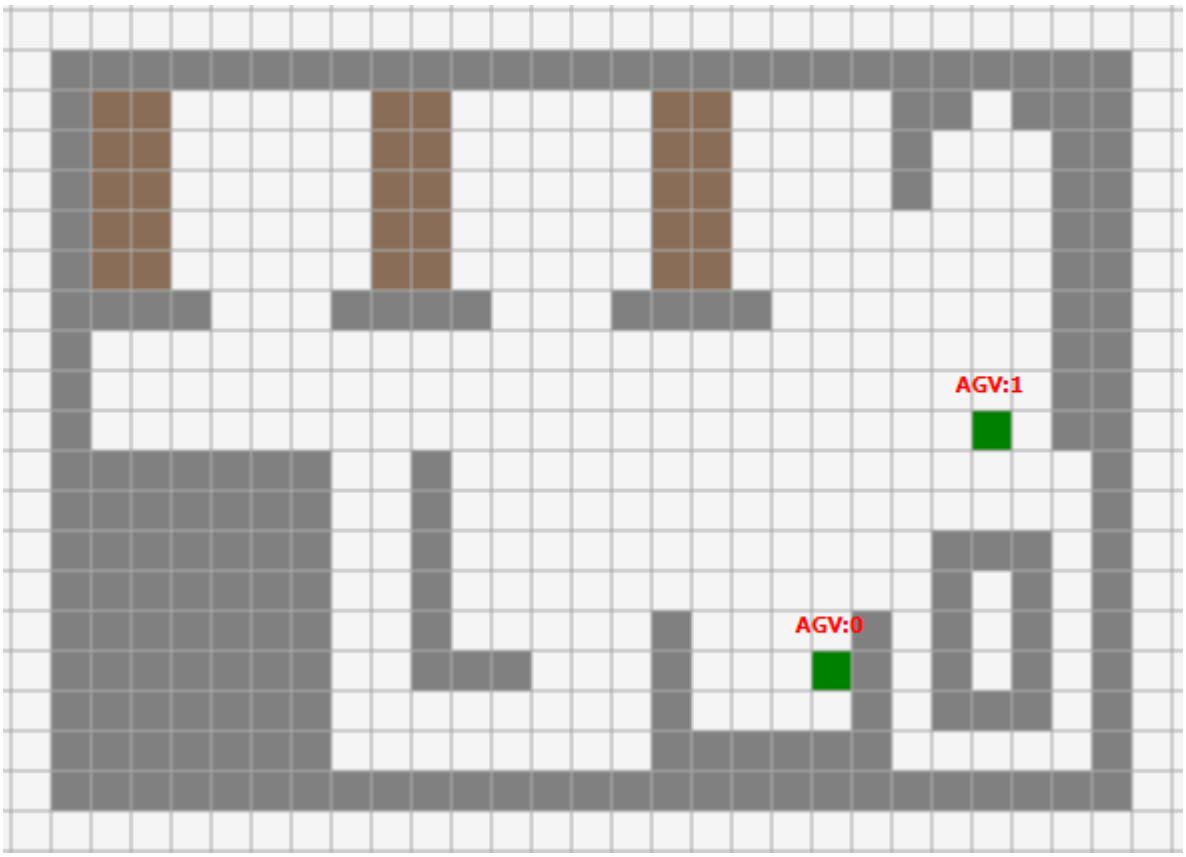


Εικόνα 49 Εισαγωγή τελικού σημείου εκφόρτωσης

Παρατηρούμε ότι το λογισμικό έχει αυτομάτως βρει τη βέλτιστη διαδρομή των οχημάτων προς το τελικό σημείο χωρίς να έχουμε προσθέσει κάποια φορτία εμείς.

5.1.5. Εισαγωγή φορτίων στο χώρο εργασίας

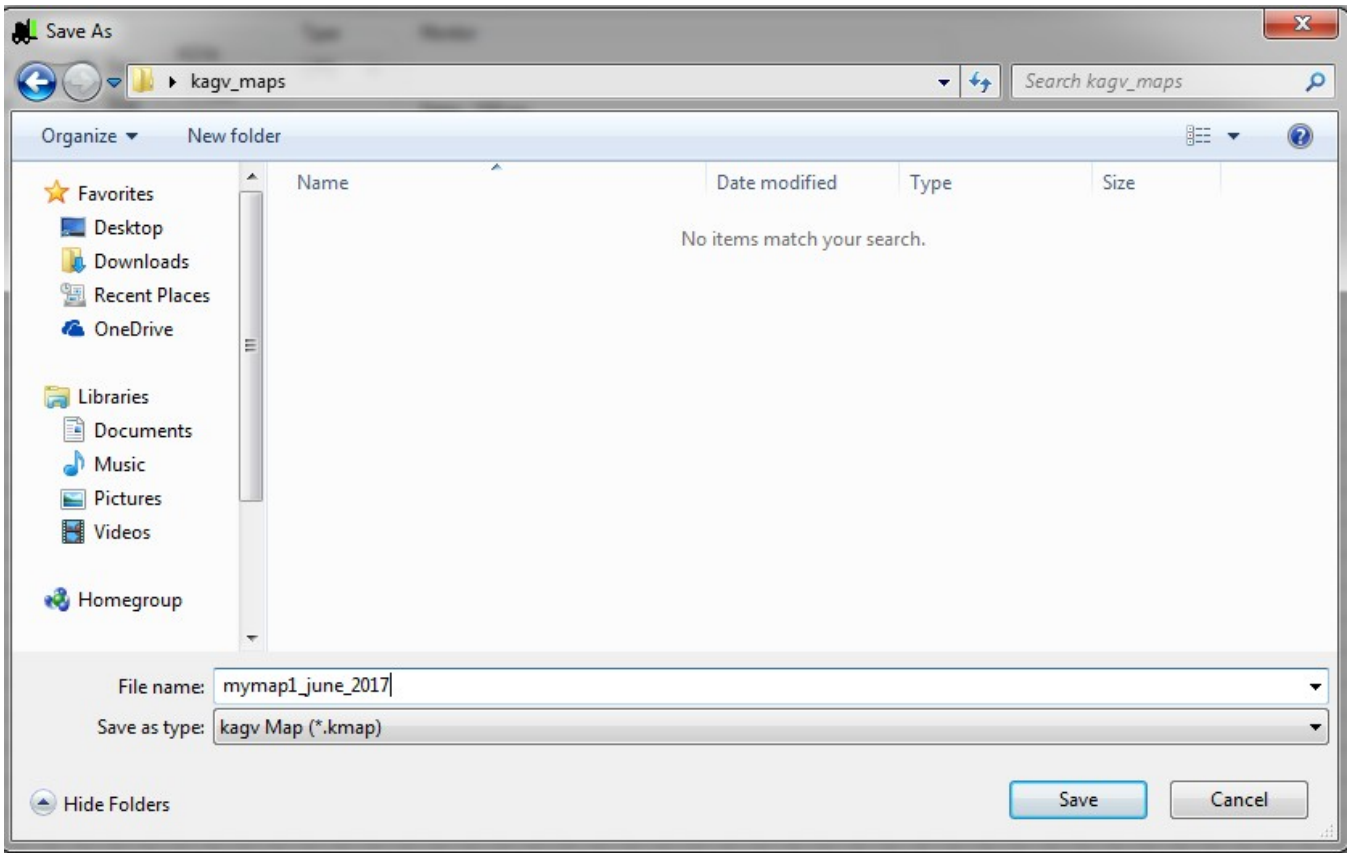
Στη συνέχεια επιλέγουμε το radio button με όνομα “Load” ώστε να τοποθετήσουμε τα φορτία στα σημεία που επιθυμούμε. Κάνουμε λοιπόν κλικ στα κελιά που αντιστοιχούν στο πλέγμα μας, βάσει της αρχικής εικόνας της αποθήκης.



Εικόνα 50 Τοποθέτηση φορτίων προς παραλαβή

5.1.6. Αποθήκευση και εξαγωγή του χώρου εργασίας

Στο σημείο αυτό, ενδεχομένως να θέλουμε να αποθηκεύσουμε το σχεδιασμένο χώρο μας. Επιλεγούμε από το μενού Simulation->Export map για να εμφανιστεί το κατάλληλο παράθυρο διαλόγου.



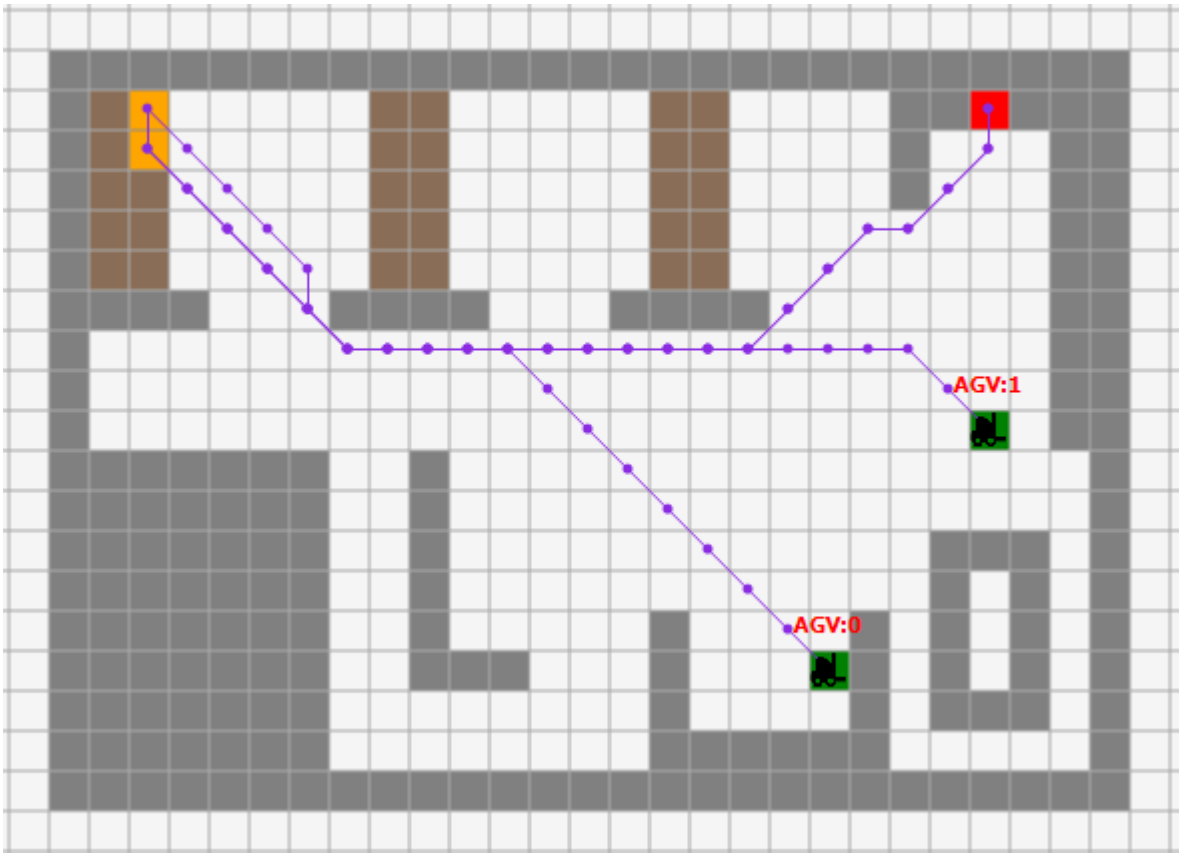
Εικόνα 51 Εξαγωγή και αποθήκευση σχεδιασμένου χώρου εργασίας

Πληκτρολογούμε το όνομα το αρχείου που θέλουμε να έχει ο χάρτης μας, και πατάμε το κουμπί “Save”. Με τον τρόπο αυτό, ουσιαστικά, εξάγουμε το σχεδιασμένο χώρο εργασίας μας σε ένα αρχείο, το οποίο όπως έχουμε ήδη αναφέρει είναι προσπελάσιμο μόνο από τη συγκεκριμένη εφαρμογή, ούτως ώστε να μην είναι απαραίτητος ο επανασχεδιασμός του σε περίπτωση που επιθυμούμε να χρησιμοποιήσουμε τον ίδιο χώρο.

5.2. Ολοκλήρωση προετοιμασίας περιβάλλοντος

5.2.1. Εκκίνηση προσομοίωσης

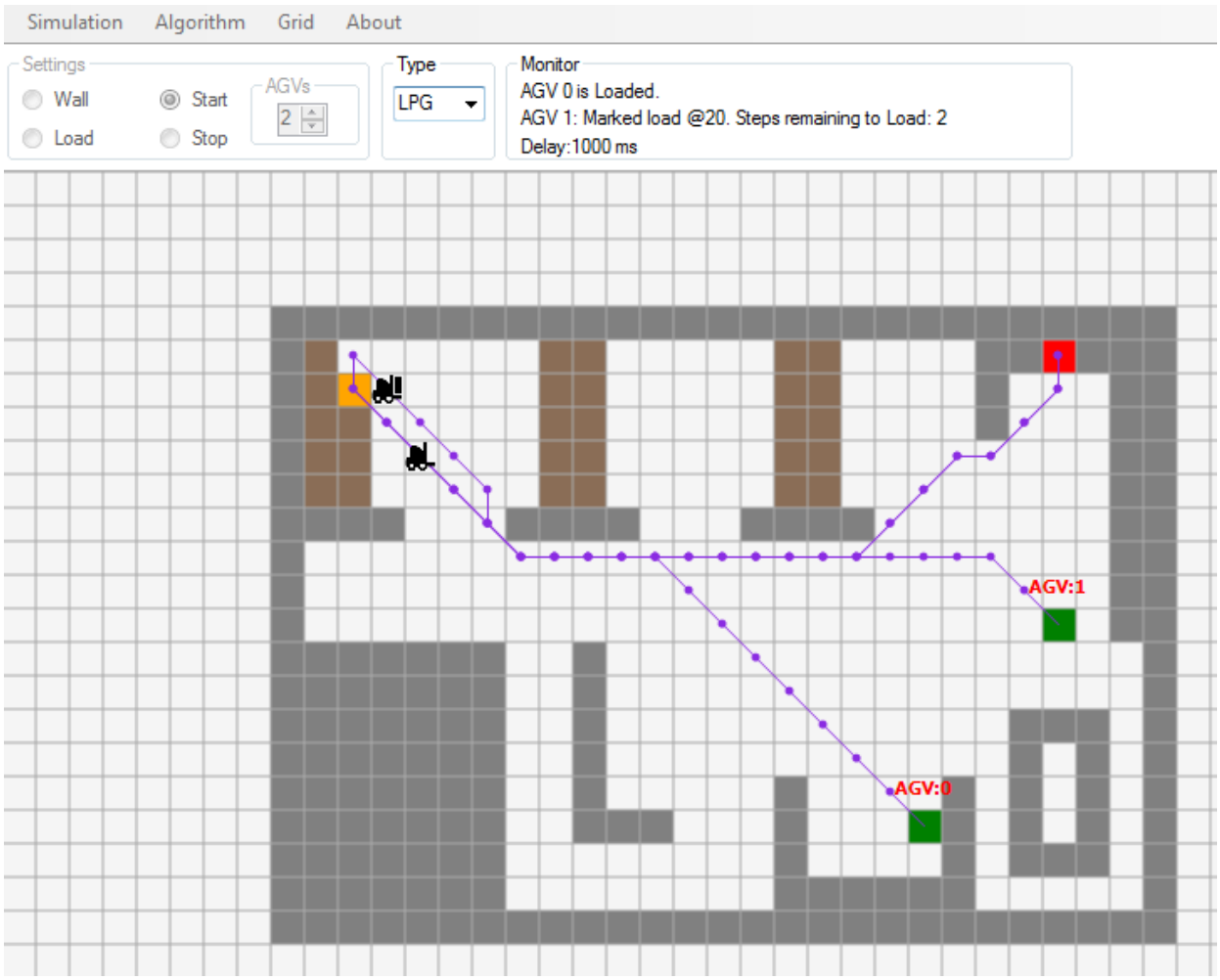
Πλέον παρατηρούμε πως το λογισμικό έχει τροποποιήσει τη βέλτιστη διαδρομή, περνώντας πρώτα από το φορτίο. Στη συνέχεια οδηγεί το όχημα στην έξοδο πάλι βρίσκοντας τη βέλτιστη διαδρομή. Αφού, λοιπόν, σχεδιάσαμε την κάτοψη της αποθήκης μπορούμε να ξεκινήσουμε την προσομοίωση



Εικόνα 52 Εκκίνηση της προσομοίωσης

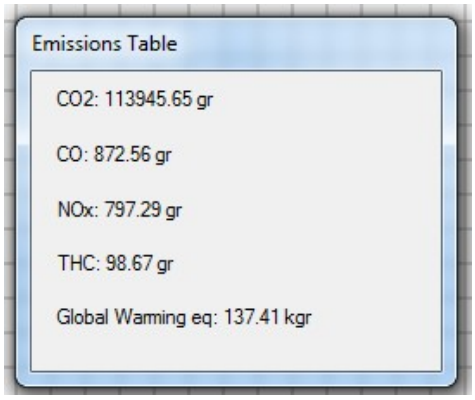
5.2.2. Εποπτεία της εν εξελίξει προσομοίωσης

Αφού αποθηκεύσαμε το layout μας, επιλέγουμε από το μενού Simulation->Start και παρατηρούμε τη σταδιακή κίνηση των οχημάτων ενώ παράλληλα μας εμφανίζεται η κατάσταση του κάθε οχήματος σε πραγματικό χρόνο.



Εικόνα 53 Κατάσταση οχημάτων: το πρώτο όχημα οδεύει προς το σημείο εκφόρτωσης για να αφήσει το φορτίο του, ενώ το δεύτερο κατευθύνεται προς φορτίο για παραλαβή

5.2.3. Παρακολούθηση των εκπομπών ρύπων



Εικόνα 54 Φόρμα εμφάνισης εκπομπών

Πέραν του πάνελ κατάστασης των οχημάτων εμφανίζεται ακόμη ένα παράθυρο διαλόγου στο οποίο εγγράφονται οι συνολικοί ρύποι που εκπέμπονται κατά την διεργασία. Έχει γίνει ήδη γνωστό πως οι εκπομπές που εμφανίζονται στην παραπάνω φόρμα, εξαρτώνται άμεσα από τον τύπο των οχημάτων που επιλέχθηκαν για την εκτέλεση της προσομοίωσης, καθώς επίσης και από την κατάσταση του εκάστοτε οχήματος. Αυτό σημαίνει πως οι ρύποι που εκπέμπονται από ένα όχημα που ήδη έχει παραλάβει ένα φορτίο, θα είναι περισσότεροι από ένα άλλο το οποίο τη δεδομένη στιγμή δε φέρει κάποιο φορτίο.

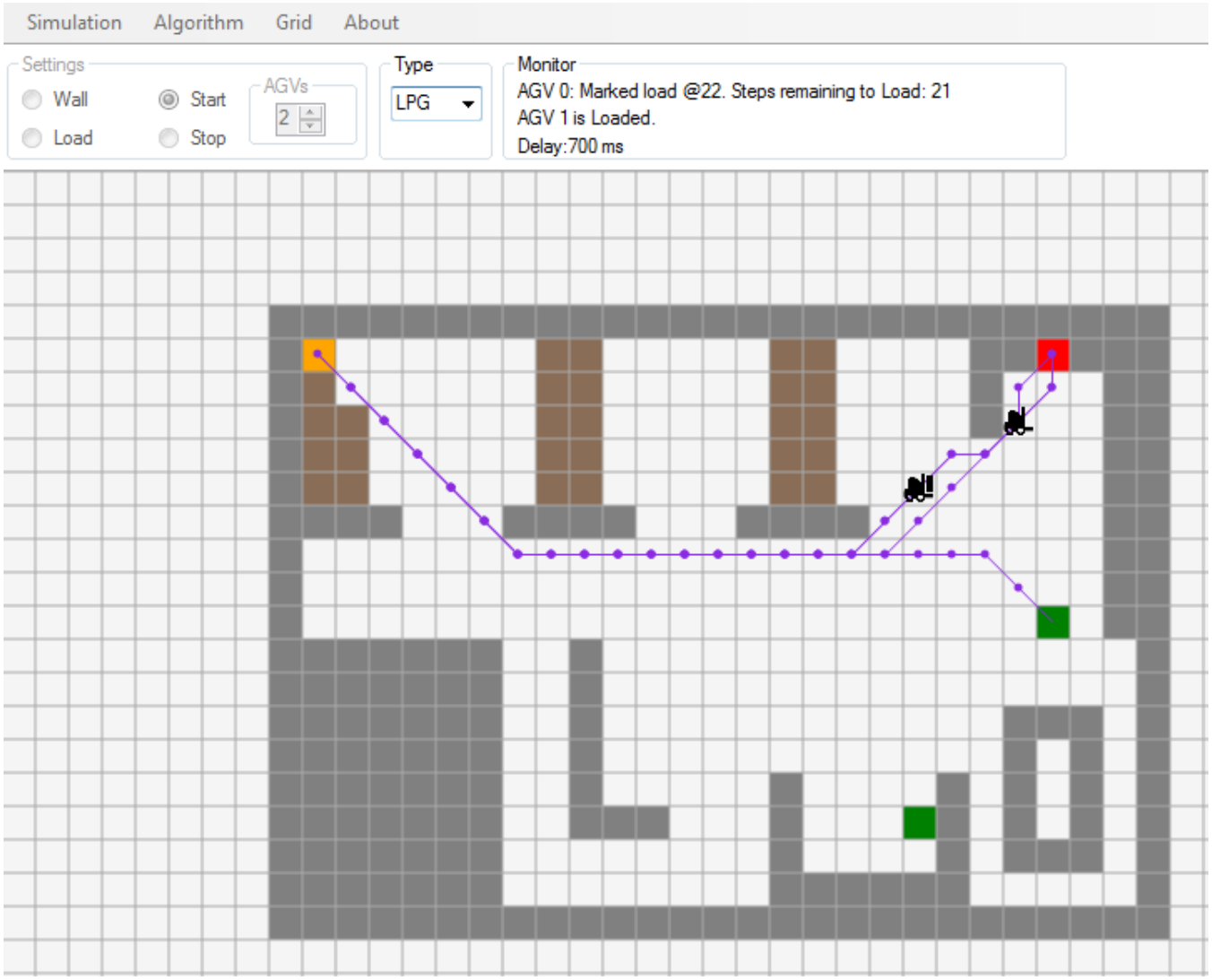
```
private void update_emissions(int whichAGV)
{

    if (cb_type.SelectedItem.ToString() == "LPG")
    {
        if (AGVs[whichAGV].Status.Busy)...
        else...
    }
    else if (cb_type.SelectedItem.ToString() == "DSL")...
    else
    { // ELE
        CO2 = 0;
        CO = 0;
        NOx = 0;
        THC = 0;
        if (AGVs[whichAGV].Status.Busy)
            GlobalWarming += 0.67;
        else
            GlobalWarming += 0.64;
    }

    emissions.CO2_label.Text = "CO2: " + Math.Round(CO2, 2) + " gr";
    emissions.CO_label.Text = "CO: " + Math.Round(CO, 2) + " gr";
    emissions.NOx_label.Text = "NOx: " + Math.Round(NOx, 2) + " gr";
    emissions.THC_label.Text = "THC: " + Math.Round(THC, 2) + " gr";
    emissions.Global_label.Text = "Global Warming eq: " + Math.Round(GlobalWarming, 2) + " kgr";
}
```

Εικόνα 55 Τμήμα κώδικα για τον υπολογισμό των εκπομπών

5.2.4. Διαδοχική παραλαβή φορτίων

Κατά την άφιξη του εκάστοτε οχήματος στην έξοδο και την εκφόρτωση του φορτίου που φέρει, ακολουθεί ο υπολογισμός της βέλτιστης διαδρομής προς το επόμενο διαθέσιμο φορτίο, εφόσον βέβαια αυτό υπάρχει, και αμέσως μόλις ολοκληρωθεί, το παραπάνω φορτίο χρωματίζεται με το αντίστοιχο

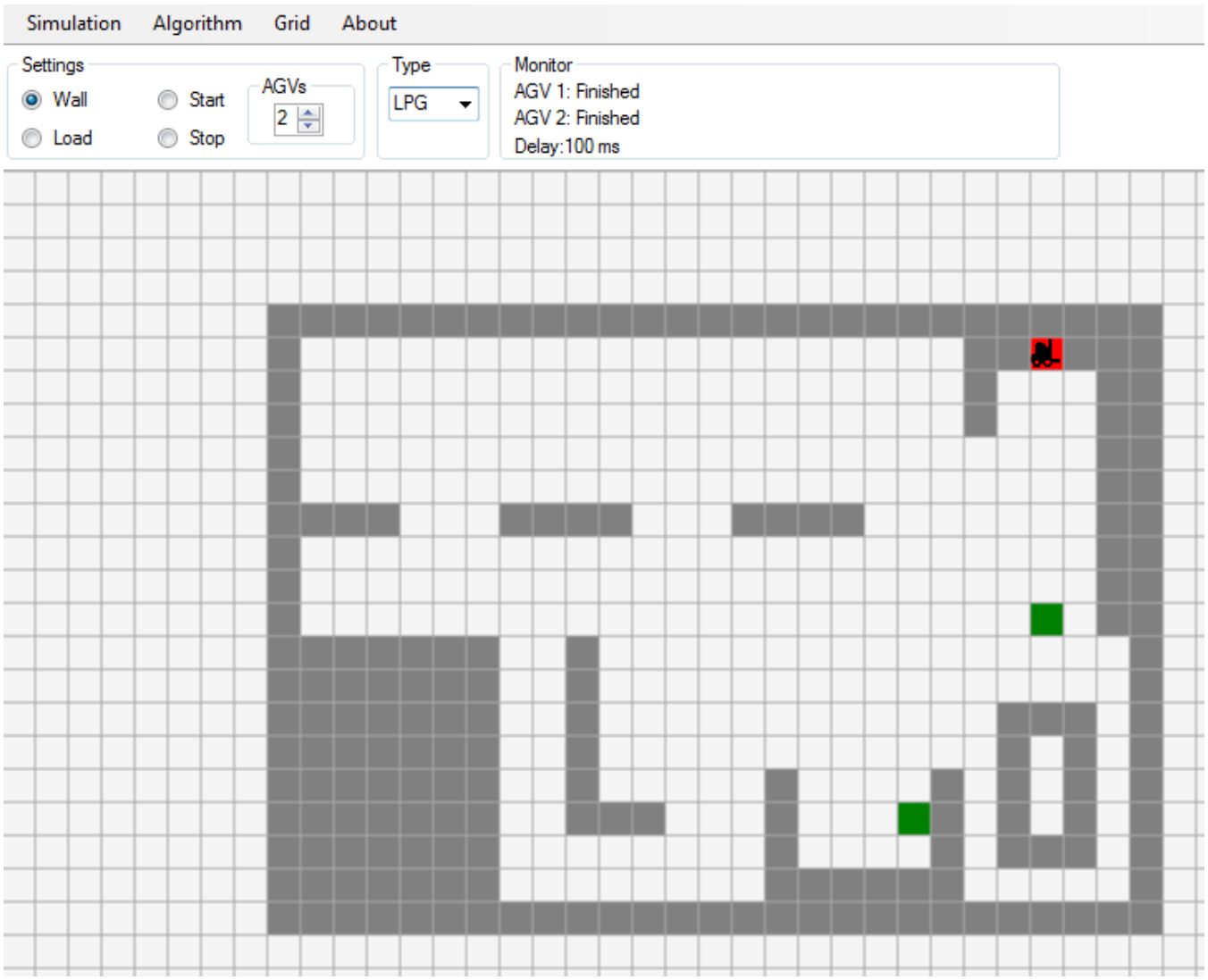
χρώμα δηλώνοντας την επικείμενη παραλαβή του. Η διαδικασία που μόλις περιγράψαμε, υλοποιείται μέσω της συνάρτησης “getNextLoad”.



Εικόνα 56 Λειτουργία συνάρτησης getNextLoad. Το δεύτερο όχημα οδηγεί στην έξοδο το φορτίο που παρέλαβε, κατά το πρώτο δρομολόγιό του, τη στιγή που το πρώτο όχημα ήδη έχει εκφορτώσει και κατευθύνεται προς το επόμενο διαθέσιμο προς παραλαβή φορτίο.

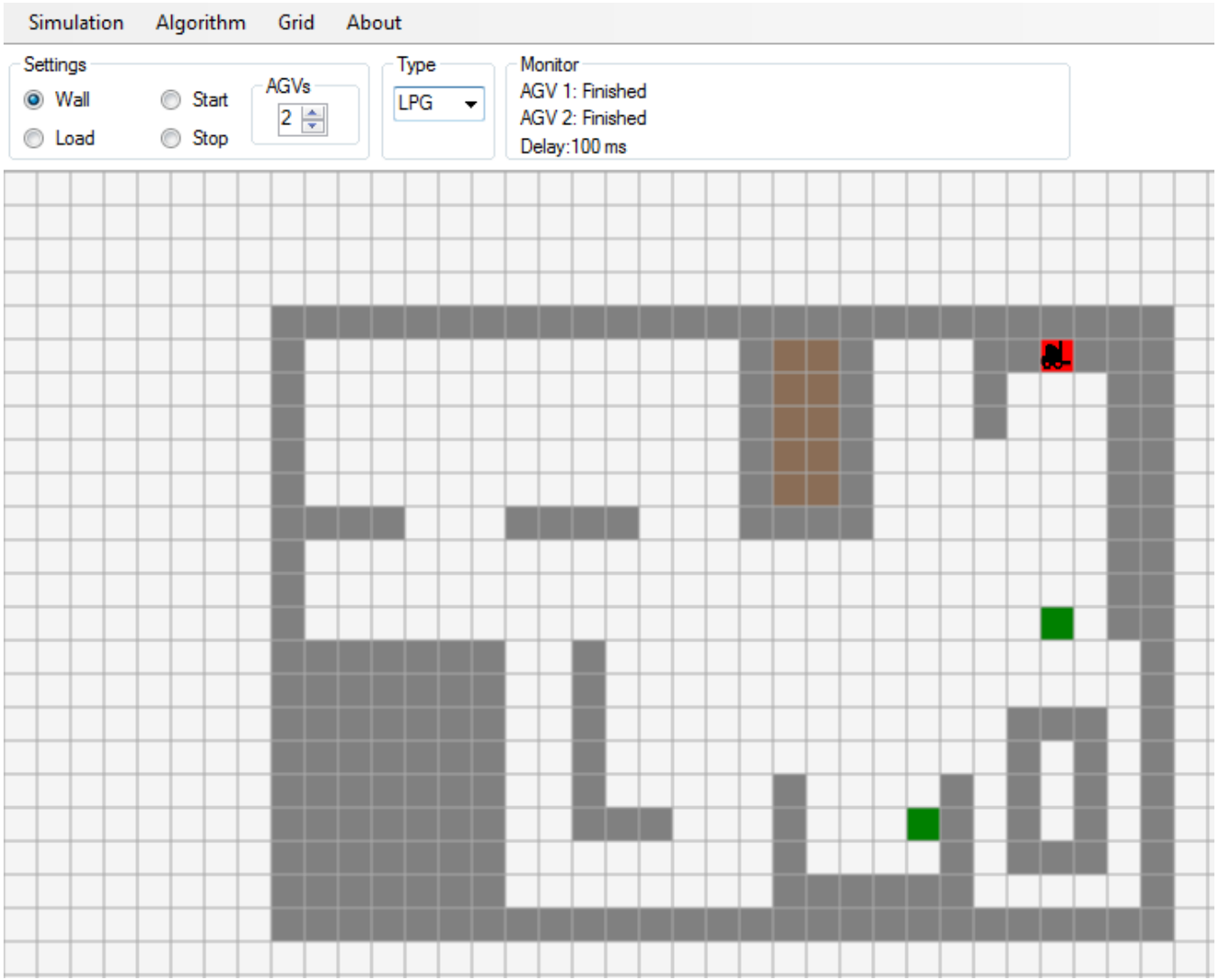
5.2.5. Ολοκλήρωση προσομοίωσης

Αυτή η διαδικασία συνεχίζεται μέχρις ότου να μην υπάρχουν άλλα διαθέσιμα φορτία μέσα στην αποθήκη μας. Μόλις, πραγματικά, δεν υπάρχει κανένα φορτίο, τα οχήματα παραμένουν στο τελικό σημείο εκφόρτωσης.



Εικόνα 57 Τέλος προσομοίωσης 1. Πλέον δεν υπάρχουν εναπομείναντα φορτία

Υπάρχει περίπτωση, τα φορτία στο χώρο εργασίας να μην έχουν εξαντληθεί αλλά να μην θεωρούνται διαθέσιμα. Η απόφαση για αυτό το ενδεχόμενο λαμβάνεται όταν οι αλγόριθμοι εύρεσης διαδρομής δεν μπορούν να δημιουργήσουν διαδρομή μεταξύ φορτίου και εξόδου.



Εικόνα 58 Τέλος προσομοίωσης 2. Τα φορτία που έχουν απομείνει, θεωρούνται μη διαθέσιμα καθώς περικλείονται από τοίχο.

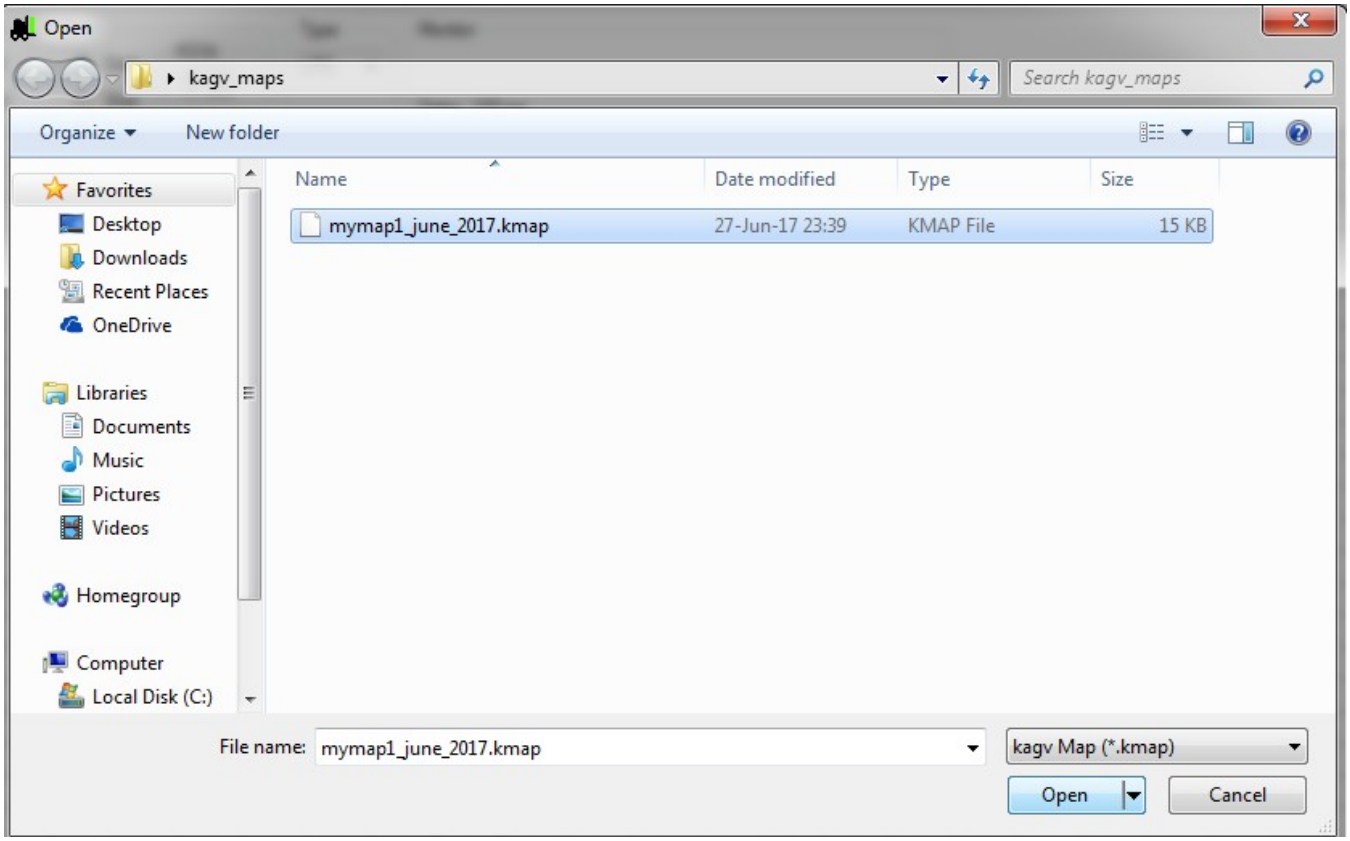
1.2.

5.3. Προετοιμασία περιβάλλοντος για την επόμενη προσομοίωση

Εάν ο χρήστης επιθυμεί να πραγματοποιήσει την ίδια διαδικασία με διαφορετικού τύπου οχήματα ή ακόμη και διαφορετικό πλήθος αυτών, χρησιμοποιεί την επιλογή Grid->Clear->All (ή απλώς με το πάτημα του “F5”) για να καθαρίσει το πλέγμα από όλα τα αντικείμενα που υπάρχουν μέσα σε αυτό.

Έπειτα ο χρήστης μπορεί να επαναλάβει τη διαδικασία σχεδίασης που περιγράψαμε παραπάνω τοποθετώντας, στο πλέγμα, τα αντικείμενα που τον ενδιαφέρουν. Αν ωστόσο, προηγουμένως, έχει αποθηκεύσει κάποιο συγκεκριμένο χώρο εργασίας με τον οποίο θέλει να πραγματοποιήσει την επόμενη προσομοίωση, έχει τη δυνατότητα να εισάγει το αντίστοιχο αρχείο όπως υποδεικνύεται παρακάτω.

Από το μενού “Simulation”, επιλέγει τη λειτουργία “Import map”. Από το διάλογο επιλογής αρχείου που θα εμφανιστεί, μπορεί να διαλέξει το αρχείο που θα χρησιμοποιήσει.



Εικόνα 59 Εισαγωγή αποθηκευμένου προσχεδιασμένου χάρτη

Τελικά παρατηρούμε πως η εφαρμογή έχει φορτώσει το χάρτη όπως ακριβώς τον αποθηκεύσαμε, χωρίς να χρειάζεται να το σχεδιάσουμε ξανά από την αρχή, καλώντας την αντίστοιχη συνάρτηση εισαγωγής “import”.

6. ΒΙΒΛΙΟΓΡΑΦΙΑ

I. Bechtsis, D., Tsolakis, N., Vlachos, D., Iakovou, E., 2017. Sustainable supply chain management in the digitalisation era: The impact of Automated Guided Vehicles. Journal of Cleaner Production, 142, 4, pp 3970-3984.

II. Bechtsis, D., Tsolakis, N., Vlachos, D., Srai, J.S.: Intelligent Autonomous Vehicles in digital supply chains: A framework for integrating innovations towards sustainable value networks, 2016. Special Issue: Innovation for sustainable development, Journal of Cleaner Production. Under review.

III. Bellifemine, G. Caire, D. Greenwood, 2007. Developing Multi-Agent Systems with JADE. John Wiley and Sons Ltd.

IV. Borenstein, J., Everett, H. R. and Feng, L. 1996. “Where am I? -- Systems and Methods for Mobile Robot Positioning”. The University of Michigan.

V. Dziwis, D., 2005. Automated/Self Guided Vehicles (AGV/SGV) and System Design Considerations.

VI. Fuc, P., Kurczewski, P., Lewandowska, A. et al. Int J Life Cycle Assess (2016) 21: 1438. https://doi.org/10.1007/s11367-016-1104-y

VII. Iris F., A. Vis, 2004. Survey of research in the design and control of automated guided vehicle systems. European journal of operational research.

VIII. Jon Skeet, 2013. C# in depth, third edition. Manning Publications.

IX. Wooldridge, M. Jennings, 1995. N.R.: Intelligent Agents: Theory and Practice. Knowledge Engineering Review 10, 115–152

X. Μανωλάκης Δημήτριος, Εργαστηριακές σημειώσεις, Προγραμματισμός ΙΙΙ, Α.Τ.Ε.Ι.Θ Τμήμα Μηχανικών Αυτοματισμού.

ΔικτυακοίΤόποι

Savant Automation (http://www.agvsystems.com)

http://www.marketsandmarkets.com/PressReleases/automated-guided-vehicle.asp

Material Handling Industry http://www.mhi.org/agvs

Foundation for Intelligent Physical Agents. URL: http://www.fipa.org/

Java Agent Development Framework. URL: http://jade.tilab.com/

C# Reference, URL: https://docs.microsoft.com/en-us/dotnet/csharp/language-reference/index

Visual Studio 2013, URL: https://msdn.microsoft.com/en-us/library/dd831853(v=vs.120).aspx

EpPathFinding for C#, URL: https://github.com/juhgiyo/EpPathFinding.cs

7. ΠΑΡΑΡΤΗΜΑΤΑ

7.1. Συναρτήσεις

Η εισαγωγή του κώδικα από το Visual Studio στο Word, έγινε μέσω του εργαλείου http://www.planetb.ca/syntax-highlight-word.

7.1.1. Initialization

Η συνάρτηση αυτή είναι υπεύθυνη για την αρχικοποίηση όλων των στοιχείων που εμπεριέχονται στην κύρια φόρμα και χωρίς τα οποία η εφαρμογή δε θα είχε καμία ουσία. Καλείται, για πρώτη φορά, κατά το άνοιγμα της εφαρμογής και έπειτα, χρησιμοποιείται είτε κάθε φορά που ο χρήστης επιλέγει να κάνει πλήρη καθαρισμό του πλέγματος είτε εφόσον επιλεχθεί η λειτουργία εισαγωγής αρχείου-χάρτη. Η τελευταία περίπτωση ελέγχεται, όπως βλέπουμε παρακάτω, στη σειρά τριάντα δύο (32) μέσω μιας μεταβλητής τύπου Boolean της οποίας η τιμή γίνεται «Αληθής» κατά την κλήση της αντίστοιχης συνάρτησης εισαγωγής.

```
1.  private void initialization() {
2.
3.              if ( Constants.__SemiTransparency)
4.                 Constants.__SemiTransparent = Color.FromArgb( Constants.__Opacity,Color.WhiteSm
    oke);
5.
6.              for (int i = 0; i < StartPos.Count; i++) {
7.                  AGVs[i] = new Vehicle(this);
8.                  AGVs[i].ID = i;
9.              }
10.
11.             this.DoubleBuffered = true;
12.             this.Width = ((Constants.__WidthBlocks + 1) * Constants.__BlockSide) - 3; //3 beca
    use 2=border and the 1 comes from "width+1"
13.             this.Height = (Constants.__HeightBlocks + 1) * Constants.__BlockSide + Constants._
    _BottomBarOffset;
14.             this.Size = new Size(this.Width, this.Height + Constants.__BottomBarOffset);
15.             this.MaximizeBox = false;
16.             this.FormBorderStyle = FormBorderStyle.FixedSingle;
17.
18.             //m_rectangles is an array of two 1d arrays
19.             //declares the length of the first 1d array
20.
21.             m_rectangles = new GridBox[Constants.__WidthBlocks][];
22.
23.             for (int widthTrav = 0; widthTrav < Constants.__WidthBlocks; widthTrav++) {
24.                 //declares the length of the seconds 1d array
25.                 m_rectangles[widthTrav] = new GridBox[Constants.__HeightBlocks];
26.                 for (int heightTrav = 0; heightTrav < Constants.__HeightBlocks; heightTrav++)
    {
27.
```

```
28.                     //dynamically add the gridboxes into the m_rectangles.
29.                     //size of the m_rectangels is constantly increasing(while adding
30.                     //the gridbox values) until
31.                     //size=height or size = width.
32.                     if (imported) {
33.                         m_rectangles[widthTrav][heightTrav] = new GridBox(widthTrav * Constant
    s.__BlockSide, heightTrav * Constants.__BlockSide + Constants.__TopBarOffset, importmap[widthT
    rav, heightTrav]);
34.                         if (importmap[widthTrav, heightTrav] == BoxType.Load) {
35.                             isLoad[widthTrav, heightTrav] = 1;
36.                             loads++;
37.                         }
38.                     } else {
39.                         m_rectangles[widthTrav][heightTrav] = new GridBox(widthTrav * Constant
    s.__BlockSide, heightTrav * Constants.__BlockSide + Constants.__TopBarOffset, BoxType.Normal);

40.                         isLoad[widthTrav, heightTrav] = 2;
41.                     }
42.
43.
44.                 }
45.             }
46.             if (imported)
47.                 imported = false;
48.
49.
50.
51.             searchGrid = new DynamicGridWPool(SingletonHolder<NodePool>.Instance);
52.             jumpParam = new JumpPointParam(searchGrid, false, crossCorners, crossAdjacent, mod
    e);
53.             //syntax of jumpParam-> JumpPointParam(searchGrid, startPos, endPos, cbCrossCorner
    s.Checked, HeuristicMode.EUCLIDEAN
54.
55.
56.         }
```

7.1.2. import

Παραπάνω αναφέραμε την επιλογή εισαγωγής αποθηκευμένου, σε ειδικού τύπου αρχείο, προσχεδιασμένου χάρτη. Η λειτουργία αυτή υλοποιείται μέσω της παρακάτω συνάρτησης και με τον εξής τρόπο. Μόλις ο χρήστης επιλέξει το αρχείο χώρου εργασίας που επιθυμεί να εισάγει στην εφαρμογή, το αρχείο αυτό διαβάζεται γραμμή προς γραμμή και, χρησιμοποιώντας διαχωριστές (delimiters), διασπάται έτσι ώστε η πληροφορία που αφορά στον τύπο του κάθε κελιού του πλέγματος να καταχωρηθεί σε έναν πίνακα “words” από όπου θα ελεγχθεί για να αναγνωριστεί ο τύπος του αντίστοιχου κελιού. Αμέσως μετά, η πληροφορία αυτή αποθηκεύεται σε έναν πίνακα “importmap” από τον οποίο, κατά την κλήση της συνάρτησης “initialization”, θα μεταφερθεί στο πλέγμα της κύριας φόρμας και θα δώσει υπόσταση σε αυτό.

```
1.  private void import() {
2.
3.              ofd_importmap.Filter = "kagv Map (*.kmap)|*.kmap";
4.              ofd_importmap.FileName = "";
5.
6.
7.              if (ofd_importmap.ShowDialog() == DialogResult.OK) {
8.                  FullyRestore();
9.
10.                 bool proceed = false;
11.                 StreamReader _tmp = new StreamReader(ofd_importmap.FileName);
12.                 if (Constants.__ResolutionMultiplier == 2)
13.                     if (_tmp.ReadToEnd().Contains("Width blocks: 128  Height blocks: 64"))
14.                         proceed = true;
15.                     else
16.                         proceed = false;
17.                 else if (Constants.__ResolutionMultiplier == 1)
18.                     if (_tmp.ReadToEnd().Contains("Width blocks: 64  Height blocks: 32"))
19.                         proceed = true;
20.                     else
21.                         proceed = false;
22.                 _tmp.Close();
23.
24.                 if (proceed) {
25.                     StreamReader reader = new StreamReader(ofd_importmap.FileName);
26.                     reader.ReadLine();
27.
28.                     imported = true;
29.
30.                     string map_details = reader.ReadLine();
31.
32.                     char[] delim = { ' ' };
33.                     string[] words = map_details.Split(delim);
34.
35.                     bool isNumber;
36.                     int _tempNumber;
37.                     int whichNumber = 1;
38.
39.                     int width_blocks = 0;
```

```
                    int height_blocks = 0;

                    foreach (string _s in words) {
                        isNumber = int.TryParse(_s, out _tempNumber);
                        if (isNumber) {
                            if (whichNumber == 1) {
                                width_blocks = Convert.ToInt32(_s);
                                whichNumber++;
                            } else if (whichNumber == 2)
                                height_blocks = Convert.ToInt32(_s);
                        }
                    }

                    reader.ReadLine();

                    importmap = new BoxType[width_blocks, height_blocks];
                    words = reader.ReadLine().Split(delim);

                    int starts_counter = 0;
                    for (int z = 0; z < importmap.GetLength(0); z++) {
                        int i = 0;
                        foreach (string _s in words)
                            if (i < importmap.GetLength(1)) {
                                if (_s == "Start") {
                                    importmap[z, i] = BoxType.Start;
                                    starts_counter++;
                                } else if (_s == "End")
                                    importmap[z, i] = BoxType.End;
                                else if (_s == "Normal")
                                    importmap[z, i] = BoxType.Normal;
                                else if (_s == "Wall")
                                    importmap[z, i] = BoxType.Wall;
                                else if (_s == "Load")
                                    importmap[z, i] = BoxType.Load;
                                i++;
                            }
                        if (z == importmap.GetLength(0) - 1) { } else
                            words = reader.ReadLine().Split(delim);
                    }
                    reader.Close();

                    for (int z = 0; z < importmap.GetLength(0); z++)
                        for (int i = 0; i < importmap.GetLength(1); i++)
                            m_rectangles[z][i].boxType = importmap[z, i];

                    nUD_AGVs.Value = starts_counter;
                    initialization();
                    Redraw();
                } else
                    MessageBox.Show(this, "You have chosen an incompatible file import.\r\
nPlease try again.", "", MessageBoxButtons.OK, MessageBoxIcon.Error);
            }
        }
```

7.1.3. export

Βέβαια, για να δημιουργηθεί το αρχείο στο οποίο θα εξαχθεί ο χώρος εργασίας, πρέπει να κληθεί η συνάρτηση “export”. Η λειτουργία της είναι απλή, καθώς ελέγχει τον δισδιάστατο πίνακα m_rectangles[ ] [ ] ο οποίος περιέχει όλες τις απαραίτητες πληροφορίες για το πλέγμα και αποθηκεύει, στο αρχείο εξαγωγής, τον τύπο του εκάστοτε κελιού.

```
1.  private void export() {
2.              sfd_exportmap.FileName = "";
3.              sfd_exportmap.Filter = "kagv Map (*.kmap)|*.kmap";
4.
5.              if (sfd_exportmap.ShowDialog() == DialogResult.OK) {
6.                  StreamWriter writer = new StreamWriter(sfd_exportmap.FileName);
7.                  writer.WriteLine("Map info:\r\nWidth blocks: " + Constants.__WidthBlocks + "
    Height blocks: " + Constants.__HeightBlocks + "\r\n");
8.                  for (int i = 0; i < Constants.__WidthBlocks; i++) {
9.                      for (int j = 0; j < Constants.__HeightBlocks; j++) {
10.                         writer.Write(m_rectangles[i][j].boxType + " ");
11.                     }
12.                     writer.Write("\r\n");
13.                 }
14.                 writer.Close();
15.             }
16.
17.         }
```

7.1.4. main_form_MouseClick

Η συνάρτηση αυτή καλείται κάθε φορά που ο χρήστης χρησιμοποιεί το αριστερό κουμπί του ποντικιού του επάνω στην κύρια φόρμα. Σε πλήρη εξάρτηση με το είδος αντικειμένου που θέλει να εισάγει στο πλέγμα (όχημα, φορτίο κλπ), κάνοντας αριστερό κλικ σε κάποιο κελί ο τύπος του συγκεκριμένου κελιού αλλάζει σε αυτό που επέλεξε ο χρήστης.

```
1.  private void main_form_MouseClick(object sender, MouseEventArgs e) {
2.
3.              if (timer0.Enabled || timer1.Enabled || timer2.Enabled || timer3.Enabled || timer4
    .Enabled) return;
4.
5.
6.              Point click_coords = new Point(e.X, e.Y);
7.              if (!isvalid(click_coords) || e.Button != MouseButtons.Left || nUD_AGVs.Value == 0
    )
8.                  return;
9.
10.             if (rb_load.Checked) {
11.                 for (int widthTrav = 0; widthTrav < Constants.__WidthBlocks; widthTrav++) {
12.                     for (int heightTrav = 0; heightTrav < Constants.__HeightBlocks; heightTrav
    ++) {
13.                         if (m_rectangles[widthTrav][heightTrav].boxRec.IntersectsWith(new Rect
    angle(e.Location, new Size(1, 1)))) {
14.                             m_lastBoxType = m_rectangles[widthTrav][heightTrav].boxType;
15.                             m_lastBoxSelect = m_rectangles[widthTrav][heightTrav];
16.                             switch (m_lastBoxType) {
17.                                 case BoxType.Normal:
18.                                     m_rectangles[widthTrav][heightTrav].SwitchLoad();
19.                                     isLoad[widthTrav, heightTrav] = 1;
20.                                     this.Invalidate();
21.                                     break;
22.                                 case BoxType.Load:
23.                                     loads--;
24.                                     m_rectangles[widthTrav][heightTrav].SwitchLoad();
25.                                     isLoad[widthTrav, heightTrav] = 2;
26.                                     this.Invalidate();
27.                                     break;
28.                                 case BoxType.Wall:
29.                                 case BoxType.Start:
30.                                 case BoxType.End:
31.                                     break;
32.                             }
33.                         }
34.
35.
36.                     }
37.                 }
38.             }
39.             if (rb_start.Checked) {
40.
41.                 if (nUD_AGVs.Value == 1)//Saves only the last Click position to place the Star
    t (1 start exists)
42.                 {
43.                     for (int widthTrav = 0; widthTrav < Constants.__WidthBlocks; widthTrav++)
```

```
                        for (int heightTrav = 0; heightTrav < Constants.__HeightBlocks; height
    Trav++)
                            if (m_rectangles[widthTrav][heightTrav].boxType == BoxType.Start)
                                m_rectangles[widthTrav][heightTrav].SwitchEnd_StartToNormal();

                } else if (nUD_AGVs.Value > 1)//Deletes the start with the smallest iX - iY co
    ords and keeps the rest
                {
                    int starts_counter = 0;
                    int[,] starts_position = new int[2, Convert.ToInt32(nUD_AGVs.Value)];


                    for (int widthTrav = 0; widthTrav < Constants.__WidthBlocks; widthTrav++)
                        for (int heightTrav = 0; heightTrav < Constants.__HeightBlocks; height
    Trav++) {
                            if (m_rectangles[widthTrav][heightTrav].boxType == BoxType.Start)
    {
                                starts_position[0, starts_counter] = widthTrav;
                                starts_position[1, starts_counter] = heightTrav;
                                starts_counter++;
                            }
                            if (starts_counter == nUD_AGVs.Value) {
                                m_rectangles[starts_position[0, 0]][starts_position[1, 0]].Swi
    tchEnd_StartToNormal();
                            }
                        }

                }


                //Converts the clicked box to Start point
                for (int widthTrav = 0; widthTrav < Constants.__WidthBlocks; widthTrav++)
                    for (int heightTrav = 0; heightTrav < Constants.__HeightBlocks; heightTrav
    ++)
                        if (m_rectangles[widthTrav][heightTrav].boxRec.Contains(click_coords)
                            &&
                            m_rectangles[widthTrav][heightTrav].boxType == BoxType.Normal) {
                                m_rectangles[widthTrav][heightTrav] = new GridBox(widthTrav *
    Constants.__BlockSide, heightTrav * Constants.__BlockSide + Constants.__TopBarOffset, BoxType.
    Start);


                        }


            }
            //same for Stop
            if (rb_stop.Checked) {
                for (int widthTrav = 0; widthTrav < Constants.__WidthBlocks; widthTrav++)
                    for (int heightTrav = 0; heightTrav < Constants.__HeightBlocks; heightTrav
    ++)
                        if (m_rectangles[widthTrav][heightTrav].boxType == BoxType.End)
                            m_rectangles[widthTrav][heightTrav].SwitchEnd_StartToNormal();//
    allow only one end point


                for (int widthTrav = 0; widthTrav < Constants.__WidthBlocks; widthTrav++)
                    for (int heightTrav = 0; heightTrav < Constants.__HeightBlocks; heightTrav
    ++)
                        if (m_rectangles[widthTrav][heightTrav].boxRec.Contains(click_coords)
                             &&
```

```
93.                             m_rectangles[widthTrav][heightTrav].boxType == BoxType.Normal) {
94.                                 m_rectangles[widthTrav][heightTrav] = new GridBox(widthTrav *
    Constants.__BlockSide, heightTrav * Constants.__BlockSide + Constants.__TopBarOffset, BoxType.
    End);
95.                             }
96.             }
97.
98.
99.
100.                    this.Invalidate();
101.        }
```

7.1.5. Redraw

Περνώντας στη λογική που κρύβεται πίσω από τα γραφικά, συναντάμε τη συνάρτηση “Redraw”. Είναι η μεγαλύτερη, σε έκταση, συνάρτηση που αναπτύχθηκε και ίσως η περιπλοκότερη, οπότε θα σταθούμε στην ανάλυσή της, παραπάνω απ’όσο χρειάστηκε για τις προηγούμενες. Κύρια λειτουργία της είναι να χρησιμοποιήσει τους αλγορίθμους εύρεσης (A*) και να παράγει τη βέλτιστη διαδρομή προς το στόχο που έχει κάθε όχημα.

Αρχικά, ελέγχει το πλέγμα της κύριας φόρμας κελί προς κελί και ανάλογα με τον τύπο που συναντάει, εκτελεί κάποιες εντολές. Εντοπίζοντας κελιά τύπου “Wall”, τα καθιστά μη προσβάσιμα ενώ κάνει το αντίθετο για τα άδεια. Οι συντεταγμένες των κελιών που δηλώνουν αρχικό σημείο οχήματος, αποθηκεύονται σε μία λίστα τύπου “GridPos” ενώ παράλληλα γίνεται δήλωση του αντικειμένου “Vehicle” σε μία λίστα τύπου “Vehicle”. Για κάθε κελί που εμφανίζεται ως φορτίο, το αντίστοιχο κελί ενός πίνακα “isLoad[,]” παίρνει τον αριθμό 1, ένας μετρητής αυξάνεται και οι συντεταγμένες του κελιού αυτού προστίθενται σε μία λίστα “loadPos”. Πρέπει να σημειωθεί πως οι πιθανές τιμές του πίνακα που μόλις αναφέραμε, είναι οι εξής: 1=διαθέσιμο φορτίο, 2=δεν είναι φορτίο, 3=φορτίο δηλωμένο προς παραλαβή, 4=προσωρινά παγιδευμένο φορτίο. Εφόσον στο πλέγμα εντοπιστεί έστω και ένα κελί που να είναι φορτίο, η τιμή της boolean μεταβλητής “mapHasLoads” γίνεται “true”.

Με το τέλος της ανάγνωσης του πλέγματος, πραγματοποιείται έλεγχος για να εντοπιστούν τα οχήματα που βρίσκονται σε θέση να συμμετέχουν στην προσομοίωση. Στη συνέχεια, η μεταβλητή “mapHasLoads” θα καθορίσει την εκτέλεση, ή μη, ελέγχου για φορτία τα οποία περιβάλλονται από κελιά που αποτελούν εμπόδιο (Wall). Κατά τον έλεγχο αυτό, υπάρχει περίπτωση η τιμή της παραπάνω μεταβλητής να αλλάξει σε “false”, γεγονός που θα οδηγήσει στην επιλογή σεναρίου όπου στο χώρο εργασίας δεν υπάρχουν φορτία προς παραλαβή. Έπειτα, ακολουθεί δεύτερος έλεγχος για φορτία που είναι προσωρινά παγιδευμένα (ή με άλλα λόγια, περιβάλλονται από άλλα φορτία). Αφού πραγματοποιηθούν οι απαραίτητοι έλεγχοι, ξεκινάει η διαδικασία εύρεσης βέλτιστης διαδρομής για κάθε διαθέσιμο όχημα.

Για το σενάριο λοιπόν που, όντως, στο facility layout υπάρχουν διαθέσιμα προς παραλαβή φορτία, θα εκτελεστεί το τμήμα κώδικα που ανήκει την αληθή περίπτωση (case: true σειρά 102). Οι εντολές στις σειρές 127 και 128 είναι αυτές που θα ενεργοποιήσουν τους ευρετικούς αλγορίθμους ώστε να δημιουργήσουν τη βέλτιστη διαδρομή από την αρχική θέση του κάθε οχήματος μέχρι το φορτίο που αιτείται παραλαβής. Τα σημεία (“JumpPoints”) που θα επιστραφούν από τους αλγορίθμους εύρεσης, θα

εισαχθούν στη λίστα βημάτων της κλάσης Vehicle. Αμέσως μετά, ακολουθεί ο υπολογισμός του τμήματος διαδρομής από το φορτίο μέχρι την έξοδο, όπου τηρείται παρόμοια διαδικασία.

Στο σενάριο που ο χρήστης δεν έχει εισάγει φορτία στο πλέγμα, γίνεται υπολογισμός μόνο για τη διαδρομή μεταξύ των αρχικών και του τελικού σημείου. Ο τρόπος εύρεσης της διαδρομής παραμένει ίδιος, απλά πλέον είναι απλουστευμένος καθώς ο υπολογισμός στο πρώτο σενάριο πραγματοποιείται για δύο διαδρομές οι οποίες στο τέλος ενοποιούνται σε μία καθολική διαδρομή για κάθε όχημα, τη στιγμή που στη δεύτερη περίπτωση η διαδρομή που πρέπει να υπολογιστεί είναι συγκεκριμένη και αφορά το τμήμα αρχικό-τελικό σημείο.

```
1.  private void Redraw() {
2.
3.              bool start_found = false;
4.              bool end_found = false;
5.              mapHasLoads = false;
6.
7.              GridPos endPos = new GridPos();
8.
9.              pos_index = 0;
10.             StartPos = new List<GridPos>();
11.             AGVs = new List<Vehicle>();
12.             loadPos = new List<GridPos>();
13.
14.
15.
16.             for (int i = 0; i < Constants.__WidthBlocks; i++)
17.                 for (int j = 0; j < Constants.__HeightBlocks; j++) {
18.                     if (m_rectangles[i][j].boxType == BoxType.Wall)
19.                         searchGrid.SetWalkableAt(new GridPos(i, j), false);
20.                     else
21.                         searchGrid.SetWalkableAt(new GridPos(i, j), true);
22.
23.                     if (beforeStart) {
24.                         if (m_rectangles[i][j].boxType == BoxType.Start)
25.                             searchGrid.SetWalkableAt(new GridPos(i, j), false);
26.
27.                     } else {
28.                         if (m_rectangles[i][j].boxType == BoxType.Start)
29.                             searchGrid.SetWalkableAt(new GridPos(i, j), true);
30.                     }
31.
32.                     if (m_rectangles[i][j].boxType == BoxType.Load)
33.                     {
34.                         mapHasLoads = true;
35.                         searchGrid.SetWalkableAt(new GridPos(i, j), false);
36.                         isLoad[i, j] = 1;
37.                         loads++;
38.                         loadPos.Add(new GridPos(i, j));
39.                     }
40.                     if (m_rectangles[i][j].boxType == BoxType.Normal)
41.                         m_rectangles[i][j].onHover(boxDefaultColor);
42.
43.                     if (m_rectangles[i][j].boxType == BoxType.Start) {
44.                         start_found = true;
```

```
45.
46.                         AGVs.Add(new Vehicle(this));
47.                         AGVs[pos_index].ID = pos_index;
48.
49.                         StartPos.Add(new GridPos(0, 0));//create space to add the next agv
50.                         StartPos[pos_index].x = i;
51.                         StartPos[pos_index].y = j;
52.
53.                         a = StartPos[pos_index].x;
54.                         b = StartPos[pos_index].y;
55.
56.                         if (pos_index < StartPos.Count) {
57.                             StartPos[pos_index] = new GridPos(StartPos[pos_index].x, StartPos[
    pos_index].y);
58.                             pos_index++;
59.                         }
60.                     }
61.
62.                     if (m_rectangles[i][j].boxType == BoxType.End) {
63.                         end_found = true;
64.                         endPos.x = i;
65.                         endPos.y = j;
66.                         endPointCoords = new Point(i * Constants.__BlockSide, j * Constants.__
    BlockSide + Constants.__TopBarOffset);
67.                     }
68.                 }
69.             Reset();
70.             if (!start_found || !end_found)
71.                 return;
72.
73.
74.             NoJumpPointsFound = true;
75.
76.
77.             pos_index = 0;
78.             is_trapped = new bool[StartPos.Count, 2];
79.
80.
81.             if (AGVs != null)
82.                 for (int i = 0; i < AGVs.Count(); i++)
83.                     if (AGVs[i] != null) {
84.                         AGVs[i].updateAGV();
85.                         AGVs[i].Status.Busy = false;//initialize the status of AGVs, as 'avail
    able'
86.                     }
87.
88.             //replaces current List with ALL AGVs with a list that
89.             //contains only NOT trapped AGVs
90.             StartPos = NotTrappedVehicles(StartPos, endPos);
91.             if(mapHasLoads)
92.                 KeepValidLoads(endPos);
93.
94.             for (int i = 0; i < StartPos.Count; i++) {
95.
96.                 if (AGVs[i].Status.Busy == false) {
97.                     if (loadPos.Count() == 0)
98.                         mapHasLoads = false;
99.
100.                        //create the path from start to load,if load exists===========
101.                        switch (mapHasLoads) {
102.                            case true:
```

```
103.
104.
105.
106.                                     checkForTrappedLoads(loadPos);
107.                                     if (loadPos.Count() == 0)
108.                                         loadPos.Add(endPos); //if EVERY load is trapped, use th
    e endPos as LoadPos so as the agvs can complete their basic route (start -> end)
109.
110.
111.                                     //Do not allow walk over any other load except the targeted
    one
112.                                     for (int k = 0; k < Constants.__WidthBlocks; k++)
113.                                     {
114.                                         for (int l = 0; l < Constants.__HeightBlocks; l++)
115.                                         {
116.                                             if (m_rectangles[k][l].x != (loadPos[0].x * Constan
    ts.__BlockSide)
117.                                                 && m_rectangles[k][l].y != (loadPos[0].y * Cons
    tants.__BlockSide)
118.                                                 && m_rectangles[k][l].boxType == BoxType.Load)
119.                                             {
120.                                                 searchGrid.SetWalkableAt(new GridPos(k, l), fal
    se);
121.                                             }
122.
123.                                         }
124.                                     }
125.
126.
127.                                     jumpParam.Reset(StartPos[pos_index], loadPos[0]);
128.                                     JumpPointsList = JumpPointFinder.FindPath(jumpParam, paper)
    ;
129.                                     AGVs[i].Status.Busy = true;
130.
131.                                     for (int k = 0; k < Constants.__WidthBlocks; k++)
132.                                     {
133.                                         for (int l = 0; l < Constants.__HeightBlocks; l++)
134.                                         {
135.                                             if (m_rectangles[k][l].x != (loadPos[0].x * Constan
    ts.__BlockSide)
136.                                                 && m_rectangles[k][l].y != (loadPos[0].y * Cons
    tants.__BlockSide)
137.                                                 && m_rectangles[k][l].boxType == BoxType.Load)
138.                                             {
139.                                                 searchGrid.SetWalkableAt(new GridPos(k, l), fal
    se);
140.
141.                                             }
142.
143.                                         }
144.                                     }
145.
146.                                     //is_trapped[i,0] -> part of route agv -> load
147.                                     //is_trapped[i,1] -> part of route load -> end
148.                                     if (JumpPointsList.Count == 0)
149.                                         is_trapped[i, 0] = true;
150.                                     else
151.                                         is_trapped[i, 0] = false;
152.
153.                                     if (!is_trapped[i, 0])
154.                                         for (int j = 0; j < JumpPointsList.Count; j++)
```

```
155.                                         AGVs[i].JumpPoints.Add(JumpPointsList[j]);
156.                                     else //leak catch
157.                                         AGVs[i].JumpPoints.Add(new GridPos());  //increases the
    size of the AGV's embedded List so the JumpPoints can fit without causing overflow
158.
159.
160.
161.
162.                                     //from load to end==============
163.
164.                                     //Do not allow walk over any other load except the targeted
    one
165.                                     for (int k = 0; k < Constants.__WidthBlocks; k++) {
166.                                         for (int l = 0; l < Constants.__HeightBlocks; l++) {
167.                                             if (m_rectangles[k][l].boxType == BoxType.Load) {
168.                                                 searchGrid.SetWalkableAt(new GridPos(k, l), fal
    se);
169.                                             }
170.
171.                                         }
172.                                     }
173.
174.                                     jumpParam.Reset(loadPos[0], endPos);
175.                                     JumpPointsList = JumpPointFinder.FindPath(jumpParam, paper)
    ;
176.
177.
178.                                     if (JumpPointsList.Count == 0)//recheck if load-to-end is a
    trapped path
179.                                         is_trapped[i, 1] = true;
180.                                     else
181.                                         is_trapped[i, 1] = false;
182.
183.
184.
185.                                     if (!is_trapped[i, 1])
186.                                         for (int j = 0; j < JumpPointsList.Count; j++) {
187.                                             AGVs[i].JumpPoints.Add(JumpPointsList[j]);  //adds
    the list containing the AGV's path, to the AGV's embedded JumpPoint List
188.                                             NoJumpPointsFound = false;
189.                                         }
190.
191.
192.                                     if (!is_trapped[i, 0] && !is_trapped[i, 1]) {
193.                                         //marks the load that each AGV picks up on the 1st rout
    e, as 3, so each agv knows where to go after delivering the 1st load
194.                                         if (fromstart[i]) {
195.                                             for (int widthtrav = 0; widthtrav < Constants.__Wid
    thBlocks; widthtrav++)
196.                                                 for (int heightrav = 0; heightrav < Constants._
    _HeightBlocks; heightrav++)
197.                                                     if (isLoad[widthtrav, heightrav] == 1) {
198.                                                         isLoad[widthtrav, heightrav] = 3;
199.                                                         AGVs[i].MarkedLoad = new Point(widthtra
    v, heightrav);
200.
201.                                                         widthtrav = Constants.__WidthBlocks;
202.                                                         heightrav = Constants.__HeightBlocks;
203.                                                     }
204.                                         }
205.                                         loadPos.Remove(loadPos[0]);
```

```
206.                                    } else if (is_trapped[i, 1])
207.                                        loadPos.Remove(loadPos[0]);
208.
209.
210.
211.
212.
213.                                    break;
214.
215.                                case false:
216.                                    jumpParam.Reset(StartPos[pos_index], endPos);
217.                                    JumpPointsList = JumpPointFinder.FindPath(jumpParam, paper)
    ;
218.
219.                                    if (JumpPointsList.Count == 0)
220.                                        is_trapped[i, 0] = true;
221.                                    else
222.                                        is_trapped[i, 0] = false;
223.
224.
225.                                    if (!is_trapped[i, 0])
226.                                        for (int j = 0; j < JumpPointsList.Count; j++) {
227.                                            AGVs[i].JumpPoints.Add(JumpPointsList[j]);  //adds
    the list containing the AGV's path, to the AGV's embedded JumpPoint List
228.
229.                                            NoJumpPointsFound = false;
230.                                        } else //leak catch
231.                                        AGVs[i].JumpPoints.Add(new GridPos()); //increases the
    size of the AGV's embedded List so the JumpPoints can fit without causing overflow
232.
233.                                    break;
234.                            }
235.                        }
236.                        pos_index++;
237.                    }
238.
239.                    int c = 0;
240.                    for (int i = 0; i < StartPos.Count; i++)
241.                        c += AGVs[i].JumpPoints.Count;
242.
243.                    for (int i = 0; i < StartPos.Count; i++)
244.                        for (int j = 0; j < AGVs[i].JumpPoints.Count - 1; j++) {
245.                            GridLine line = new GridLine(m_rectangles[AGVs[i].JumpPoints[j].x]
    [AGVs[i].JumpPoints[j].y],
246.                                                         m_rectangles[AGVs[i].JumpPoints[j + 1].
    x][AGVs[i].JumpPoints[j + 1].y]
247.                                                        );
248.
249.
250.                            AGVs[i].Paths[j] = line;
251.
252.                        }
253.
254.                    for (int i = 0; i < StartPos.Count; i++) {
255.                        if ((c - 1) > 0) {
256.                            Array.Resize(ref AGVs[i].Paths, c - 1);
257.                        }
258.                    }
259.
260.                    this.Invalidate();
261.       }
```

7.1.6. NotTrappedVehicles

Ένας από τους βασικούς ελέγχους που πραγματοποιούνται στη συνάρτηση “Redraw”, γίνεται προκειμένου να ληφθεί η απόφαση για το εάν ένα όχημα είναι διαθέσιμο να συμμετέχει στην προσομοίωση. Αυτό κρίνεται από την δυνατότητά του να διεκπεραιώσει την αποστολή του η οποία, στην πλέον απλή περίπτωση, είναι να οδηγήσει τον εαυτό του στην έξοδο. Για να συμβεί όμως αυτό, το όχημα θα πρέπει να μην είναι παγιδευμένο ή, με άλλα λόγια, να μπορούν οι ευρετικοί αλγόριθμοι να δημιουργήσουν διαδρομή ανάμεσα σε αυτό και στην έξοδο. Εάν κάτι τέτοιο δεν είναι εφικτό, τότε το όχημα αφαιρείται από τη λίστα που περιέχει όλα τα οχήματα και μένει αποκλεισμένο από την προσομοίωση.

```
1.  private List<GridPos> NotTrappedVehicles(List<GridPos> Vehicles, GridPos End) {
2.              int list_index = 0;
3.              int trapped_index = 0;
4.              bool removed;
5.
6.              for (int i = 0; i < TrappedStatus.Length; i++)
7.                  TrappedStatus[i] = true;
8.
9.              do {
10.                 removed = false;
11.                 jumpParam.Reset(Vehicles[list_index], End);
12.
13.                 if (JumpPointFinder.FindPath(jumpParam, paper).Count == 0)
14.                 {
15.                     Vehicles.Remove(Vehicles[list_index]);
16.                     AGVs.Remove(AGVs[list_index]);
17.                     removed = true;
18.                 }
19.                 else
20.                     TrappedStatus[trapped_index] = false;
21.
22.                 if (!removed) {
23.                     AGVs[list_index].ID = list_index;
24.                     list_index++;
25.                 }
26.                 trapped_index++;
27.             }
28.             while (list_index < Vehicles.Count);
29.             return Vehicles; //list with NOT TRAPPED AGVs' starting points (trapped AGVs have
    been removed)
30.         }
```

7.1.7. KeepValidLoads

Παρόμοιος με τον παραπάνω έλεγχο, πραγματοποιείται για την αξιολόγηση των φορτίων που περιέχονται στο χώρο εργασίας. Καθότι τα φορτία εκδίδουν αίτημα παραλαβής προς τα οχήματα, γνωστοποιώντας σε αυτά τη διαθεσιμότητά τους, πρέπει να εξασφαλιστεί ότι μόνο όσα φορτία δεν είναι εγκλωβισμένα από κελιά τύπου “Wall” θα είναι ικανά να ζητούν την παραλαβή τους από κάποιο όχημα. Πιο συγκεκριμένα, αρχικά, όλα τα φορτία θεωρούνται έγκυρα και εισάγονται σε μία λίστα, από την οποία ελέγχεται εάν κάθε ένα από αυτά μπορεί να οδηγηθεί στην έξοδο. Όσα δε τηρούν την προϋπόθεση αυτή, αφαιρούνται από την παραπάνω λίστα και το κελί του πίνακα “isLoad[,]”, το οποίο αντιστοιχεί στο εκάστοτε εγκλωβισμένο φορτίο, παίρνει την τιμή “2” και θεωρείται ως μη διαθέσιμο φορτίο. Στην περίπτωση που όλα τα φορτία είναι αποκλεισμένα από την έξοδο, τότε θα εκτελεστεί το τμήμα κώδικα, στη συνάρτηση “Redraw”, το οποίο αφορά το σενάριο κατά το οποίο δεν υπάρχουν φορτία στο χώρο εργασίας.

```
1.  private void KeepValidLoads(GridPos EndPoint)
2.          {
3.              int list_index = 0;
4.              bool removed;
5.
6.              for (int i = 0; i < loadPos.Count; i++)
7.                  searchGrid.SetWalkableAt(loadPos[i], true);
8.
9.              do
10.             {
11.                 removed = false;
12.                 jumpParam.Reset(loadPos[list_index], EndPoint);
13.                 if(JumpPointFinder.FindPath(jumpParam,paper).Count==0)
14.                 {
15.                     isLoad[loadPos[list_index].x, loadPos[list_index].y] = 2;
16.                     loads--;
17.                     loadPos.RemoveAt(list_index);
18.                     removed = true;
19.                 }
20.                 if (!removed)
21.                     list_index++;
22.             }while(list_index< loadPos.Count);
23.
24.             for (int i = 0; i < loadPos.Count; i++)
25.                 searchGrid.SetWalkableAt(loadPos[i], false);
26.
27.             if (loadPos.Count == 0)
28.                 mapHasLoads = false;
29.         }
```

7.1.8. checkForTrappedLoads

Αφού ολοκληρωθεί ο έλεγχος για να αποσαφηνιστεί το πλήθος των, πραγματικά, διαθέσιμων φορτίων στο χώρο εργασίας, ακολουθεί ένας ακόμη έλεγχος για να αποκλείσει, προσωρινά, όσα φορτία δεν είναι προσβάσιμα λόγω της θέσης τους. Ένα φορτίο μπορεί να περιβάλλεται από πολλά φορτία, με αποτέλεσμα ένα όχημα να μπορεί να το παραλάβει μόνο εφόσον μετακινηθούν τα υπόλοιπα που το εγκλωβίζουν. Έτσι λοιπόν, ακολουθώντας την ίδια λογική με την παραπάνω συνάρτηση, τα φορτία για τα οποία οι ευρετικοί αλγόριθμοι δεν μπορούν να δημιουργήσουν διαδρομή προς την έξοδο, χαρακτηρίζονται ως μη διαθέσιμα, έως ότου απεγκλωβιστούν, αλλάζοντας την τιμή τους στον αντίστοιχο πίνακα “isLoad[,]” σε “4”.

```
1.  private void checkForTrappedLoads(List<GridPos> pos) {
2.              int list_index = 0;
3.              bool removed;
4.
5.              //if the 1st AGV  cannot reach a Load, then that Load is
6.              //removed from the loadPos and not considered as available - marked as "4"
7.              do
8.              {
9.                  removed = false;
10.                 searchGrid.SetWalkableAt(new GridPos(pos[list_index].x, pos[list_index].y), tr
    ue);
11.                 jumpParam.Reset(StartPos[0], pos[list_index]);
12.                 if (JumpPointFinder.FindPath(jumpParam, paper).Count == 0)
13.                 {
14.                     searchGrid.SetWalkableAt(new GridPos(pos[list_index].x, pos[list_index].y)
    , false);
15.                     isLoad[pos[list_index].x, pos[list_index].y] = 4;
16.                     pos.Remove(pos[list_index]);
17.                     removed = true;
18.                 }
19.                 else
20.                 {
21.                     isLoad[pos[list_index].x, pos[list_index].y] = 1;
22.                 }
23.
24.                 if (!removed)
25.                     list_index++;
26.             } while (list_index < pos.Count);
27.
28.         }
```

7.1.9. main_form_Paint (event)

Η συγκεκριμένη συνάρτηση αποτελεί event της κύριας φόρμας. Το συγκεκριμένο event, καλείται κάθε φορά που γίνεται κάποια γραφική μεταβολή επάνω στην κύρια φόρμα. Τέτοιες μεταβολές μπορεί να είναι πολλές, όπως για παράδειγμα η εναλλαγή χρώματος ενός button όταν ο χρήστης οδηγεί τον κέρσορα του ποντικιού επάνω σε αυτό. Συνεπώς, κάθε φορά που ο χρήστης επεμβαίνει στην κύρια φόρμα και σχεδιάζει οτιδήποτε, η συνάρτηση αυτή καλείται και εκτελεί μια σειρά εντολών. Συνοπτικά λοιπόν, αρχικά δηλώνει τα γραφικά που περιέχονται στη φόρμα και έπειτα, σχεδιάζει το πλέγμα ζωγραφίζοντας τα κελιά, ανάλογα τον τύπο του καθενός, από τα οποία αποτελείται αυτό. Αφού ολοκληρωθεί ο σχεδιασμός του πλέγματος, σειρά έχει η εμφάνιση της διαδρομής του κάθε οχήματος. Σε αυτό το στάδιο, η διαδρομή έχει ακόμα τη μορφή σημείων καμπής (JumpPoints), που ενώνουν τις νοητές ευθείες που αποτελούν τη βέλτιστη διαδρομή, και όχι διαδοχικών βημάτων.

```
1.  private void main_form_Paint(object sender, PaintEventArgs e) {
2.              paper = e.Graphics;
3.              paper.SmoothingMode = System.Drawing.Drawing2D.SmoothingMode.HighQuality;
4.
5.              try {
6.
7.                  if (importedLayout != null) {
8.
9.                      Rectangle r = new Rectangle (new Point (m_rectangles[0]
    [0].x,m_rectangles[0][0].y)
10.                         , new Size((m_rectangles[Constants.__WidthBlocks-1]
    [Constants.__HeightBlocks-1].x)+Constants.__BlockSide
11.                             , (m_rectangles[Constants.__WidthBlocks - 1]
    [Constants.__HeightBlocks - 1].y) - Constants.__TopBarOffset + Constants.__BlockSide));
12.                     paper.DrawImage(importedLayout, r);
13.
14.                 }
15.                 //draws the grid
16.                 for (int widthTrav = 0; widthTrav < Constants.__WidthBlocks; widthTrav++) {
17.                     for (int heightTrav = 0; heightTrav < Constants.__HeightBlocks; heightTrav
    ++) {
18.                         m_rectangles[widthTrav][heightTrav].DrawBox(paper, BoxType.Normal);
19.                         m_rectangles[widthTrav][heightTrav].DrawBox(paper, BoxType.Start);
20.                         m_rectangles[widthTrav][heightTrav].DrawBox(paper, BoxType.End);
21.                         m_rectangles[widthTrav][heightTrav].DrawBox(paper, BoxType.Wall);
22.                         m_rectangles[widthTrav][heightTrav].DrawBox(paper, BoxType.Load);
23.
24.                         if (m_rectangles[widthTrav][heightTrav].boxType == BoxType.Load
25.                             && isLoad[widthTrav, heightTrav] == 3)
26.                             m_rectangles[widthTrav][heightTrav].SetAsTargetted(paper);
27.                     }
28.                 }
29.
30.                 int c = 0;
31.                 for (int i = 0; i < StartPos.Count; i++)
32.                     c += AGVs[i].JumpPoints.Count;
33.
34.                 for (int i = 0; i < StartPos.Count; i++) {
```

```
                    AGVs[i].StepsCounter = 0;
                    if (!NoJumpPointsFound) {
                        for (int resultTrav = 0; resultTrav < c; resultTrav++) {
                            try {
                                if (linesToolStripMenuItem.Checked)
                                    AGVs[i].Paths[resultTrav].drawLine(paper);
                                if (!isMouseDown)
                                    DrawPoints(AGVs[i].Paths[resultTrav], i);
                            } catch { }
                        }
                    }
                }
                int AGVs_list_index = 0;
                if (aGVIndexToolStripMenuItem.Checked) {
                    for (int i = 0; i < nUD_AGVs.Value; i++) {
                        if (!TrappedStatus[i]) {
                            paper.DrawString("AGV:" + AGVs[AGVs_list_index].ID,
                                             new Font("Tahoma", 8, FontStyle.Bold),
                                             new SolidBrush(Color.Red),
                                             new Point((StartPos[AGVs_list_index].x * Constant
s.__BlockSide) - 10, ((StartPos[AGVs_list_index].y * Constants.__BlockSide) + Constants.__TopB
arOffset) - Constants.__BlockSide));
                            AGVs_list_index++;

                        }
                    }
                }


            } catch { }


        }
```

7.1.10. DrawPoints

Η συνάρτηση αυτή δέχεται ως ορίσματα 2 παραμέτρους, υπολογίζει και έπειτα αποθηκεύει στον ενσωματωμένο πίνακα βημάτων, του κάθε οχήματος, όλα τα επιμέρους σημεία τα οποία αποτελούν τη διαδρομή που προδιαγράφεται από την παράμετρο “x”. Το όρισμα αυτό έχει τέσσερις ιδιότητες οι οποίες είναι ένα ζεύγος τιμών που δηλώνουν συντεταγμένες αρχικού σημείου και ένα δεύτερο ζεύγος συντεταγμένων τελικού σημείου. Στη σειρά οχτώ (8) καλείται η συνάρτηση “GetLength” που αναπτύχθηκε για τον υπολογισμό της Ευκλείδειας απόστασης ανάμεσα στα 2 παραπάνω ζεύγη συντεταγμένων και λίγο παρακάτω (σειρά δέκα), καλώντας την “getSide” γίνεται υπολογισμός της υποτείνουσας του κάθε κελιού. Έπειτα, αξιοποιώντας τα 2 παραπάνω δεδομένα, υπολογίζουμε όλα τα ενδιάμεσα σημεία και τα εισάγουμε στον αντίστοιχο πίνακα βημάτων της κλάσης Vehicle.

```
1.  private void DrawPoints(GridLine x, int agv_index) {
2.              Point[] currentLinePoints;//1d array of points.used to track all the points of cur
    rent line
3.
4.              int x1 = x.fromX;
5.              int y1 = x.fromY;
6.              int x2 = x.toX;
7.              int y2 = x.toY;
8.              double distance = __f.GetLength(x1, y1, x2, y2);
9.
10.             double side = __f.getSide(m_rectangles[0][0].height
11.                             , m_rectangles[0][0].height);
12.
13.             int distanceBlocks = -1; //the quantity of blocks,matching the current line's leng
    th
14.
15.             if ((x1 < x2) && (y1 < y2)) //diagonal-right bottom direction
16.                 distanceBlocks = Convert.ToInt32(distance / side);
17.             else if ((x1 < x2) && (y1 > y2)) //diagonal-right top direction
18.                 distanceBlocks = Convert.ToInt32(distance / side);
19.             else if ((x1 > x2) && (y1 < y2)) //diagonal-left bottom direction
20.                 distanceBlocks = Convert.ToInt32(distance / side);
21.             else if ((x1 > x2) && (y1 > y2)) //diagonal-left top direction
22.                 distanceBlocks = Convert.ToInt32(distance / side);
23.             else if ((y1 == y2) || (x1 == x2)) //horizontal or vertical
24.             {
25.                 distanceBlocks = Convert.ToInt32(distance / m_rectangles[0][0].width);
26.                 //we do not need hypotenuse here
27.             } else
28.                 MessageBox.Show(this, "Unexpected error", "", MessageBoxButtons.OK, MessageBox
    Icon.Error);
29.
30.             currentLinePoints = new Point[distanceBlocks];
31.             double t;
32.
33.             for (int i = 0; i < distanceBlocks; i++) {
34.                 calibrated = false;
35.                 if (distance != 0)
36.                     t = ((side) / distance);
```

```
37.                 else
38.                     return;
39.
40.                 a = Convert.ToInt32(((1 - t) * x1) + (t * x2));
41.                 b = Convert.ToInt32(((1 - t) * y1) + (t * y2));
42.                 Point _p = new Point(a, b);
43.
44.                 for (int k = 0; k < Constants.__WidthBlocks; k++) {
45.
46.                     for (int l = 0; l < Constants.__HeightBlocks; l++) {
47.
48.                         if (m_rectangles[k][l].boxRec.Contains(_p)) {
49.
50.
51.                             //+9 is the width/2 - handling boxes from their centre
52.                             int sideX = m_rectangles[k][l].boxRec.X + ((Constants.__BlockSide
    / 2) - 1);
53.                             int sideY = m_rectangles[k][l].boxRec.Y + ((Constants.__BlockSide
    / 2) - 1);
54.                             currentLinePoints[i].X = sideX;
55.                             currentLinePoints[i].Y = sideY;
56.
57.                             if (dotsToolStripMenuItem.Checked) {
58.                                 using (SolidBrush br = new SolidBrush(Color.BlueViolet))
59.                                     paper.FillEllipse(br, currentLinePoints[i].X - 3,
60.                                         currentLinePoints[i].Y - 3,
61.                                         5, 5);
62.                             }
63.
64.                             using (Font stepFont = new Font("Tahoma", 8, FontStyle.Bold))//
    Font used for numbering the steps/current block)
65.                             {
66.                                 using (SolidBrush fontBR = new SolidBrush(Color.FromArgb(53, 1
    53, 153)))
67.                                     if (stepsToolStripMenuItem.Checked)
68.                                         paper.DrawString(AGVs[agv_index].StepsCounter + ""
69.                                         , stepFont
70.                                         , fontBR
71.                                         , currentLinePoints[i]);
72.                             }
73.                             calibrated = true;
74.                         }
75.                     }
76.                 }
77.
78.                 if (calibrated) {
79.                     AGVs[agv_index].Steps[AGVs[agv_index].StepsCounter].X = currentLinePoints[
    i].X;
80.                     AGVs[agv_index].Steps[AGVs[agv_index].StepsCounter].Y = currentLinePoints[
    i].Y;
81.                     AGVs[agv_index].StepsCounter++;
82.                 }
83.                 //init next steps
84.                 x1 = currentLinePoints[i].X;
85.                 y1 = currentLinePoints[i].Y;
86.                 distance = __f.GetLength(x1, y1, x2, y2);
87.             }
88.         }
```

7.1.11. timers

Η συνάρτηση αυτή καλείται όταν ο χρήστης επιλέξει να ξεκινήσει η προσομοίωση. Η διαθεσιμότητα των οχημάτων είναι πλέον γνωστή με τους ελέγχους που έχουν προηγηθεί και έχει δηλωθεί στον Boolean πίνακα “TrappedStatus[ ]”. Το πλήθος των κελιών με περιεχόμενο “true” καθορίζει και το πλήθος των χρονιστών (Timers) που θα ενεργοποιηθούν, καθένας από του οποίους χειρίζεται και το αντίστοιχο όχημα.

```
1.  private void timers() {
2.              //every timer is responsible for every agv for up to 5 AGVs
3.
4.              int _c = 0;
5.              for (int i=0; i< TrappedStatus.Length;i++)
6.                  if (!TrappedStatus[i])
7.                      _c++;
8.
9.              switch(_c)
10.             {
11.                 case 1:
12.                     timer0.Start();
13.                     break;
14.                 case 2:
15.                     timer0.Start();
16.                     timer1.Start();
17.                     break;
18.                 case 3:
19.                     timer0.Start();
20.                     timer1.Start();
21.                     timer2.Start();
22.                     break;
23.                 case 4:
24.                     timer0.Start();
25.                     timer1.Start();
26.                     timer2.Start();
27.                     timer3.Start();
28.                     break;
29.                 case 5:
30.                     timer0.Start();
31.                     timer1.Start();
32.                     timer2.Start();
33.                     timer3.Start();
34.                     timer4.Start();
35.                     break;
36.             }
37.         }
```

7.1.12. timer0_Tick

Το παρακάτω τμήμα κώδικα αποτελεί event του αντικειμένου timer0, το οποίο αντικείμενο είναι χρονιστής. Το event αυτό έχει περίοδο που μπορεί να μεταβληθεί ανάλογα με τις ανάγκες της εφαρμογής, και εκτελείται κάθε φορά που συμπληρώνεται ο χρόνος της περιόδου αυτής. Η εφαρμογή περιέχει πέντε χρονιστές, καθένας από τους οποίους είναι υπεύθυνος για το χειρισμό ενός οχήματος. Ο κώδικας που περιέχεται στα events των υπόλοιπων χρονιστών είναι πανομοιότυπος με το τμήμα κώδικα που φαίνεται εδώ, με μόνη διαφορά την τιμή των δεικτών (indexes) να αλλάζει ανάλογα με το χρονιστή.

```
1.  private void timer0_Tick(object sender, EventArgs e) {
2.              int mysteps = 0;
3.              for (int i = 0; i < Constants.__MaximumSteps; i++) {
4.                  if (AGVs[0].Steps[i].X == 0 || AGVs[0].Steps[i].Y == 0)
5.                      i = Constants.__MaximumSteps;
6.                  else
7.                      mysteps++;
8.              }
9.              AGVs[0].StepsCounter = mysteps;
10.
11.             animator(timer_counter[0], 0);
12.
13.             timer_counter[0]++;
14.         }
```

7.1.13. animator

Όπως είναι φανερό, η συνάρτηση αυτή καλείται μέσα από τον κάθε χρονιστή, δίνοντας ως παραμέτρους το δείκτη του χρονιστή, που καλεί τη συνάρτηση, καθώς επίσης και το βήμα στο οποίο βρίσκεται το εκάστοτε όχημα. Πραγματοποιείται υπολογισμός της απόστασης, σε βήματα, από το φορτίο που κάθε όχημα έχει ως αποστολή να παραλάβει και εμφανίζονται οι σχετικές πληροφορίες στο παράθυρο παρακολούθησης στην κύρια φόρμα. Έπειτα, ελέγχεται εάν το όχημα, που τη δεδομένη στιγμή χειρίζεται η συνάρτηση, πρόκειται να συμπέσει στο ίδιο κελί του πλέγματος που βρίσκεται κάποιο άλλο, και ανάλογα την περίπτωση παραχωρείται προτεραιότητα ανάμεσα στα οχήματα. Σε κάθε κλήση της συνάρτησης, επίσης, γίνεται έλεγχος ώστε να διαπιστωθεί αν το εκάστοτε όχημα έχει φτάσει στον προορισμό του, ο οποίος μπορεί να είναι είτε το φορτίο που έχει ως στόχο είτε το τελικό σημείο εκφόρτωσης. Στη δεύτερη περίπτωση, μόλις φτάσει στην έξοδο, γίνεται έλεγχος για εναπομείναντα διαθέσιμα φορτία και εφόσον αυτός είναι αληθής, καλείται η συνάρτηση “getNextLoad”. Σε αντίθετη περίπτωση βέβαια, οι χρονιστές που χειρίζονται τα αντίστοιχα οχήματα, τερματίζονται και ολοκληρώνεται η διαδικασία.

```
1.  private void animator(int steps_counter, int agv_index) {
2.
3.              int stepx = Convert.ToInt32(AGVs[agv_index].Steps[steps_counter].X);
4.              int stepy = Convert.ToInt32(AGVs[agv_index].Steps[steps_counter].Y);
5.
6.              if (stepx == 0 || stepx == 0)
7.                  return;
8.
9.              bool isfreeload = false;
10.             bool halted = false;
11.             displayStepsToLoad(steps_counter, agv_index);
12.
13.             update_emissions(agv_index);
14.
15.             //RULES OF WHICH AGV WILL STOP WILL BE ADDED
16.
17.             for (int i = 0; i < StartPos.Count; i++) {
18.                 if (agv_index != i
19.                     && AGVs[i].GetLocation() != new Point(0, 0)
20.                     && AGVs[agv_index].GetLocation() == AGVs[i].GetLocation()
21.                     && AGVs[agv_index].GetLocation() != endPointCoords
22.                     ) {
23.                     halt(agv_index, steps_counter);
24.                     halted = true;
25.                 } else {
26.                     if (!halted)
27.                         AGVs[agv_index].SetLocation(stepx - ((Constants.__BlockSide / 2) - 1),
    stepy - ((Constants.__BlockSide / 2) - 1));
28.                 }
29.             }
30.
31.
```

```
32.             if (AGVs[agv_index].MarkedLoad.X * Constants.__BlockSide == AGVs[agv_index].GetLoc
    ation().X &&
33.                 (AGVs[agv_index].MarkedLoad.Y * Constants.__BlockSide) + Constants.__TopBarOff
    set == AGVs[agv_index].GetLocation().Y &&
34.                 !AGVs[agv_index].Status.Busy) {
35.
36.                 //marks the pickedup load as walkable AGAIN (since it is now a normal gridbox)

37.                 m_rectangles[AGVs[agv_index].MarkedLoad.X]
    [AGVs[agv_index].MarkedLoad.Y].SwitchLoad();
38.                 searchGrid.SetWalkableAt(AGVs[agv_index].MarkedLoad.X, AGVs[agv_index].MarkedL
    oad.Y, true);
39.
40.                 AGVs[agv_index].Status.Busy = true;
41.                 AGVs[agv_index].setLoaded();
42.                 this.Refresh();
43.
44.                 if (fromstart[agv_index]) {
45.                     loads--;
46.                     isLoad[AGVs[agv_index].MarkedLoad.X, AGVs[agv_index].MarkedLoad.Y] = 2;
47.
48.                     fromstart[agv_index] = false;
49.                 }
50.             }
51.
52.             if (!fromstart[agv_index]) {
53.                 if (AGVs[agv_index].GetLocation().X == m_rectangles[endPointCoords.X / Constan
    ts.__BlockSide][(endPointCoords.Y - Constants.__TopBarOffset) / Constants.__BlockSide].x &&
54.                     AGVs[agv_index].GetLocation().Y == m_rectangles[endPointCoords.X / Constan
    ts.__BlockSide][(endPointCoords.Y - Constants.__TopBarOffset) / Constants.__BlockSide].y) {
55.
56.
57.                     AGVs[agv_index].Status.Busy = false;
58.
59.                     for (int k = 0; k < Constants.__WidthBlocks; k++) {
60.                         for (int b = 0; b < Constants.__HeightBlocks; b++) {
61.                             if (isLoad[k, b] == 1 || isLoad[k,b]==4)
62.                             {
63.                                 isfreeload = true;
64.                                 k = Constants.__WidthBlocks;
65.                                 b = Constants.__HeightBlocks;
66.                             }
67.                         }
68.                     }
69.
70.                     if (loads > 0 && isfreeload) {
71.
72.                         Reset(agv_index);
73.                         AGVs[agv_index].Status.Busy = true;
74.                         AGVs[agv_index].MarkedLoad = new Point();
75.                         getNextLoad(agv_index);
76.
77.
78.                         AGVs[agv_index].Status.Busy = false;
79.                         AGVs[agv_index].setEmpty();
80.
81.                     } else {
82.                         AGVs[agv_index].setEmpty();
83.                         isfreeload = false;
84.
85.                         switch (agv_index) {
```

```
                                case 0:
                                    timer0.Stop();
                                    agv1steps_LB.Text = "AGV 1: Finished";
                                    break;
                                case 1:
                                    timer1.Stop();
                                    agv2steps_LB.Text = "AGV 2: Finished";
                                    break;
                                case 2:
                                    timer2.Stop();
                                    agv3steps_LB.Text = "AGV 3: Finished";
                                    break;
                                case 3:
                                    timer3.Stop();
                                            agv4steps_LB.Text = "AGV 4: Finished";
                                            break;
                                        case 4:
                                            timer4.Stop();
                                            agv5steps_LB.Text = "AGV 5: Finished";
                                            break;
                                    }

                                }

                                timer_counter[agv_index] = -1;
                                steps_counter = 0;

                            }
                        } else {
                            if (isLoad[AGVs[agv_index].MarkedLoad.X, AGVs[agv_index].MarkedLoad.Y]
    == 2)
                                if (AGVs[agv_index].GetLocation().X == m_rectangles[endPointCoords.
    X / Constants.__BlockSide][(endPointCoords.Y - Constants.__TopBarOffset) / Constants.__BlockSi
    de].x &&
                                    AGVs[agv_index].GetLocation().Y == m_rectangles[endPointCoords.
    X / Constants.__BlockSide][(endPointCoords.Y - Constants.__TopBarOffset) / Constants.__BlockSi
    de].y)
                                    switch (agv_index) {
                                        case 0:
                                            timer0.Stop();
                                            agv1steps_LB.Text = "AGV 0: Finished";
                                            break;
                                        case 1:
                                            timer1.Stop();
                                            agv2steps_LB.Text = "AGV 1: Finished";
                                            break;
                                        case 2:
                                            timer2.Stop();
                                            agv3steps_LB.Text = "AGV 2: Finished";
                                            break;
                                        case 3:
                                            timer3.Stop();
                                            agv4steps_LB.Text = "AGV 3: Finished";
                                            break;
                                        case 4:
                                            timer4.Stop();
                                            agv5steps_LB.Text = "AGV 4: Finished";
                                            break;
                                    }
                        }

```

```
142.
143.                     if (!(timer0.Enabled || timer1.Enabled || timer2.Enabled || timer3.Enabled
    || timer4.Enabled)) //if at least 1 timer is active, do not let the user access the Checkboxes
    etc. etc
144.                     {
145.                         gb_settings.Enabled = true;
146.                         settings_menu.Enabled = true;
147.                     }
148.
149.                     if (!timer0.Enabled && !timer1.Enabled && !timer2.Enabled && !
    timer3.Enabled && !timer4.Enabled)//when all agvs has finished their tasks
150.                     {
151.                         //clear all the paths
152.                         for (int i = 0; i < StartPos.Count(); i++) {
153.                             AGVs[i].JumpPoints = new List<GridPos>();
154.                         }
155.                         allowHighlight = false;
156.                         highlightOverCurrentBoxToolStripMenuItem.Checked = allowHighlight;
157.                         triggerStartMenu(false);
158.                         this.Refresh();
159.                         this.Invalidate();
160.                     }
161.
162.                 }
```

7.1.14. getNextLoad

Η κλήση της getNextLoad συνεπάγεται πως το όχημα έχει ολοκληρώσει την πρώτη του αποστολή και ετοιμάζεται να ξεκινήσει την επόμενη. Αυτό σημαίνει πως το σημείο που αρχικά θεωρείτο ως τελικός στόχος, τώρα θα θεωρείται και σημείο αφετηρίας διότι το όχημα ξεκινάει από το σημείο εκφόρτωσης με στόχο το επόμενο φορτίο που αιτείται παραλαβής και έπειτα οφείλει να επιστρέψει ξανά στην έξοδο. Συνεπώς, η λειτουργία της συνάρτησης είναι να δημιουργήσει δύο διαδρομές και στο τέλος να τις ενώσει. Αρχικά λοιπόν, καλείται η “checkForTrappedLoads” για να ολοκληρώσει τον έλεγχό της ούτως ώστε να αποφασιστεί ποιό φορτίο θα καλέσει ποιό όχημα. Μόλις γίνει αυτό, υπολογίζεται η βέλτιστη διαδρομή προς το φορτίο και έπειτα ακολουθεί η εύρεση της διαδρομής για την επιστροφή του οχήματος, μαζί με το φορτίο, προς την έξοδο.

```
1.  private void getNextLoad(int whichAGV) {
2.
3.              aGVIndexToolStripMenuItem.Checked = false;
4.              GridPos endPos = new GridPos();
5.
6.              for (int widthTrav = 0; widthTrav < Constants.__WidthBlocks; widthTrav++)
7.                  for (int heightTrav = 0; heightTrav < Constants.__HeightBlocks; heightTrav++)
8.                      if (m_rectangles[widthTrav][heightTrav].boxType == BoxType.End)
9.                          try {
10.                             StartPos[whichAGV] = new GridPos(widthTrav, heightTrav);
11.                             a = StartPos[whichAGV].x;
12.                             b = StartPos[whichAGV].y;
13.                         }
14.                         catch {  }
15.
16.             List<GridPos> loadPos = new List<GridPos>();
17.             int loadPos_index = 0;
18.
19.             for (int i = 0; i < Constants.__WidthBlocks; i++)
20.                 for (int j = 0; j < Constants.__HeightBlocks; j++)
21.                 {
22.                     if (m_rectangles[i][j].boxType == BoxType.Load)
23.                         searchGrid.SetWalkableAt(new GridPos(i, j), false);
24.
25.                     if (isLoad[i, j] == 1 || isLoad[i, j] == 4)
26.                     {
27.                         loadPos.Add(new GridPos());
28.                         loadPos[loadPos_index].x = i;
29.                         loadPos[loadPos_index].y = j;
30.                         loadPos_index++;
31.                     }
32.                 }
33.             checkForTrappedLoads(loadPos);
34.
35.             for (int widthTrav = 0; widthTrav < Constants.__WidthBlocks; widthTrav++)
36.                 for (int heightTrav = 0; heightTrav < Constants.__HeightBlocks; heightTrav++)
37.                     if (m_rectangles[widthTrav][heightTrav].boxType == BoxType.Load && isLoad[w
    idthTrav, heightTrav] == 1) {
38.                         isLoad[widthTrav, heightTrav] = 3;
39.                         AGVs[whichAGV].MarkedLoad = new Point(widthTrav, heightTrav);
```

```
40.
41.                         endPos.x = widthTrav;
42.                         endPos.y = heightTrav;
43.                         loads--;
44.
45.                         //Mark all loads as unwalkable,except the targetted ones
46.                         for (int k = 0; k < Constants.__WidthBlocks; k++)
47.                           for (int l = 0; l < Constants.__HeightBlocks; l++)
48.                            if (m_rectangles[k][l].boxType == BoxType.Load && isLoad[k, l] != 3
49.                                     searchGrid.SetWalkableAt(new GridPos(k, l), false);
50.
51.                         widthTrav = Constants.__WidthBlocks;
52.                         heightTrav = Constants.__HeightBlocks;
53.                     }
54.
55.             jumpParam.Reset(StartPos[whichAGV], endPos);
56.             JumpPointsList = JumpPointFinder.FindPath(jumpParam, paper);
57.
58.             //Mark all loads as unwalkable
59.             for (int k = 0; k < Constants.__WidthBlocks; k++)
60.                 for (int l = 0; l < Constants.__HeightBlocks; l++)
61.                     if (m_rectangles[k][l].boxType == BoxType.Load)
62.                         searchGrid.SetWalkableAt(new GridPos(k, l), false);
63.
64.
65.             for (int j = 0; j < JumpPointsList.Count; j++)
66.                 AGVs[whichAGV].JumpPoints.Add(JumpPointsList[j]);
67.
68.             int c = 0;
69.             for (int i = 0; i < StartPos.Count; i++)
70.                 c += AGVs[i].JumpPoints.Count;
71.
72.             for (int i = 0; i < StartPos.Count; i++)
73.                 if ((c - 1) > 0)
74.                     Array.Resize(ref AGVs[i].Paths, c - 1);
75.
76.
77.             for (int j = 0; j < AGVs[whichAGV].JumpPoints.Count - 1; j++) {
78.                 GridLine line = new GridLine(
79.                     m_rectangles
80.                         [AGVs[whichAGV].JumpPoints[j].x]
81.                         [AGVs[whichAGV].JumpPoints[j].y],
82.                    m_rectangles
83.                         [AGVs[whichAGV].JumpPoints[j + 1].x]
84.                         [AGVs[whichAGV].JumpPoints[j + 1].y]
85.                                         );
86.
87.                 AGVs[whichAGV].Paths[j] = line;
88.             }
89.
90.
91.             //2nd part of route: Go to exit
92.             int old_c = c-1;
93.
94.             jumpParam.Reset(endPos, StartPos[whichAGV]);
95.             JumpPointsList = JumpPointFinder.FindPath(jumpParam, paper);
96.             for (int j = 0; j < JumpPointsList.Count; j++)
97.                 AGVs[whichAGV].JumpPoints.Add(JumpPointsList[j]);
98.
99.             c = 0;
100.             for (int i = 0; i < StartPos.Count; i++)
```

```
101.                c += AGVs[i].JumpPoints.Count;
102.
103.            for (int i = 0; i < StartPos.Count; i++)
104.                if ((c - 1) > 0)
105.                    Array.Resize(ref AGVs[i].Paths, old_c+(c - 1));
106.
107.
108.            for (int i = 0; i < StartPos.Count; i++)
109.                for (int j = 0; j < AGVs[i].JumpPoints.Count - 1; j++) {
110.                    GridLine line = new GridLine(
111.                        m_rectangles
112.                            [AGVs[i].JumpPoints[j].x]
113.                            [AGVs[i].JumpPoints[j].y],
114.                        m_rectangles
115.                            [AGVs[i].JumpPoints[j + 1].x]
116.                            [AGVs[i].JumpPoints[j + 1].y]
117.                                                );
118.
119.                    AGVs[i].Paths[j] = line;
120.                }
121.
122.            this.Invalidate();
123.        }
```

7.2. Κλάση οχήματος – Vehicle.cs

```
1.  namespace kagv {
2.
3.      class Vehicle {
4.
5.          //***************************************
6.          //AGV Status
7.          internal class AGVStatus {
8.              public bool Busy { get; set; }
9.              public bool Loaded { get; set; }
10.         }
11.
12.         private AGVStatus status = new AGVStatus();
13.         public AGVStatus Status {
14.             get { return this.status; }
15.         }
16.         //=======================================
17.
18.         //***************************************
19.         //AGV Steps
20.         internal class AGVSteps {
21.             public double X { get; set; }
22.             public double Y { get; set; }
23.         }
24.
25.         private AGVSteps[] steps;
26.         public AGVSteps[] Steps {
27.             get { return this.steps; }
28.         }
29.         //=======================================
30.         //AGV Path
31.         public GridLine[] Paths = new GridLine[Constants.__MaximumSteps];
32.         public Point Location;
33.         public Point MarkedLoad;
34.
35.         //get-set is not a mandatory here
36.         public int StartX;
37.         public int StartY;
38.         public int SizeX;
39.         public int SizeY;
40.         public int ID = -1;
41.
42.         private Panel AgvPortrait;
43.         private PictureBox AgvIcon;
44.         private Point AgvLocation;
45.         private Form mirroredForm;
46.
47.
48.         //***************************************
49.         //AGV JumpPoints
50.         private List<GridPos> jmp_pnts = new List<GridPos>();
51.         public List<GridPos> JumpPoints {
52.             get {
53.                 return this.jmp_pnts;
54.             }
55.             set {
56.                 this.jmp_pnts = value;
57.             }
```

```
58.         }
59.         //=======================================
60.
61.         //***************************************
62.         //AGV StepsCounter
63.         private int steps_counter;
64.         public int StepsCounter {
65.             get {
66.                 return this.steps_counter;
67.             }
68.             set {
69.                 this.steps_counter = value;
70.             }
71.         }
72.         //=======================================
73.
74.
75.        //constructor
76.         public Vehicle(Form handle) {
77.             mirroredForm = handle;
78.             this.status.Busy = false;
79.             this.status.Loaded = false;
80.             this.steps = new AGVSteps[Constants.__MaximumSteps];
81.             for (int i = 0; i < steps.Length; i++) {
82.                 steps[i] = new AGVSteps();
83.                 steps[i].X = -1;
84.                 steps[i].Y = -1;
85.             }
86.         }
87.
88.         public void Init() {
89.
90.             //init vars
91.             this.status.Busy = false;
92.             this.status.Loaded = false;
93.
94.             AgvPortrait = new Panel();
95.             AgvPortrait.Name = "AGVPORTRAIT";
96.             AgvIcon = new PictureBox();
97.
98.             AgvPortrait.Controls.Add(AgvIcon);
99.
100.                Size _size = new Size(Constants.__BlockSide - 2, Constants.__BlockSide - 2)
    ;
101.                Point _location = new Point(StartX, StartY);
102.                AgvPortrait.Size = _size;
103.                AgvPortrait.Location = _location;
104.                AgvPortrait.Visible = true;
105.                AgvPortrait.BringToFront();
106.                AgvPortrait.BackColor = Color.Transparent;
107.
108.                mirroredForm.Controls.Add(AgvPortrait);
109.
110.                AgvIcon.BackColor = mirroredForm.BackColor;
111.                AgvIcon.BorderStyle = BorderStyle.None;
112.                AgvIcon.SizeMode = PictureBoxSizeMode.StretchImage;
113.                AgvIcon.Size = _size;
114.                AgvIcon.Visible = true;
115.
116.                AgvIcon.Image = _getEmbedResource("empty.png");
117.
```

```
118.                     AgvIcon.BackColor = Color.Transparent;
119.
120.                     //public exports
121.                     Location = AgvPortrait.Location;
122.
123.                 }
124.
125.
126.
127.                 private Image _getEmbedResource(string a) {
128.                     System.Reflection.Assembly _assembly;
129.                     Stream _myStream;
130.                     _assembly = System.Reflection.Assembly.GetExecutingAssembly();
131.
132.                     _myStream = _assembly.GetManifestResourceStream("kagv.Resources." + a);
133.                     Image _b = Image.FromStream(_myStream);
134.                     return _b;
135.
136.                 }
137.
138.                 public void killIcon() {
139.                     try {
140.                         this.AgvIcon.Dispose();
141.                         this.AgvPortrait.Dispose();
142.                     } catch { }
143.                 }
144.
145.
146.                 public void setLoaded() {
147.                     this.AgvIcon.Image = _getEmbedResource("loaded.png");
148.                     this.status.Loaded = true;
149.                 }
150.
151.                 public void setEmpty() {
152.                     this.AgvIcon.Image = _getEmbedResource("empty.png");
153.                     this.status.Loaded = false;
154.                 }
155.
156.                 public void updateAGV() {
157.                     if (mirroredForm.Controls.Count != 0) {
158.                         foreach (Control p in mirroredForm.Controls) {
159.                             if (p == AgvIcon)
160.                                 mirroredForm.Controls.Remove(p);
161.                             if (p.Name == "AGVPORTRAIT")
162.                                 mirroredForm.Controls.Remove(p);
163.                         }
164.
165.                     } else
166.                         return;
167.                 }
168.
169.
170.                 public void SetLocation(int X, int Y) {
171.                     AgvLocation = new Point(X, Y);
172.                     AgvPortrait.Location = AgvLocation;
173.                     Location = AgvLocation;
174.                 }
175.                 public void SetLocation(Point loc) {
176.                     AgvLocation = loc;
177.                     AgvPortrait.Location = AgvLocation;
178.                     Location = AgvLocation;
```

```
179.                }
180.
181.                public Point GetLocation() {
182.                    return new Point(AgvLocation.X, AgvLocation.Y);
183.                }
184.
185.            }
186.
187.        }
```