Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής
Πρόγραμμα Μεταπτυχιακών Σπουδών
«Πληροφορική»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Χωρική Μνήμη και GOAP : Μια ρεαλιστική προσέγγιση στην πλοήγηση πρακτόρων** **Spatial Memory and GOAP : A Realistic Approach to Agent Navigation**
Ονοματεπώνυμο Φοιτητή	**Ανδρέας Ξερουδάκης**
Πατρώνυμο	**Γεώργιος**
Αριθμός Μητρώου	**ΜΠΠΛ/18052**
Επιβλέπων	**Θεμιστοκλής Παναγιωτόπουλος, Καθηγητής**

Ημερομηνία Παράδοσης **Ιούνιος 2021**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Θεμιστοκλής Παναγιωτόπουλος, Καθηγητής	Άγγελος Πικράκης, Επίκουρος Καθηγητή	Δημήτριος Αποστόλου, Καθηγητής

SPATIAL MEMORY AND GOAP

A REALISTIC APPROACH TO AGENT NAVIGATION

ΧΩΡΙΚΗ ΜΝΗΜΗ ΚΑΙ GOAP

ΜΙΑ ΡΕΑΛΙΣΤΙΚΗ ΠΡΟΣΕΓΓΙΣΗ ΣΤΗΝ ΠΛΟΗΓΗΣΗ ΠΡΑΚΤΟΡΩΝ

ΕΥΡΕΤΗΡΙΟ

ABSTRACT

Usually, when we apply shortest path algorithms, we simply want to find the shortest path from the position of an agent to the position of a target. But this is not exactly the case in the real world. What if the person has never visited the area before? What if the available paths are obscured by obstacles like buildings or trees?

Therefore, when a person is trying to reach a target by passing through an area they have never visited before, they cannot know in advance the shortest path to follow. In order to successfully navigate the area, they will most likely start moving in a straight line heading in the general direction of his target until they find an obstacle and then will try to avoid it by looking for a new path around it. The more they explore the area the better they learn to navigate it and eventually will be able to come to a conclusion about which is the shortest path and whether or not they can even reach their target. This is the behavior I tried to simulate with the application I created.

The agent's primary goal is to reach his girlfriend who is waiting for him to begin their date, and he is already late! To reach her he has to cross a maze, which, in addition to being complicated with many possible routes, it even has colored doors that need keys of the same color to open. If the agent has encountered a door while exploring and has exhausted all possible routes, then he will try to open one of the doors by first searching for the right key. So, the agent will have to explore the area and make a plan to finally reach his goal.

ΠΕΡΙΛΗΨΗ

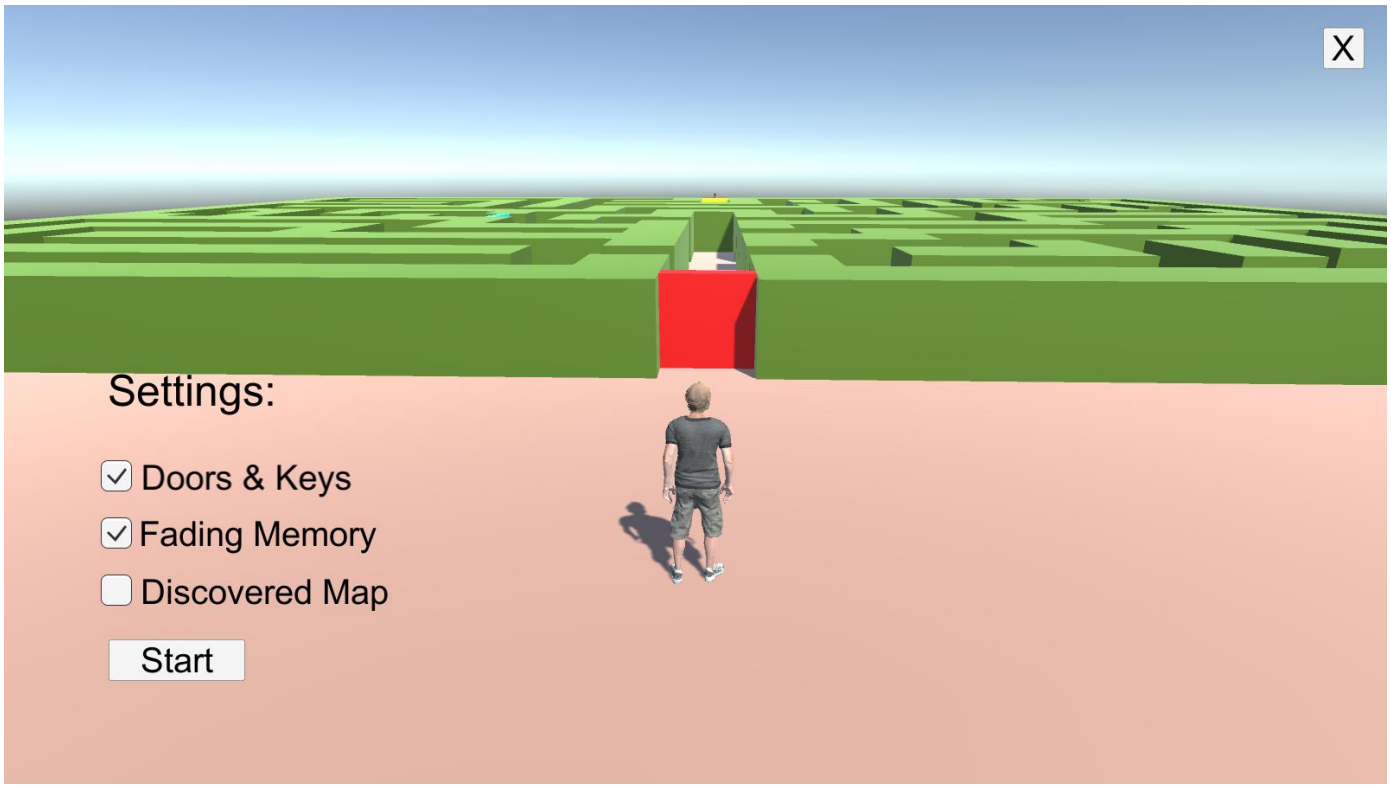
Συνήθως, όταν εφαρμόζουμε αλγόριθμους συντομότερης διαδρομής, θέλουμε απλά να βρούμε τη συντομότερη διαδρομή από τη θέση ενός πράκτορα στη θέση ενός στόχου. Αλλά αυτό δεν συμβαίνει ακριβώς έτσι στην πραγματικότητα. Τι γίνεται αν το άτομο δεν έχει επισκεφθεί ποτέ την περιοχή στο παρελθόν; Τι γίνεται αν τα διαθέσιμα μονοπάτια επισκιάζονται από εμπόδια όπως κτίρια ή δέντρα;

Επομένως, όταν ένα άτομο προσπαθεί να φτάσει σε έναν στόχο περνώντας από μια περιοχή που δεν έχει επισκεφθεί ποτέ πριν δεν μπορεί να ξέρει εκ των προτέρων τη συντομότερη διαδρομή που πρέπει να ακολουθήσει. Προκειμένου να πλοηγηθεί με επιτυχία στην περιοχή, πιθανότατα θα αρχίσει να κινείται σε μία ευθεία γραμμή προς τη γενική κατεύθυνση του στόχου του μέχρι να βρει κάποιο εμπόδιο και στη συνέχεια θα προσπαθήσει να το αποφύγει αναζητώντας ένα νέο μονοπάτι γύρω από αυτό. Όσο περισσότερο εξερευνά την περιοχή τόσο καλύτερα μαθαίνει να την διασχίζει και τελικά θα μπορέσει να καταλήξει σε ένα συμπέρασμα σχετικά με το ποιό είναι το πιο σύντομο μονοπάτι και αν μπορεί ή όχι να φτάσει στον στόχο του. Αυτή είναι η συμπεριφορά που προσπάθησα να προσομοιώσω με την εφαρμογή που δημιούργησα.

Ο πρωταρχικός στόχος του πράκτορα είναι να φτάσει στη φίλη του που τον περιμένει για να ξεκινήσουν το ραντεβού τους, και έχει ήδη αργήσει! Για να φτάσει σε αυτή πρέπει να διασχίσει έναν λαβύρινθο, ο οποίος εκτός του ότι είναι περίπλοκος με πολλές πιθανές διαδρομές, έχει ακόμη και πόρτες που χρειάζονται κλειδιά του ίδιου χρώματος για να ανοίξουν. Εάν ο πράκτορας έχει συναντήσει μια πόρτα ενώ εξερευνά και έχει εξαντλήσει όλες τις πιθανές διαδρομές, τότε θα προσπαθήσει να ανοίξει την πόρτα αναζητώντας πρώτα το σωστό κλειδί. Έτσι, ο πράκτορας θα πρέπει να εξερευνήσει την περιοχή και να κάνει ένα σχέδιο για να επιτύχει τελικά τον στόχο του.

Η ΕΦΑΡΜΟΓΗ

Ανοίγοντας την εφαρμογή ο χρήστης μπορεί να θέσει κάποιες παραμέτρους αναζήτησης του πράκτορα. Κάποιες από τις επιλογές είναι ήδη ενεργοποιημένες.

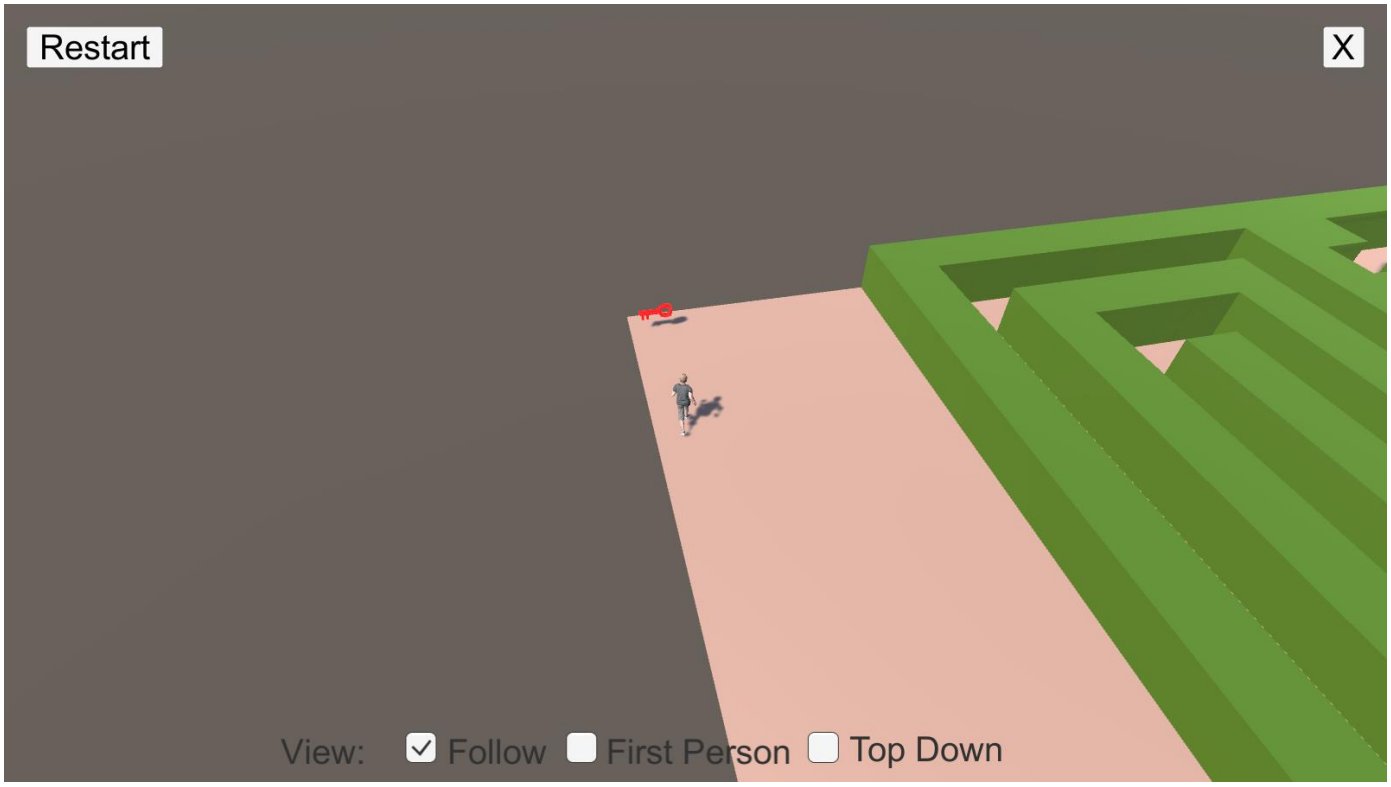


Πατώντας το κουμπί **Start** χωρίς να αλλάξει τις επιλογές ο πράκτορας ξεκινάει να κινείται σε μία ευθεία γραμμή προς το στόχο του. Μόλις ανακαλύψει το πρώτο εμπόδιο αναγκάζεται να εξερευνήσει το χώρο γύρω από αυτό για να ελέγξει αν υπάρχει κάποιο εναλλακτικό πέρασμα.

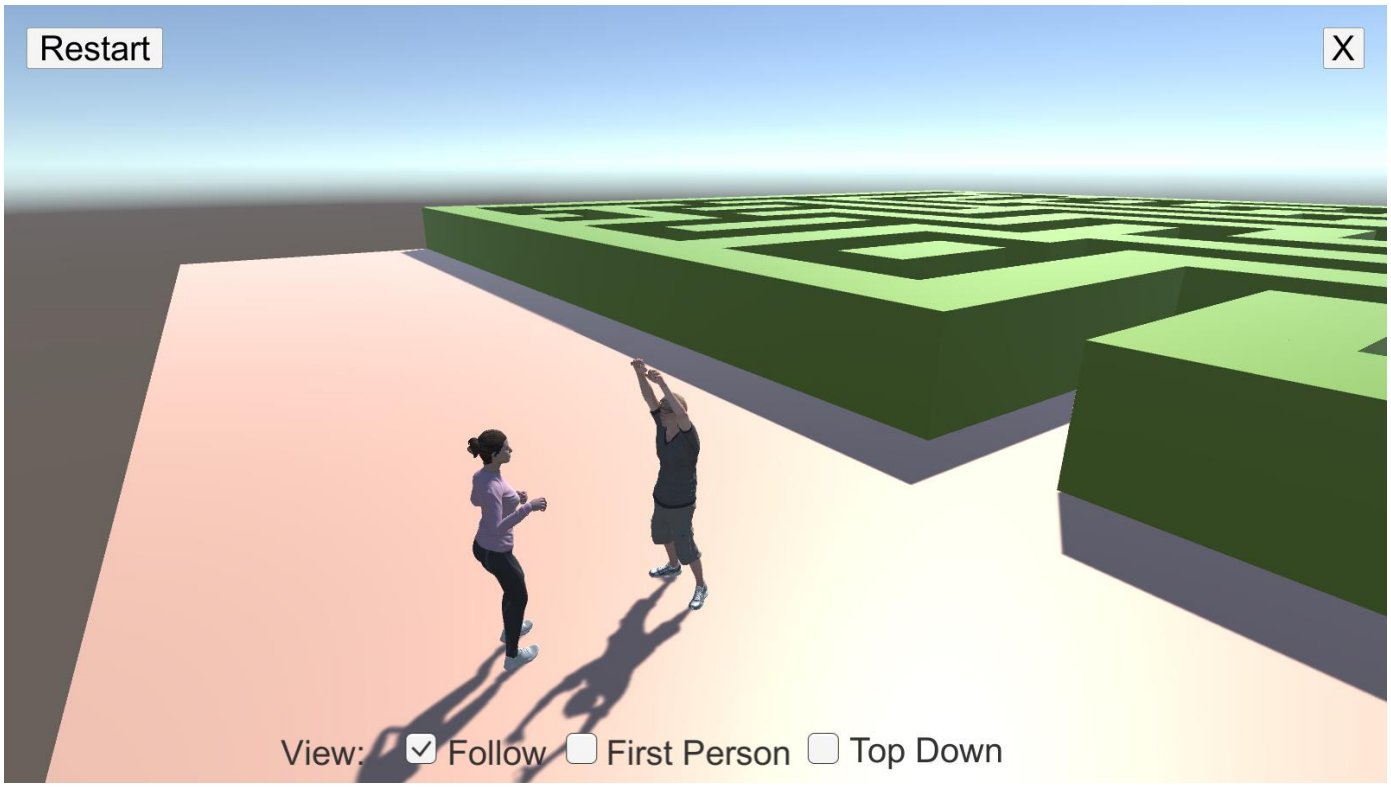
Όταν εξερευνήσει όλο το χώρο στον οποίο έχει πρόσβαση να κινηθεί τότε μπαίνει στη διαδικασία να ανοίξει τυχόν πόρτες που έχει ανακαλύψει. Αν έχει βρει κάποια πόρτα η επόμενη του δράση είναι να βρει ένα κλειδί που έχει αντίστοιχο χρώμα με αυτό της πόρτας με σκοπό να την ανοίξει.

- ➢ Απενεργοποιώντας την επιλογή **Doors & Keys** αφαιρούνται οι πόρτες και τα κλειδιά από το λαβύρινθο και ο πράκτορας αποκτά πρόσβαση σε όλα τα μονοπάτια εξαρχής.
- ➢ Απενεργοποιώντας την επιλογή **Fading Memory** ο πράκτορας δεν ξεχνάει τα σημεία του χάρτη με την πάροδο του χρόνου αλλά μπορεί και θυμάται όποιο σημείο του χάρτη ανακαλύπτει για πάντα.
- ➢ Ενεργοποιώντας την επιλογή **Discovered Map** ο πράκτορας γνωρίζει από πριν όλη τη διάταξη του χάρτη καθώς και τις πόρτες και τα κλειδιά που είναι μέσα σε αυτόν (αν αυτά είναι ενεργοποιημένα). Έτσι μπορεί να βρει κατευθείαν το συντομότερο μονοπάτι και τις δράσεις που απαιτούνται για να φτάσει στο στόχο του.

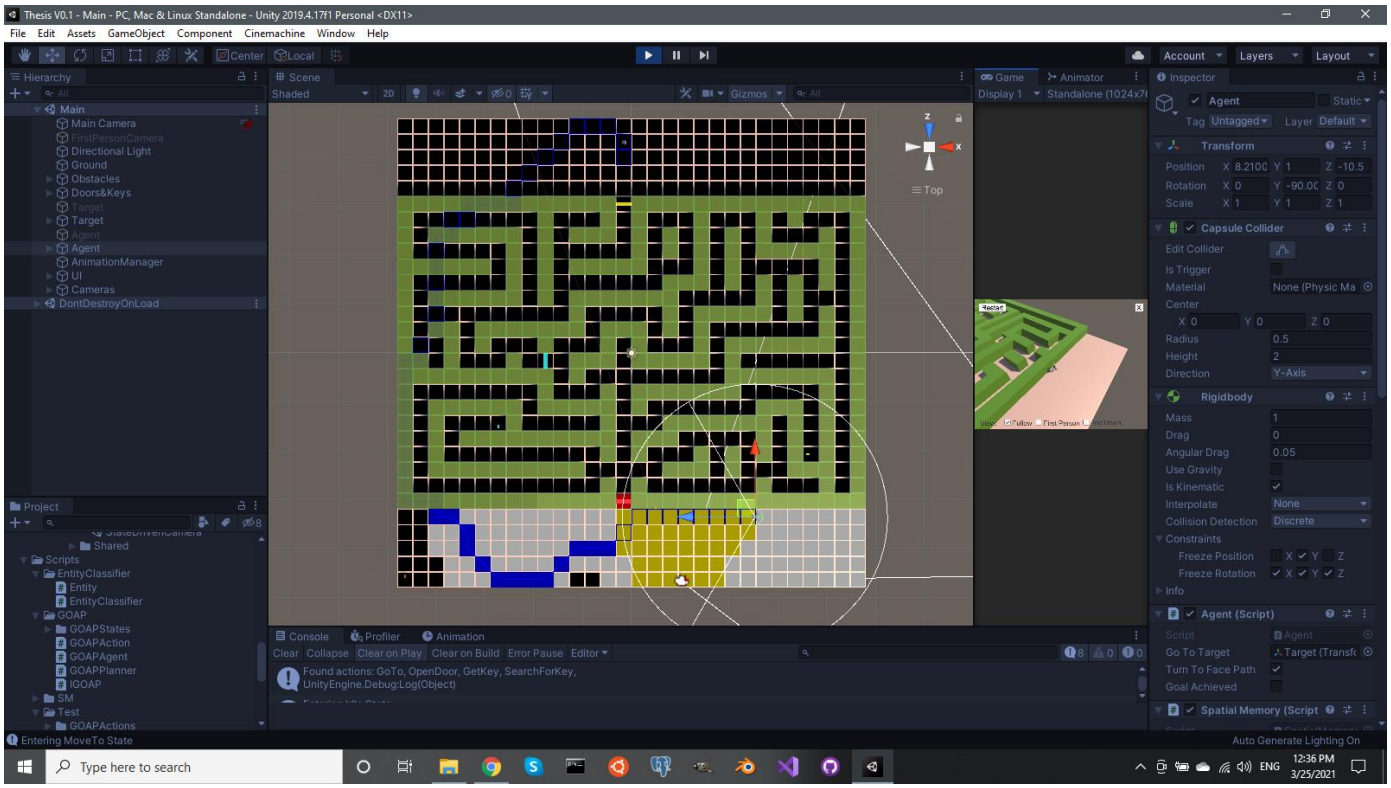
Δοκιμάζοντας διάφορους συνδυασμούς αυτών των τριών επιλογών μπορούμε να παρατηρήσουμε πολύ διαφορετικές συμπεριφορές του πράκτορα.

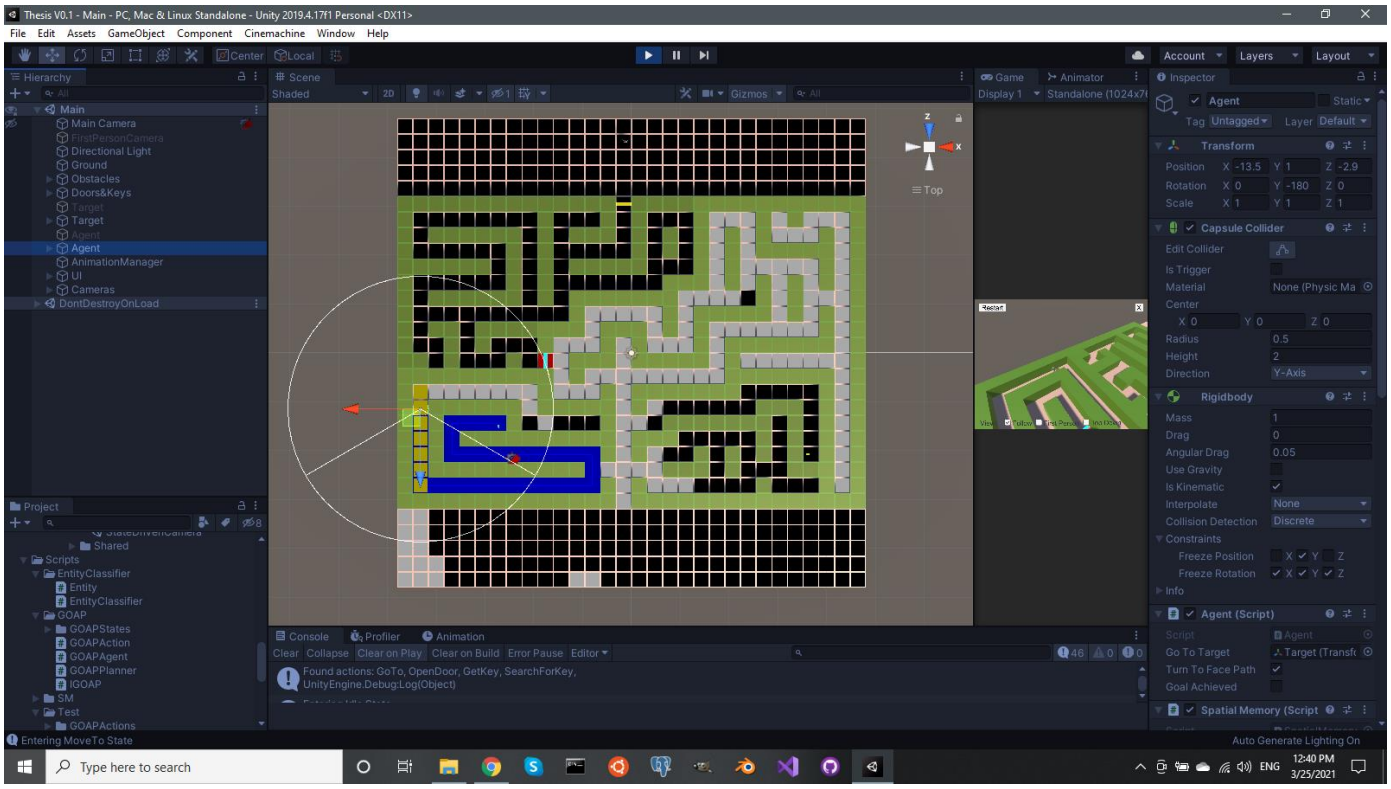


Όταν βρει το κατάλληλο κλειδί τότε το συλλέγει και μπορεί πλέον να ανοίξει την πόρτα. Τότε μπορεί να συνεχίσει να εξερευνά τον υπόλοιπο χώρο επαναλαμβάνοντας αυτή τη διαδικασία μέχρι να φτάσει τελικά στο στόχο του.

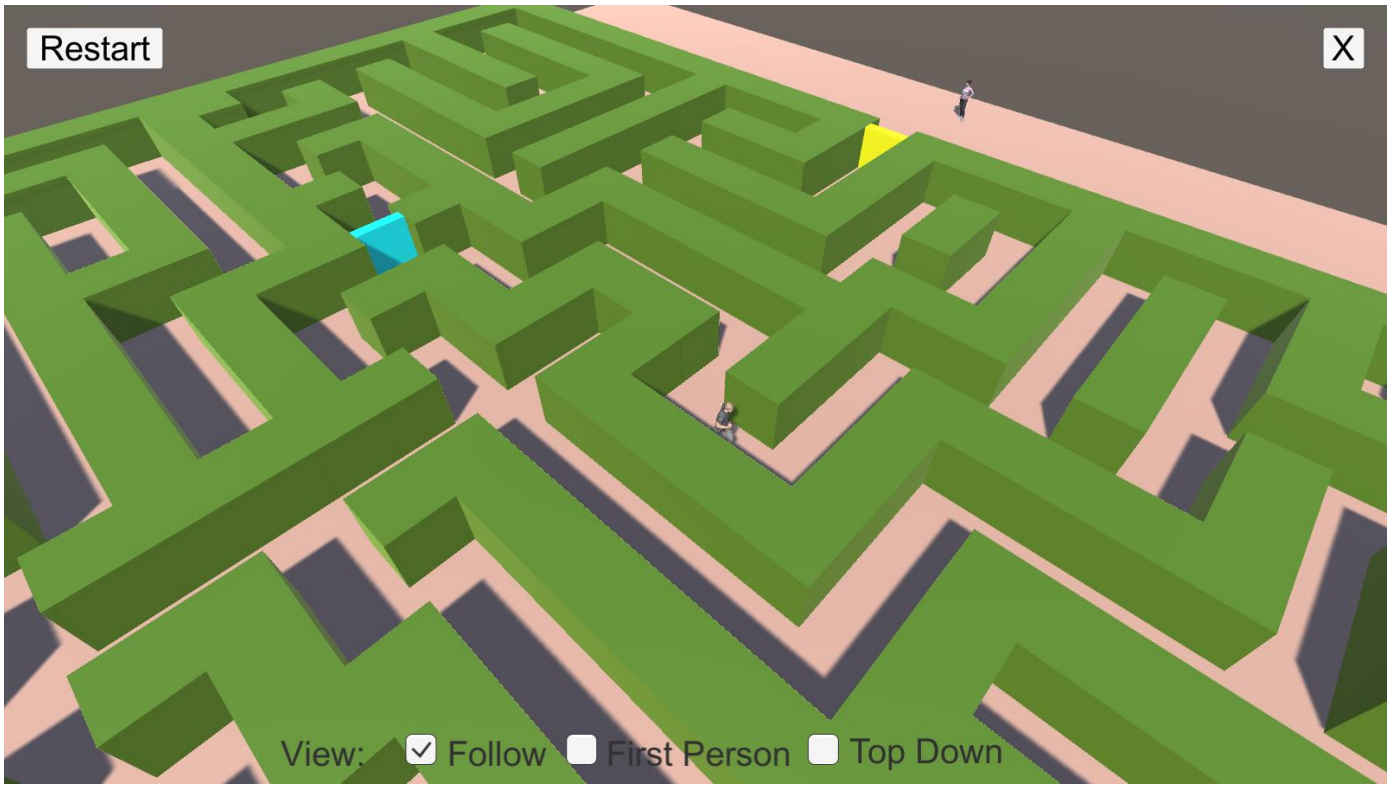


Όμως, με την πάροδο του χρόνου ο πράκτορας αρχίζει να ξεχνάει τις περιοχές του χάρτη που ανακάλυψε αρχικά και μόνο όταν τις επισκέπτεται πολλές φορές μπορεί να τις θυμάται για περισσότερο χρόνο. Διαθέτει δηλαδή χωρική μνήμη η οποία εξασθενεί όσο περνάει η ώρα.

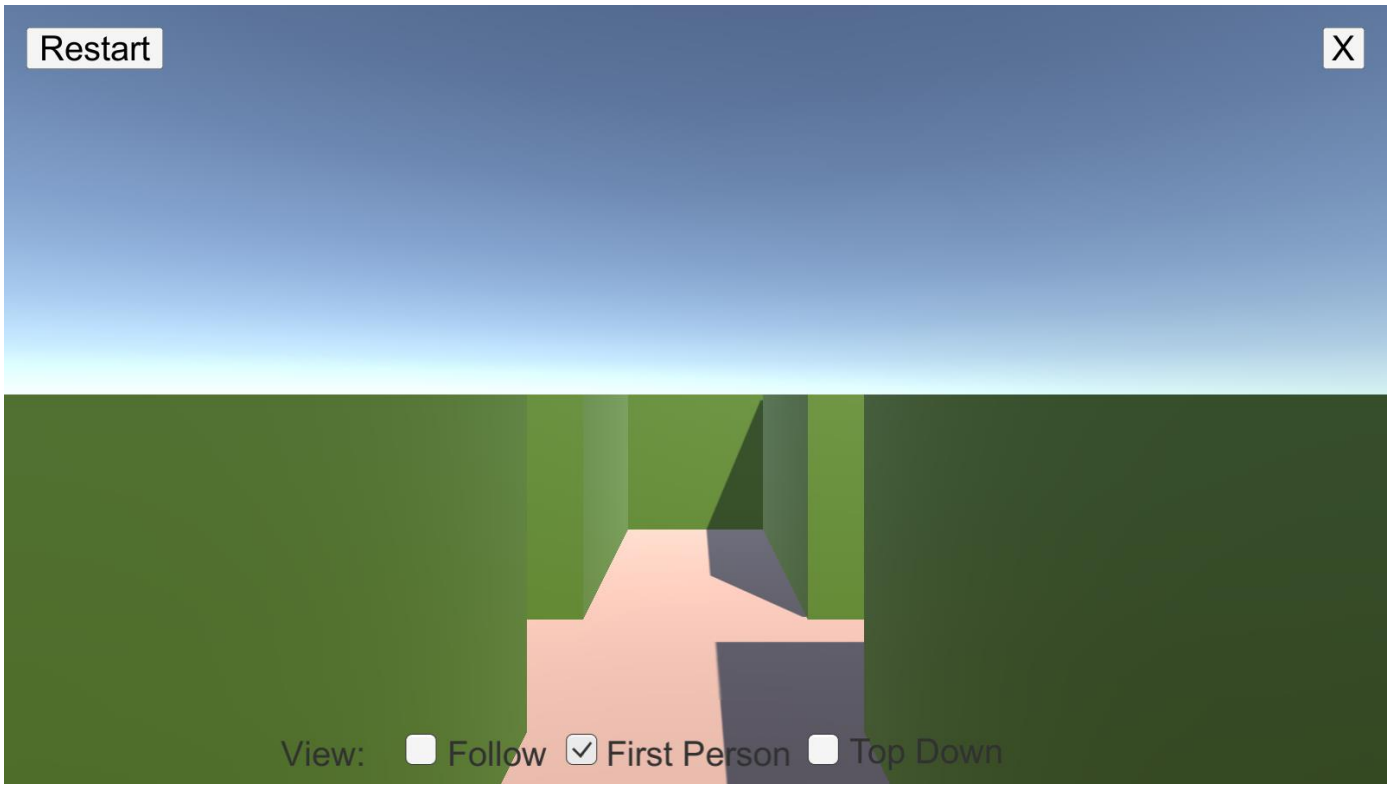


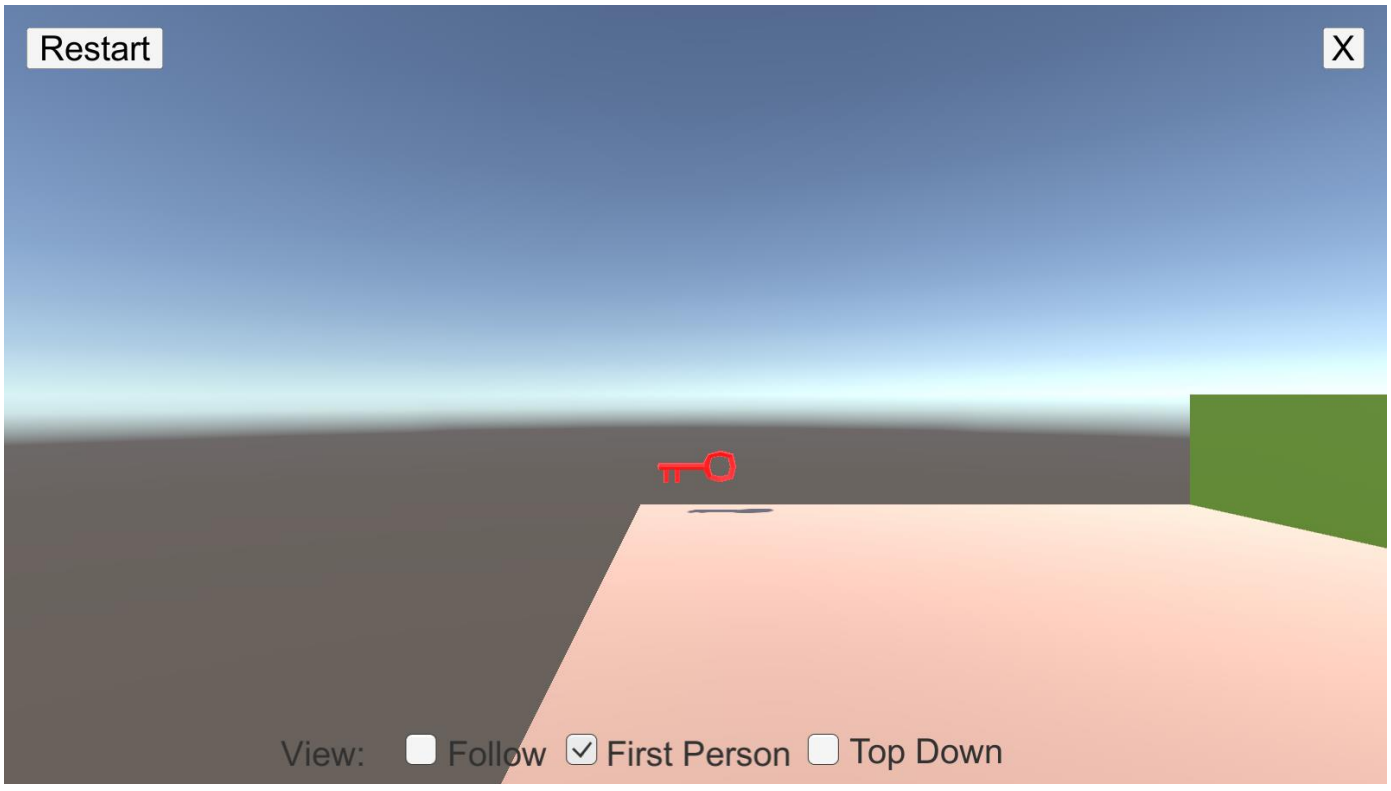


Όσο ο πράκτορας κινείται στο χώρο ο χρήστης έχει τη δυνατότητα να αλλάζει την οπτική γωνία της προσομοίωσης επιλέγοντας μεταξύ τριών διαφορετικών προοπτικών. Η προκαθορισμένη επιλογή **Follow** αντιστοιχεί στην προοπτική τρίτου προσώπου που ακολουθεί τον πράκτορα από ψηλά.

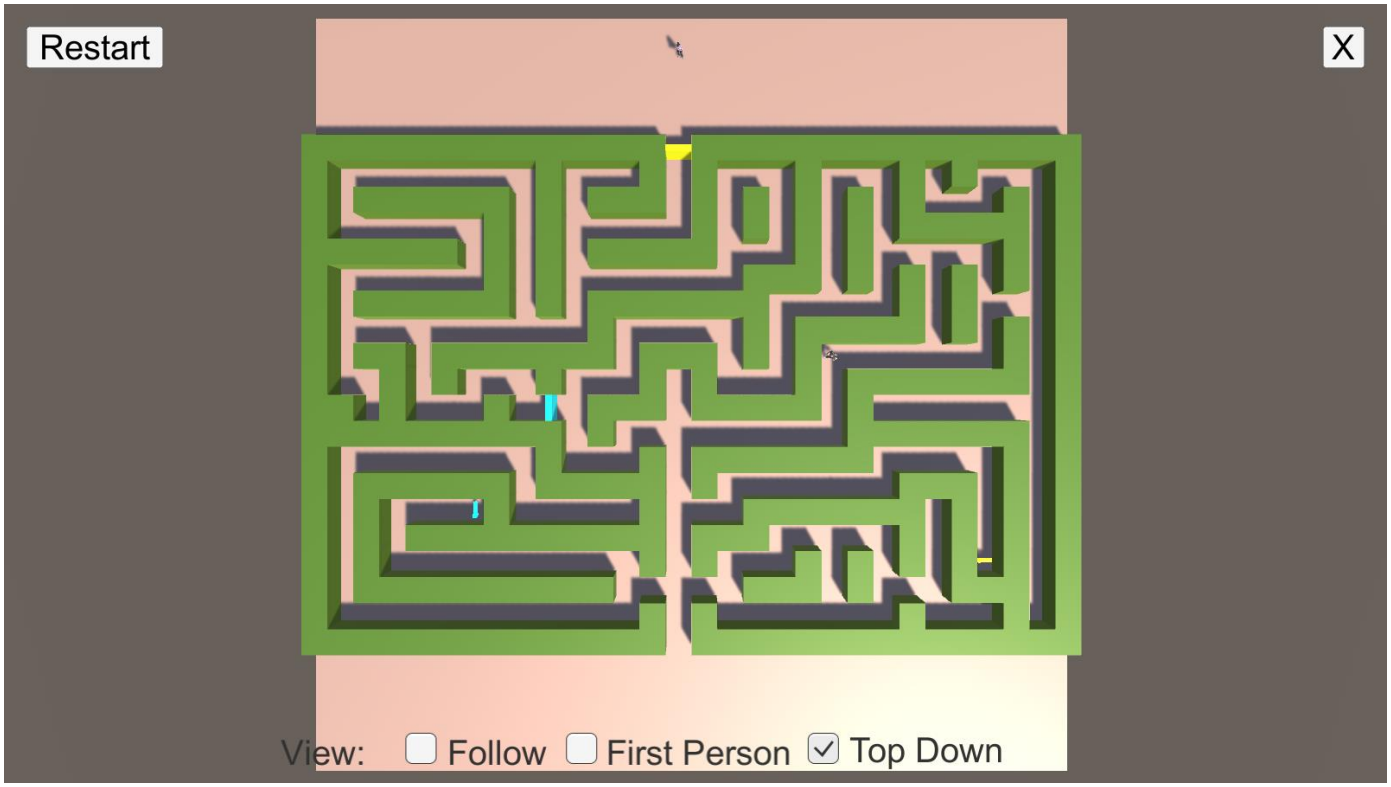


Η δεύτερη επιλογή είναι η προοπτική πρώτου προσώπου, **First Person**, που δείχνει με ακρίβεια το πεδίο όρασης του πράκτορα και επιτρέπει στον χρήστη να βλέπει από τα μάτια του τη στιγμή που έχει εντοπίσει κάτι, όπως για παράδειγμα ένα κλειδί.





Η τρίτη επιλογή, **Top Down**, είναι η προοπτική από πάνω προς τα κάτω όπου μπορεί ο χρήστης να παρατηρήσει όλα τα πιθανά μονοπάτια που μπορεί να πάρει ο πράκτορας και έχει με αυτό τον τρόπο την εποπτεία ολόκληρου του λαβυρίνθου και του υπόλοιπου χώρου έξω από αυτόν. Πατώντας **Restart** η προσομοίωση ξεκινάει από την αρχή.

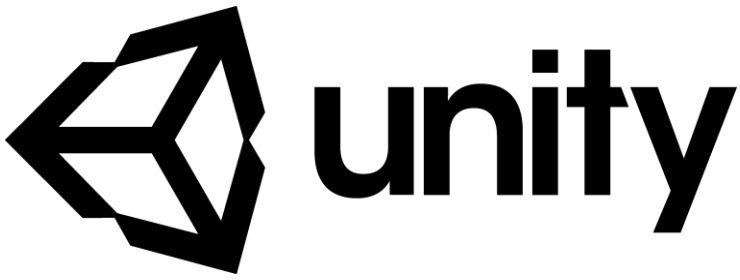


Ενώ η Unity, η μηχανή που χρησιμοποίησα για να δημιουργήσω την εφαρμογή, διαθέτει πολλά έτοιμα εργαλεία που βοηθούν στην πλοήγηση ενός πράκτορα, επέλεξα να υλοποιήσω δικά μου για να εμβαθύνω τις γνώσεις μου πάνω στην διαδικασία. Όσον αφορά το στήσιμο των προοπτικών και τις μεταβάσεις χρησιμοποίησα το Cinemachine, ένα package που παρέχει μεγάλο εύρος δυνατοτήτων και ελέγχου της βασικής κάμερας. Τα 3D μοντέλα των χαρακτήρων πάρθηκαν από το Mixamo, μια διαδικτυακή πλατφόρμα που διαθέτει δωρεάν μοντέλα και animations χαρακτήρων.

UNITY

Η Unity είναι μια μηχανή ανάπτυξης παιχνιδιών που αναπτύχθηκε από την Unity Technologies και κυκλοφόρησε για πρώτη φορά τον Ιούνιο του 2005. Σήμερα, η UNAT δίνει στους χρήστες τη δυνατότητα να δημιουργούν παιχνίδια, animation, εικονική πραγματικότητα (VR), επαυξημένη πραγματικότητα (AR), αρχιτεκτονικές απεικονίσεις, προσομοιώσεις, διαδραστικές εμπειρίες τόσο σε 2D όσο και σε 3D και υποστηρίζει περισσότερες από 25 πλατφόρμες. Η κύρια γλώσσα προγραμματισμού που χρησιμοποιείται σε αυτήν είναι η C# η οποία είναι ιδανική για παιχνίδια με γνώμονα την απόδοση χάρη στην αυτόματη διαχείριση μνήμης που παρέχει.

Κάποια από τα βασικά πλεονεκτήματά της σε σύγκριση με άλλες μηχανές είναι η γρήγορη διαδικασία ανάπτυξης πρωτοτύπων, η ενσωματωμένη λειτουργικότητα πολλαπλών χρήσεων (editor, scripting, graphics, physics κλπ.) και μια συνεχώς αυξανόμενη κοινότητα που παρέχει τεκμηρίωση, υποστήριξη, forums και tutorials που βοηθούν στην γρήγορη και αποτελεσματική επίλυση προβλημάτων. Ακόμα, διαθέτει το Unity Asset Store μέσω του οποίου οι χρήστες μπορούν να αναπτύξουν και να πουλήσουν assets σε άλλους χρήστες. Τα assets μπορούν να επιταχύνουν κατά πολύ την ανάπτυξη ενός παιχνιδιού γιατί παρέχουν έτοιμες λύσεις τόσο στους προγραμματιστές αλλά και στους καλλιτέχνες που τα χρησιμοποιούν.



ΤΕΧΝΗΤΗ ΝΟΗΜΟΣΥΝΗ ΣΤΑ ΒΙΝΤΕΟΠΑΙΧΝΙΔΙΑ

Στα βιντεοπαιχνίδια, η τεχνητή νοημοσύνη (A.I. - AI) χρησιμοποιείται για τη δημιουργία αποκριτικών, προσαρμοστικών ή έξυπνων συμπεριφορών που μοιάζουν με την ανθρώπινη νοημοσύνη κυρίως σε χαρακτήρες που δεν χειρίζονται οι παίκτες (NPC). Η τεχνητή νοημοσύνη αποτελεί αναπόσπαστο κομμάτι των βιντεοπαιχνιδιών από την έναρξή τους στη δεκαετία του 1950. Το AI στα βιντεοπαιχνίδια είναι ένα ξεχωριστό υποπεδίο και διαφέρει από το ακαδημαϊκό AI. Χρησιμεύει περισσότερο στη βελτίωση της εμπειρίας του παίκτη παρά στη μηχανική εκμάθηση ή στη λήψη αποφάσεων. Κατά τη διάρκεια της χρυσής εποχής των βιντεοπαιχνιδιών (arcade), η ιδέα των αντιπάλων που διαθέτουν τεχνητή νοημοσύνη διαδόθηκε σε μεγάλο βαθμό με τη μορφή κλιμακωτών επιπέδων δυσκολίας, διακριτών μοτίβων κίνησης και γεγονότων εντός του παιχνιδιού που εξαρτώνται από την αλληλεπίδρασή του με τον παίκτη.

Σήμερα, αναμφισβήτητα η πιο δημοφιλής κατηγορία μεθόδων AI για την ανάπτυξη παιχνιδιών είναι η Ad-Hoc Behavior Authoring. Οι **Finite state machines**, **Behavior Trees** και **Utility-Based** είναι τεχνικές ad-hoc behavior authoring που παραδοσιακά κυριαρχούσαν για να καθοδηγήσουν τις ενέργειες των NPCs. Η κυριαρχία τους είναι εμφανής από το γεγονός ότι ο όρος «*game AI*» στη σκηνή ανάπτυξης παιχνιδιών είναι ακόμα σήμερα συνώνυμος με τη χρήση αυτών των μεθόδων.

FINITE STATE MACHINES

Οι **Finite State Machines** (**FSM**) παρουσιάζονται ως γραφήματα. Συγκεκριμένα, το γράφημα περιέχει κόμβους (states) που ενσωματώνουν κάποια μαθηματική αφαίρεση (δομή) και ακμές (transitions) που αντιπροσωπεύουν μια υπό όρους σχέση μεταξύ των κόμβων. Μια **FSM** μπορεί να είναι μόνο σε μία κατάσταση κάθε φορά. Η τρέχουσα κατάσταση μπορεί να μεταβεί σε μια άλλη αν πληρείται η κατάλληλη συνθήκη. Οι **FSMs** είναι απίστευτα απλές στο σχεδιασμό, την υλοποίηση, την απεικόνιση και τον εντοπισμό σφαλμάτων. Επιπλέον, έχουν αποδείξει ότι δουλεύουν καλά με τα παιχνίδια με τα χρόνια της συνύπαρξής τους. Ωστόσο, μπορεί να γίνουν εξαιρετικά περίπλοκες για το σχεδιασμό σε μεγάλη κλίμακα και περιορίζονται υπολογιστικά σε ορισμένες εργασίες εντός του παιχνιδιού AI. Ένας επιπλέον κρίσιμος περιορισμός όλων των ad-hoc μεθόδων είναι ότι δεν είναι ευέλικτες και δυναμικές. Μετά την ολοκλήρωση του σχεδιασμού, τη δοκιμή και την αποσφαλμάτωση, υπάρχει περιορισμένος χώρος για προσαρμοστικότητα και εξέλιξη. Ως αποτέλεσμα, οι **FSMs** χρησιμοποιούνται μόνο για την υλοποίηση πολύ προβλέψιμων συμπεριφορών στα παιχνίδια.

BEHAVIOR TREES

Ένα **Behavior Tree** (**BT**), παρόμοια με μια **FSM**, μοντελοποιεί μεταβάσεις μεταξύ ενός πεπερασμένου συνόλου εργασιών (ή συμπεριφορών). Η ισχύς των **BTs** σε σύγκριση με τις **FSMs** είναι η αρθρωτότητά τους (modularity): εάν έχουν σχεδιαστεί καλά, μπορούν να αποδώσουν πολύπλοκες συμπεριφορές που αποτελούνται από απλές εργασίες. Η κύρια διαφορά μεταξύ **FSM** και **BT** είναι ότι αποτελούνται από συμπεριφορές και όχι από καταστάσεις. Όπως και με τις **FSMs**, τα **BTs** είναι εύκολο να σχεδιαστούν, να δοκιμαστούν και να εντοπιστούν τυχόν σφάλματα. Το **BT** χρησιμοποιεί μια δομή δέντρου με έναν αρχικό κόμβο και έναν αριθμό γονικών και αντίστοιχων κόμβων-παιδιά που αντιπροσωπεύουν συμπεριφορές. Διασχίζουμε ένα **BT** ξεκινώντας από τη ρίζα. Στη συνέχεια, ενεργοποιούμε την εκτέλεση ζευγών γονέα-παιδιού όπως δηλώνεται στο δέντρο. Ένα παιδί μπορεί να επιστρέψει τις ακόλουθες τιμές στον γονέα σε προκαθορισμένα βήματα χρόνου (ticks): *run* εάν η συμπεριφορά είναι ακόμα ενεργή, *success* εάν η συμπεριφορά έχει ολοκληρωθεί, *failure* εάν η συμπεριφορά απέτυχε. Σε σύγκριση με το **FSM**, τα **BT** είναι πιο ευέλικτα στο σχεδιασμό και πιο εύκολα στη δοκιμή. Ωστόσο, πάσχουν από παρόμοια μειονεκτήματα. Συγκεκριμένα, η δυναμικότητά τους είναι μάλλον χαμηλή δεδομένου ότι αντιπροσωπεύουν στατική γνώση.

UTILITY-BASED

Μια ολοένα και πιο δημοφιλής μέθοδος Ad-Hoc Behavior Authoring που εξαλείφει τους περιορισμούς των **FSMs** και των **BTs** διατηρώντας το modularity, είναι η **Utility-Based**. Ακολουθώντας αυτή την προσέγγιση, σε κάθε στιγμιότυπο του παιχνιδιού ανατίθεται μια συγκεκριμένη μέτρηση χρησιμότητας (utility) η οποία δίνει μια αξία στη σημασία του.

Για παράδειγμα, η σημασία του εχθρού να βρίσκεται σε μια συγκεκριμένη απόσταση από τον παίκτη για να μπορέσει να του επιτεθεί. Δεδομένου του συνόλου όλων των utilities που διατίθενται σε έναν πράκτορα και όλων των επιλογών που διαθέτει, το **Utility-Based** AI αποφασίζει ποια είναι η πιο σημαντική επιλογή που πρέπει να εξετάσει αυτή τη στιγμή. Αυτή η προσέγγιση στηρίζεται στη θεωρία Utility των οικονομικών. Ένα utility μπορεί να μετρήσει οτιδήποτε, από παρατηρήσιμα αντικειμενικά δεδομένα (π.χ. υγεία του εχθρού) έως υποκειμενικές έννοιες όπως συναισθήματα, διάθεση και απειλές. Τα διάφορα utilities σχετικά με πιθανές ενέργειες ή αποφάσεις μπορούν να συγκεντρωθούν σε γραμμικούς ή μη γραμμικούς τύπους και να καθοδηγήσουν τον πράκτορα να λαμβάνει αποφάσεις με βάση τη συγκεντρωτική χρησιμότητα. Οι τιμές χρησιμότητας μπορούν να ελέγχονται σε κάθε καρέ του παιχνιδιού. Έτσι, ενώ οι **FSMs** και τα **BTs** θα εξέταζαν μία απόφαση κάθε φορά, οι αρχιτεκτονικές **Utility-Based** AI θα εξετάσουν όλες τις διαθέσιμες επιλογές, θα τους αναθέσουν ένα utility και θα επιλέξουν την πιο κατάλληλη επιλογή (υψηλότερη χρησιμότητα).

Το **Utility-Based** AI έχει ορισμένα πλεονεκτήματα σε σύγκριση με άλλες Ad-Hoc Behavior Authoring μεθόδους :

- Είναι αρθρωτό (modular), καθώς η απόφαση του πράκτορα του παιχνιδιού εξαρτάται από έναν αριθμό διαφορετικών παραγόντων (ή εκτιμήσεων).
- Αυτή η λίστα παραγόντων μπορεί να είναι δυναμική.
- Το **Utility-Based** AI είναι επίσης επεκτάσιμο, καθώς μπορούμε εύκολα να συντάξουμε νέους τύπους προβληματισμών όταν τους θεωρήσουμε κατάλληλους.
- Τέλος, η μέθοδος μπορεί να επαναχρησιμοποιηθεί καθώς τα utilities μπορούν να μεταφερθούν από τη μία απόφαση στην άλλη και από ένα παιχνίδι σε ένα άλλο παιχνίδι.

Ως αποτέλεσμα αυτών των πλεονεκτημάτων, το **Utility-Based** AI γίνεται όλο και πιο δημοφιλές στη βιομηχανία παιχνιδιών.

GOAP

Έχοντας δοκιμάσει κάποιες από τις προαναφερθείσες μεθόδους παρατήρησα ότι για πράκτορες που έχουν πολλές συμπεριφορές αυξάνεται κατά πολύ η πολυπλοκότητα των διαφόρων καταστάσεων, ώσπου τελικά φτάνει σε ένα σημείο όπου δεν μπορεί κάποιος να προσθέσει νέες συμπεριφορές επειδή προκαλούν πάρα πολλές παρενέργειες στην AI. Λύση στο παραπάνω πρόβλημα έρχεται να δώσει η μέθοδος **GOAP**.

Το **Goal Oriented Action Planning** (**GOAP**) είναι μία μέθοδος τεχνητής νοημοσύνης που δίνει εύκολα στους πράκτορες επιλογές και εργαλεία για τη λήψη έξυπνων αποφάσεων χωρίς να χρειάζεται μια μεγάλη και πολύπλοκη **FSM**. Το **GOAP** επιτρέπει στους πράκτορες να σχεδιάζουν μια ακολουθία ενεργειών για την ικανοποίηση ενός συγκεκριμένου στόχου. Η συγκεκριμένη σειρά ενεργειών εξαρτάται όχι μόνο από τον στόχο αλλά και από την τρέχουσα κατάσταση του κόσμου και του πράκτορα. Αυτό σημαίνει ότι εάν παρέχεται ο ίδιος στόχος σε διαφορετικούς πράκτορες μπορεί να παρθεί μια εντελώς διαφορετική ακολουθία ενεργειών που κάνει τις συμπεριφορές τους να φαίνονται πιο δυναμικές και ρεαλιστικές.

Αυτό επιτυγχάνεται αποσυνδέοντας τις δράσεις μεταξύ τους. Έτσι μπορούμε πλέον να επικεντρωθούμε σε κάθε δράση ξεχωριστά. Αυτό καθιστά τον κώδικα αρθρωτό και εύκολο στη δοκιμή και τη συντήρηση του. Αν θελήσουμε να προσθέσουμε μια νέα δράση το μόνο που έχουμε να κάνουμε είναι να τη βάλουμε στη λίστα δράσεων και το **GOAP** θα φροντίσει τα υπόλοιπα χωρίς να χρειάζεται να αλλάξουμε τον κώδικα ή να κάνουμε κάποια άλλη ενέργεια.

ΔΡΑΣΕΙΣ (GOAP Actions)

Μια δράση είναι κάτι που μπορεί να κάνει ο πράκτορας. Στην εφαρμογή που δημιούργησα ο πράκτορας μπορεί να κάνει τέσσερις δράσεις: **GoTo**, **OpenDoor**, **GetKey**, και **SearchForKey**.

Η δράση **GoTo** είναι η σημαντικότερη δράση που έχει ο πράκτορας και είναι και ο βασικός του στόχος στην προσομοίωση. Να φτάσει δηλαδή σε ένα συγκεκριμένο χωρικό σημείο στον κόσμο όπου εκεί βρίσκεται ο στόχος του. Στην περίπτωσή μας ο στόχος είναι το σημείο στο οποίο βρίσκεται η φίλη του.

Η δράση **OpenDoor** επιτρέπει στον πράκτορα να ξεκλειδώσει μία πόρτα εφόσον έχει συλλέξει το κατάλληλο κλειδί για αυτήν.

Η δράση **SearchForKey** δίνει την δυνατότητα στον πράκτορα να ψάξει για ένα κλειδί στον χώρο εφόσον έχει εντοπίσει μία πόρτα και δεν έχει βρει το κατάλληλο κλειδί για να την ανοίξει.

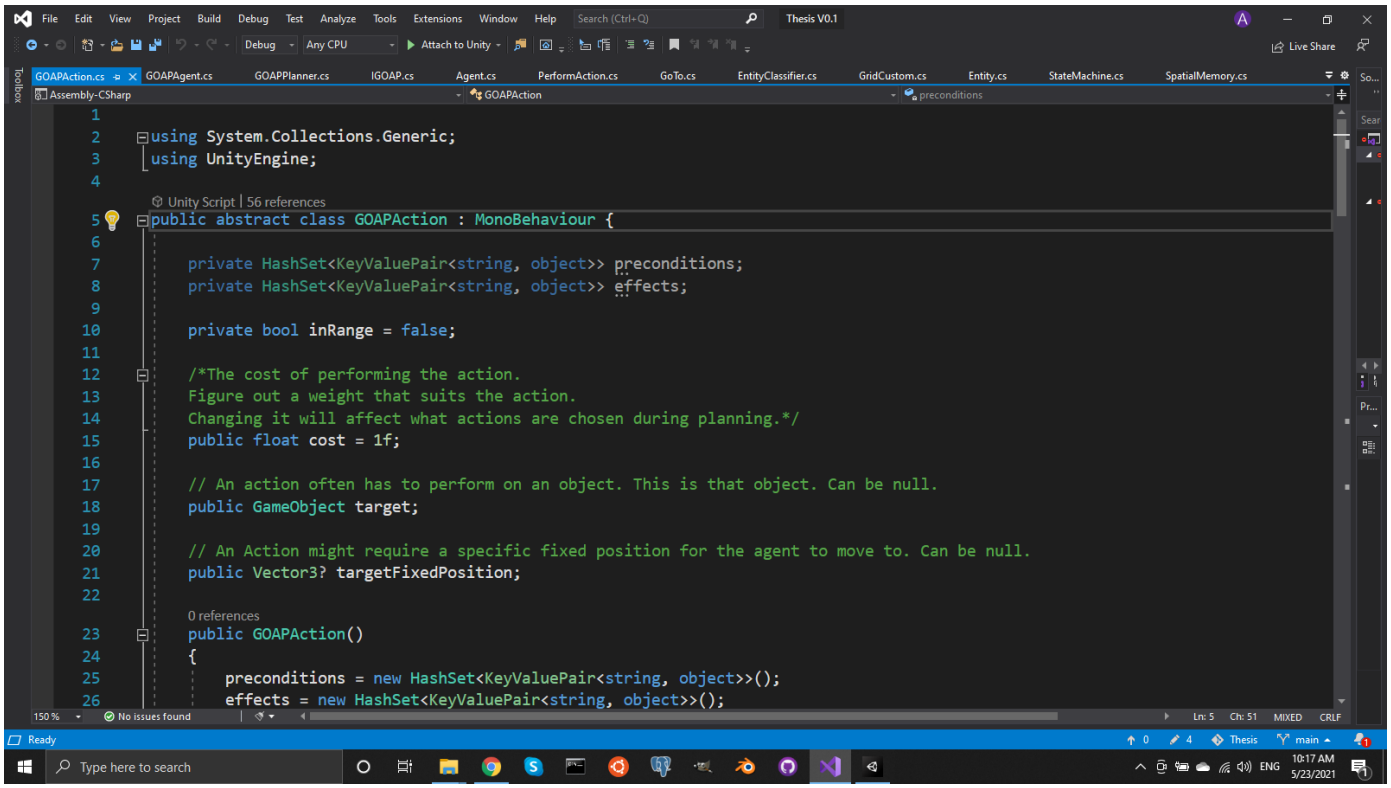
Η δράση **GetKey** κάνει τον πράκτορα να συλλέξει ένα κλειδί αφού το έχει εντοπίσει.

Περιγράφοντας τις παραπάνω δράσεις παρατηρούμε ότι η κάθε δράση έχει κάποιες προϋποθέσεις (preconditions) και κάποια αποτελέσματα (effects). Προϋπόθεση είναι η κατάσταση που απαιτείται για την εκτέλεση της δράσης και το αποτέλεσμα είναι η αλλαγή στην κατάσταση μετά την εκτέλεση της δράσης.

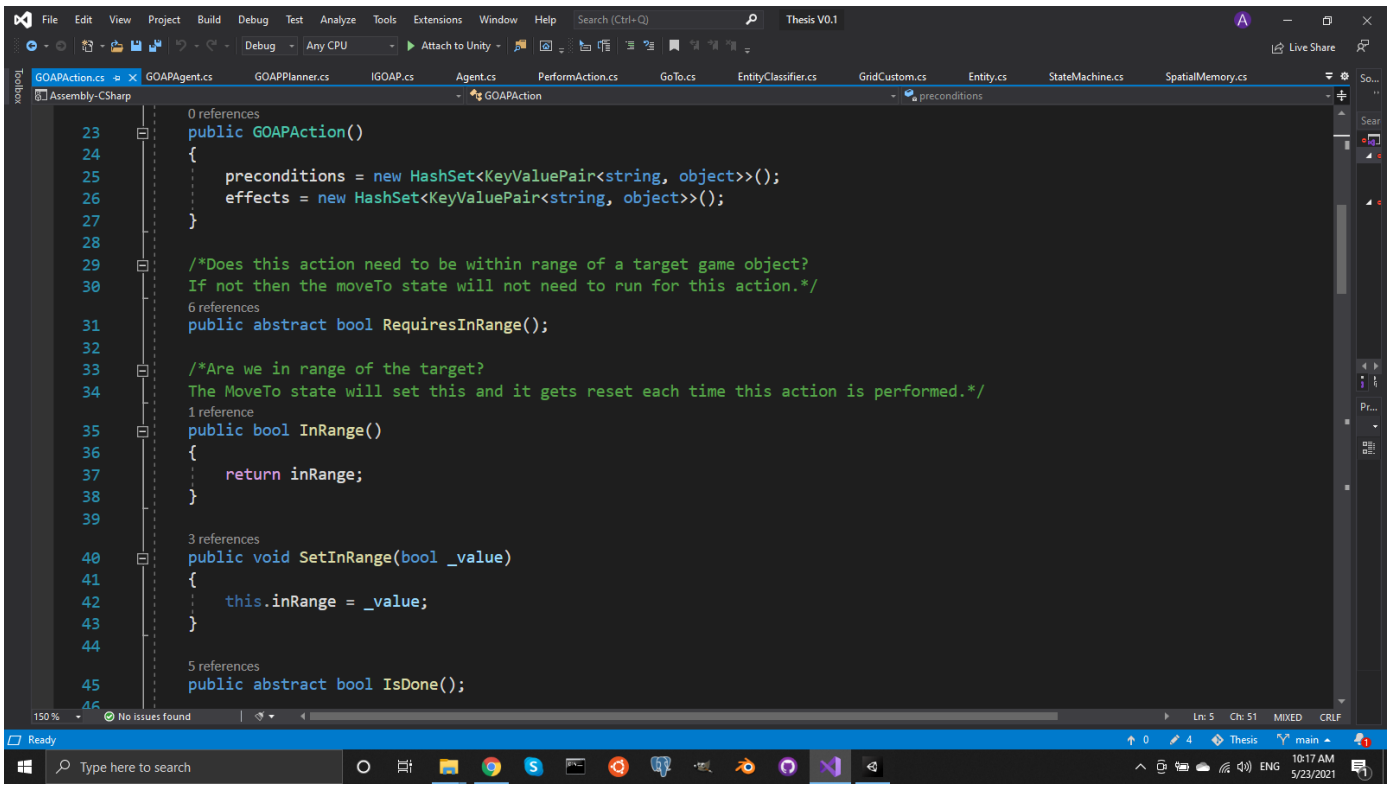
Για παράδειγμα, η ενέργεια **OpenDoor** απαιτεί από τον πράκτορα να έχει ένα κλειδί ίδιου χρώματος με την πόρτα για να εκτελεστεί. Εάν ο πράκτορας δεν έχει κλειδί, πρέπει να βρει μια άλλη δράση που μπορεί να εκπληρώσει αυτήν την προϋπόθεση για να επιτρέψει την εκτέλεση της δράσης **OpenDoor**. Ευτυχώς, η ενέργεια **GetKey** έχει αυτό το αποτέλεσμα.

Η κάθε δράση κληρονομεί από την κλάση **GOAPAction** η οποία είναι υπεύθυνη για την αποθήκευση των προϋποθέσεων και των αποτελεσμάτων. Γνωρίζει επίσης εάν πρέπει να βρίσκεται εντός ενός στόχου και τότε ενημερώνει την **FSM** για να ενεργοποιήσει την κατάσταση **MoveTo** όταν χρειάζεται. Επίσης, ξέρει πότε έχει ολοκληρωθεί, το οποίο καθορίζεται από την δράση που την κληρονομεί.

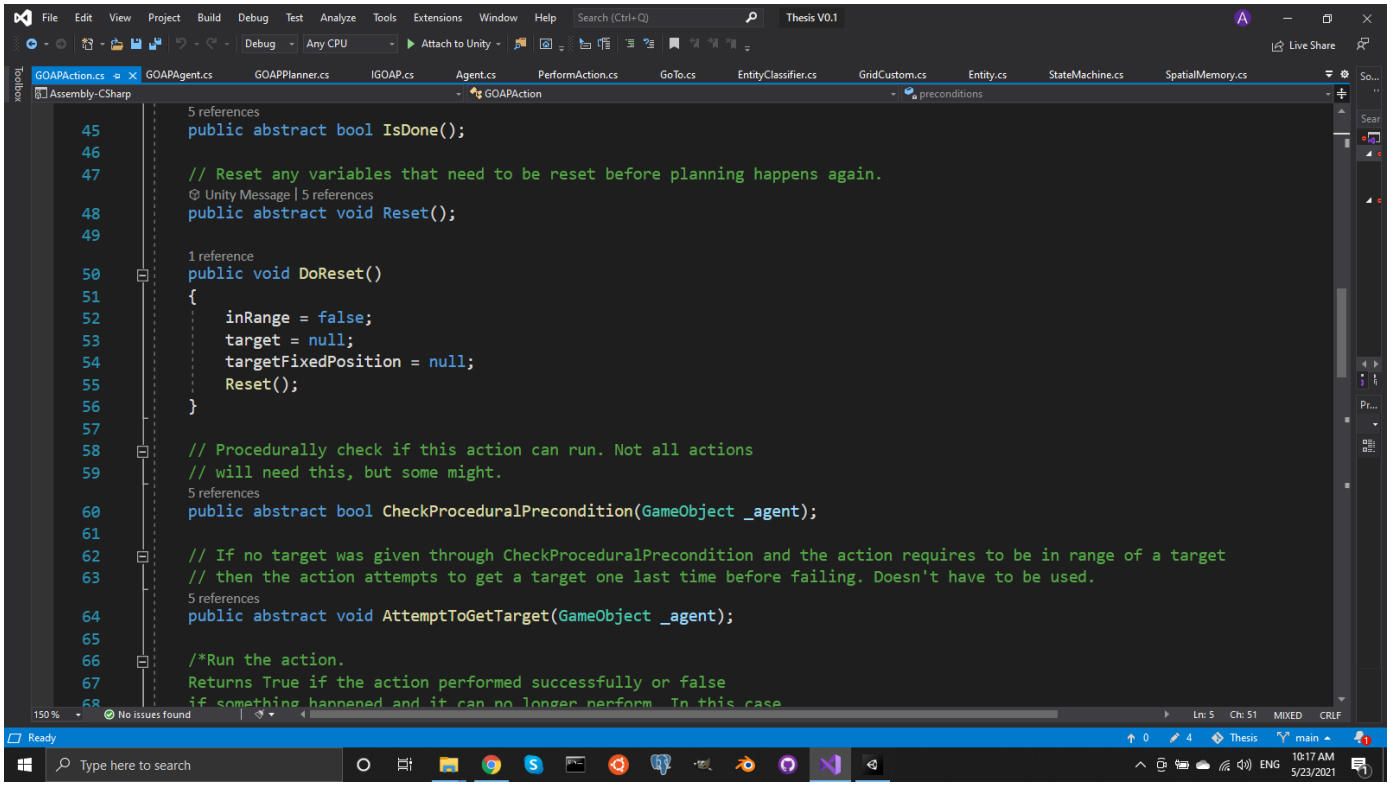
ΤΟ SCRIPT ΔΡΑΣΗΣ GOAPAction



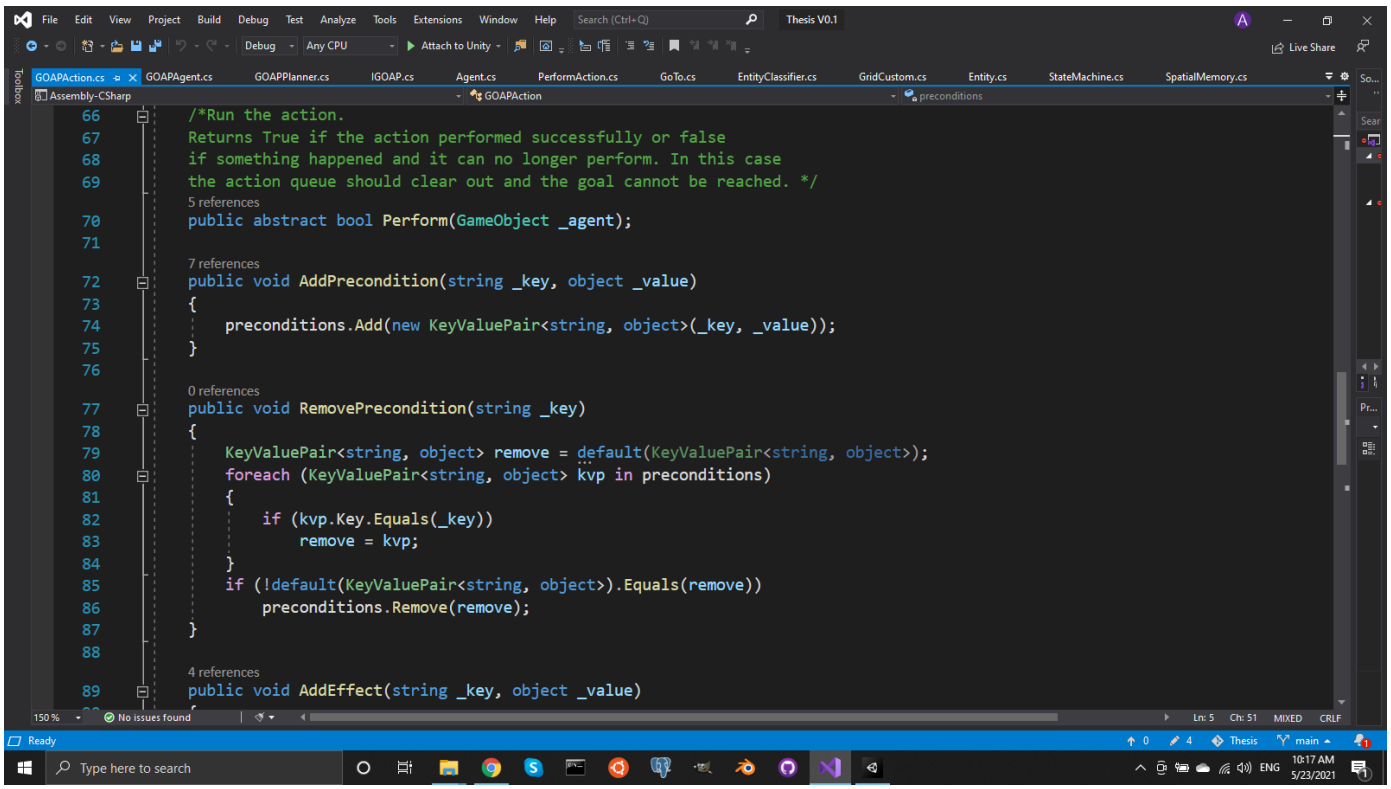
```
using System.Collections.Generic;
using UnityEngine;

Unity Script | 56 references
public abstract class GOAPAction : MonoBehaviour {

    private HashSet<KeyValuePair<string, object>> preconditions;
    private HashSet<KeyValuePair<string, object>> effects;

    private bool inRange = false;

    /*The cost of performing the action.
    Figure out a weight that suits the action.
    Changing it will affect what actions are chosen during planning.*/
    public float cost = 1f;

    // An action often has to perform on an object. This is that object. Can be null.
    public GameObject target;

    // An Action might require a specific fixed position for the agent to move to. Can be null.
    public Vector3? targetFixedPosition;

    0 references
    public GOAPAction()
    {
        preconditions = new HashSet<KeyValuePair<string, object>>();
        effects = new HashSet<KeyValuePair<string, object>>();
```



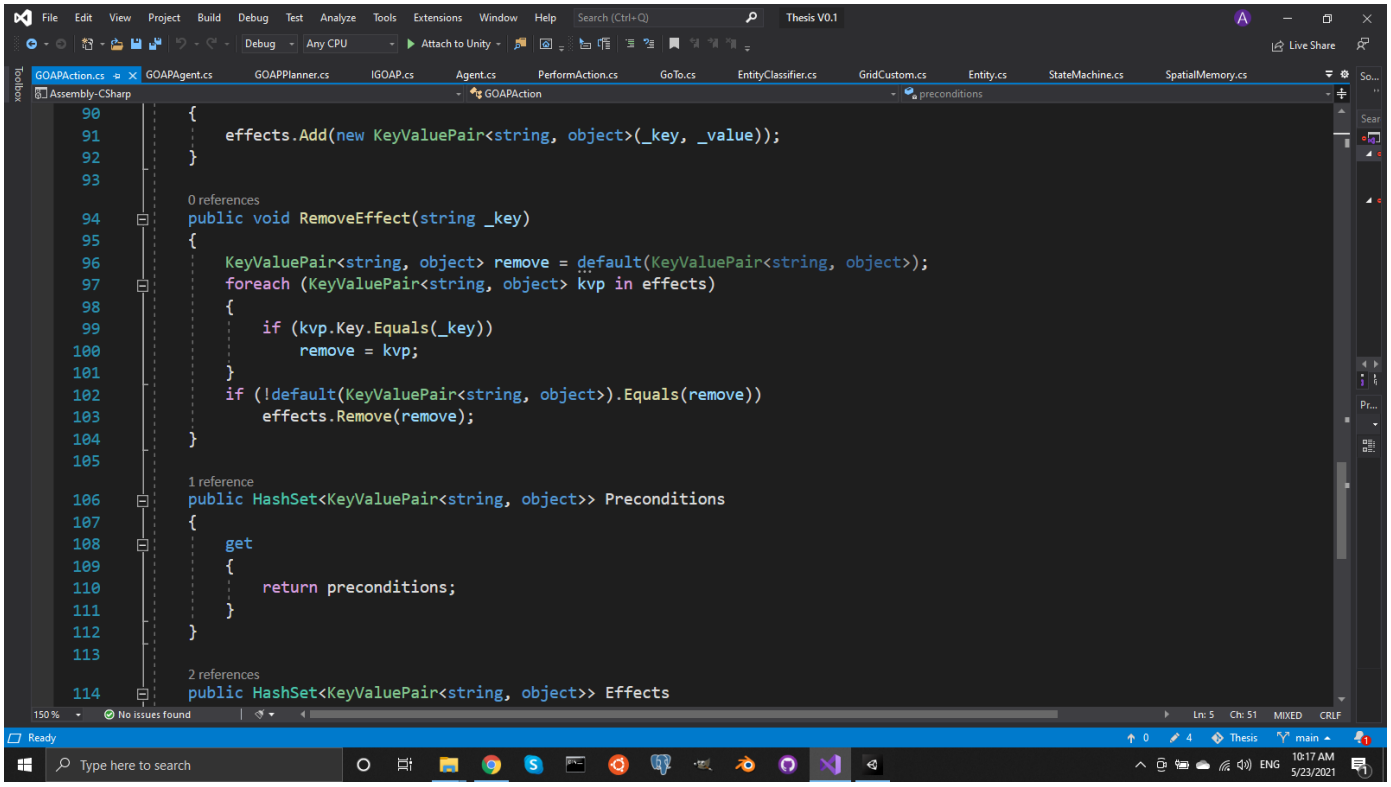
```
    0 references
    public GOAPAction()
    {
        preconditions = new HashSet<KeyValuePair<string, object>>();
        effects = new HashSet<KeyValuePair<string, object>>();
    }

    /*Does this action need to be within range of a target game object?
    If not then the moveTo state will not need to run for this action.*/
    6 references
    public abstract bool RequiresInRange();

    /*Are we in range of the target?
    The MoveTo state will set this and it gets reset each time this action is performed.*/
    1 reference
    public bool InRange()
    {
        return inRange;
    }

    3 references
    public void SetInRange(bool _value)
    {
        this.inRange = _value;
    }

    5 references
    public abstract bool IsDone();
```



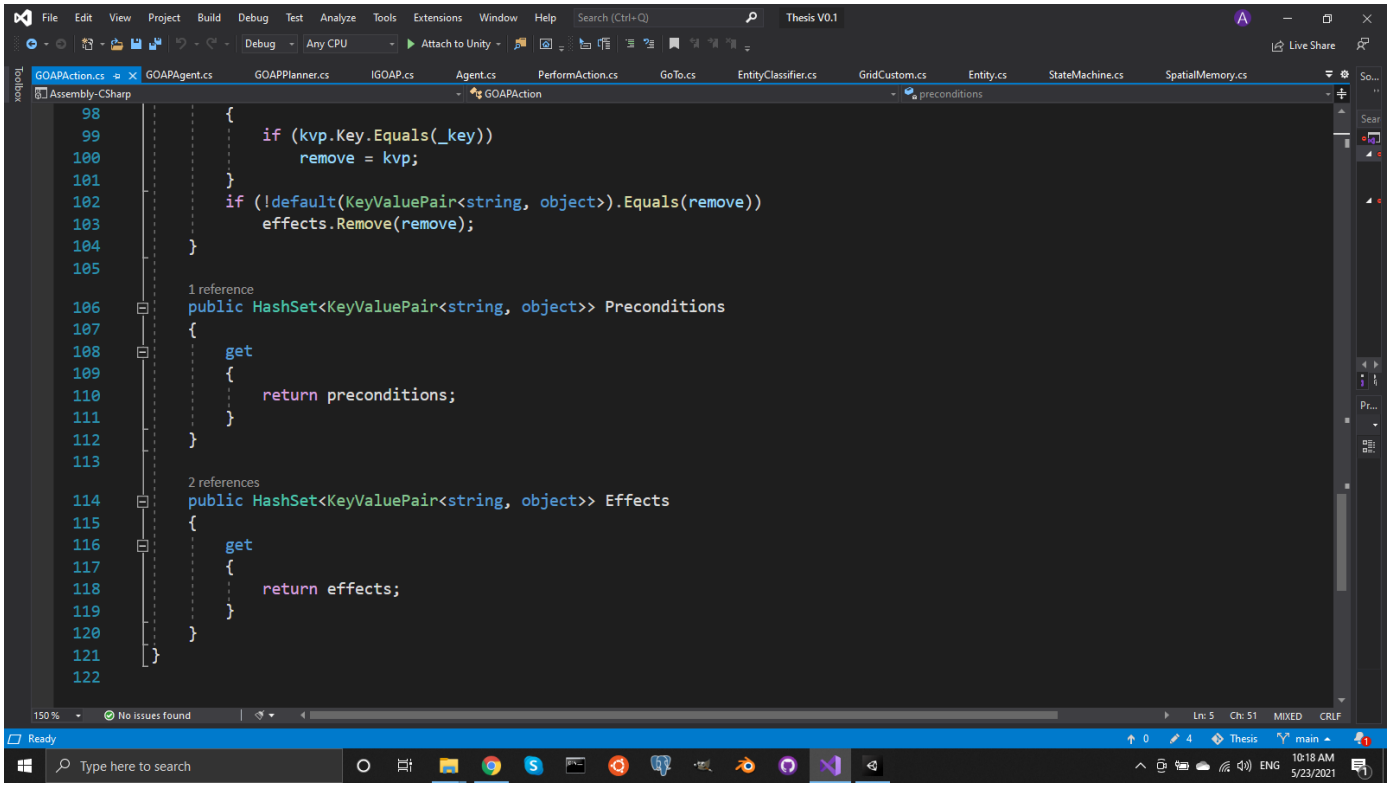
```
5 references
public abstract bool IsDone();

// Reset any variables that need to be reset before planning happens again.
Unity Message | 5 references
public abstract void Reset();

1 reference
public void DoReset()
{
    inRange = false;
    target = null;
    targetFixedPosition = null;
    Reset();
}

// Procedurally check if this action can run. Not all actions
// will need this, but some might.
5 references
public abstract bool CheckProceduralPrecondition(GameObject _agent);

// If no target was given through CheckProceduralPrecondition and the action requires to be in range of a target
// then the action attempts to get a target one last time before failing. Doesn't have to be used.
5 references
public abstract void AttemptToGetTarget(GameObject _agent);

/*Run the action.
Returns True if the action performed successfully or false
if something happened and it can no longer perform. In this case
```



```
/*Run the action.
Returns True if the action performed successfully or false
if something happened and it can no longer perform. In this case
the action queue should clear out and the goal cannot be reached. */
5 references
public abstract bool Perform(GameObject _agent);

7 references
public void AddPrecondition(string _key, object _value)
{
    preconditions.Add(new KeyValuePair<string, object>(_key, _value));
}

0 references
public void RemovePrecondition(string _key)
{
    KeyValuePair<string, object> remove = default(KeyValuePair<string, object>);
    foreach (KeyValuePair<string, object> kvp in preconditions)
    {
        if (kvp.Key.Equals(_key))
            remove = kvp;
    }
    if (!default(KeyValuePair<string, object>).Equals(remove))
        preconditions.Remove(remove);
}

4 references
public void AddEffect(string _key, object _value)
```

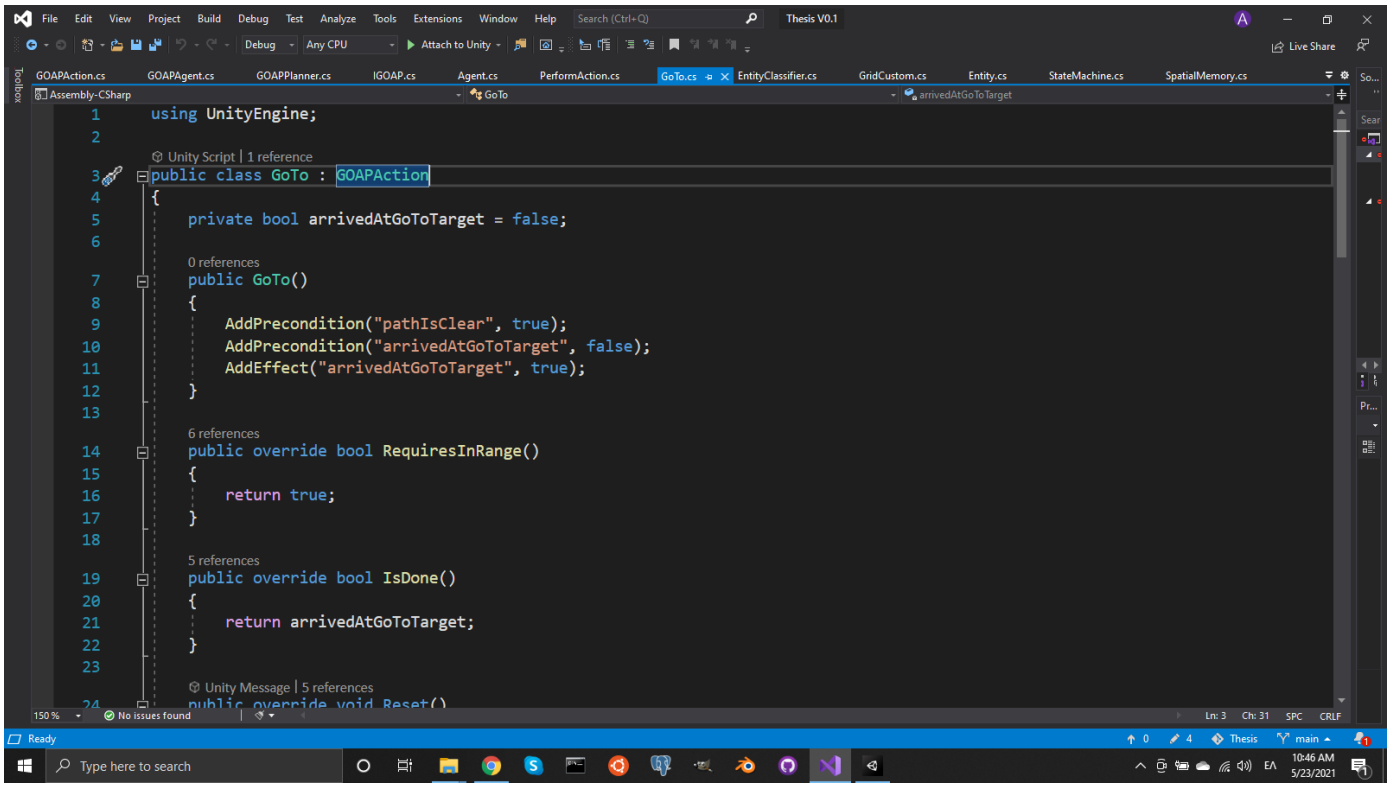


```
90          {
91              effects.Add(new KeyValuePair<string, object>(_key, _value));
92          }
93
            0 references
94          public void RemoveEffect(string _key)
95          {
96              KeyValuePair<string, object> remove = default(KeyValuePair<string, object>);
97              foreach (KeyValuePair<string, object> kvp in effects)
98              {
99                  if (kvp.Key.Equals(_key))
100                     remove = kvp;
101             }
102             if (!default(KeyValuePair<string, object>).Equals(remove))
103                 effects.Remove(remove);
104         }
105
            1 reference
106         public HashSet<KeyValuePair<string, object>> Preconditions
107         {
108             get
109             {
110                 return preconditions;
111             }
112         }
113
            2 references
114         public HashSet<KeyValuePair<string, object>> Effects
```

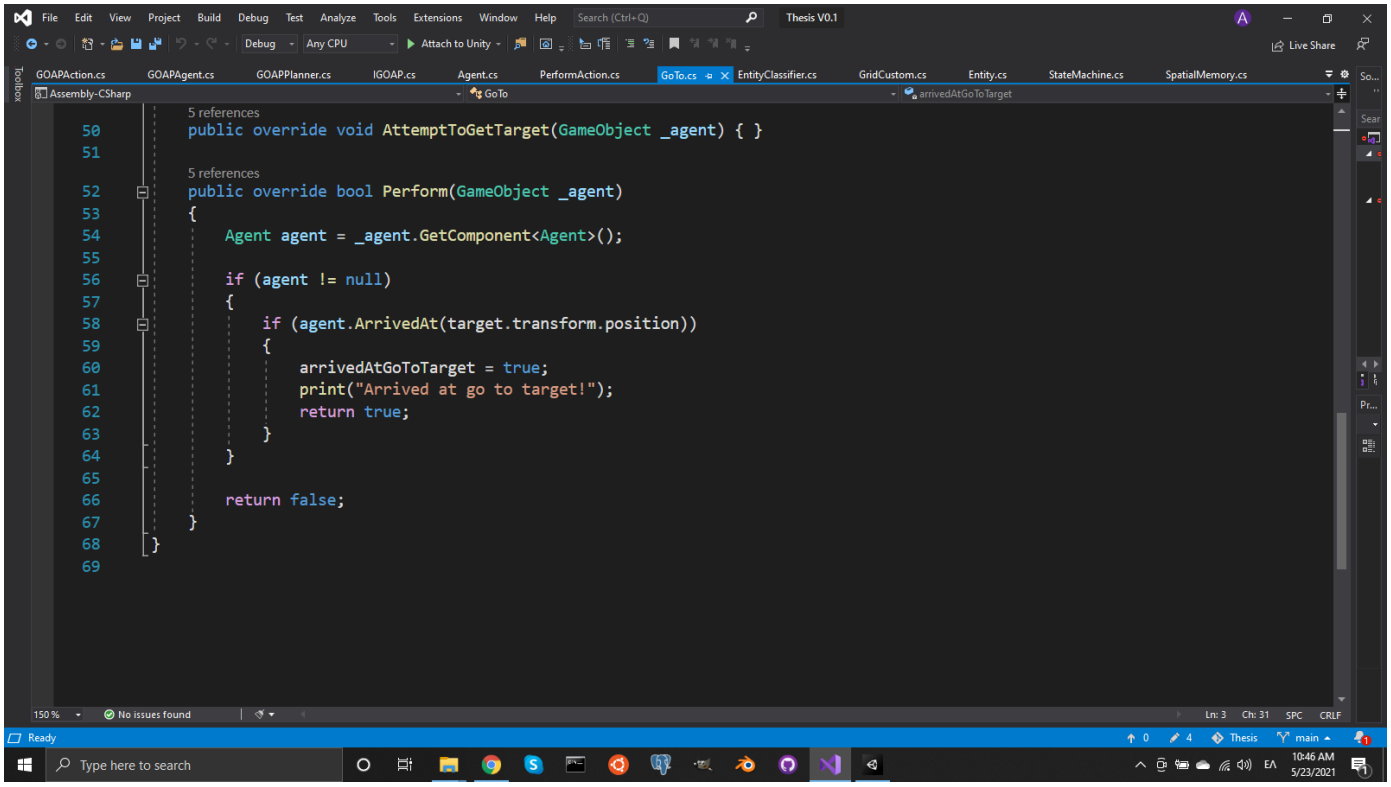


```
98          {
99              if (kvp.Key.Equals(_key))
100                 remove = kvp;
101             }
102             if (!default(KeyValuePair<string, object>).Equals(remove))
103                 effects.Remove(remove);
104         }
105
            1 reference
106         public HashSet<KeyValuePair<string, object>> Preconditions
107         {
108             get
109             {
110                 return preconditions;
111             }
112         }
113
            2 references
114         public HashSet<KeyValuePair<string, object>> Effects
115         {
116             get
117             {
118                 return effects;
119             }
120         }
121     }
122
```

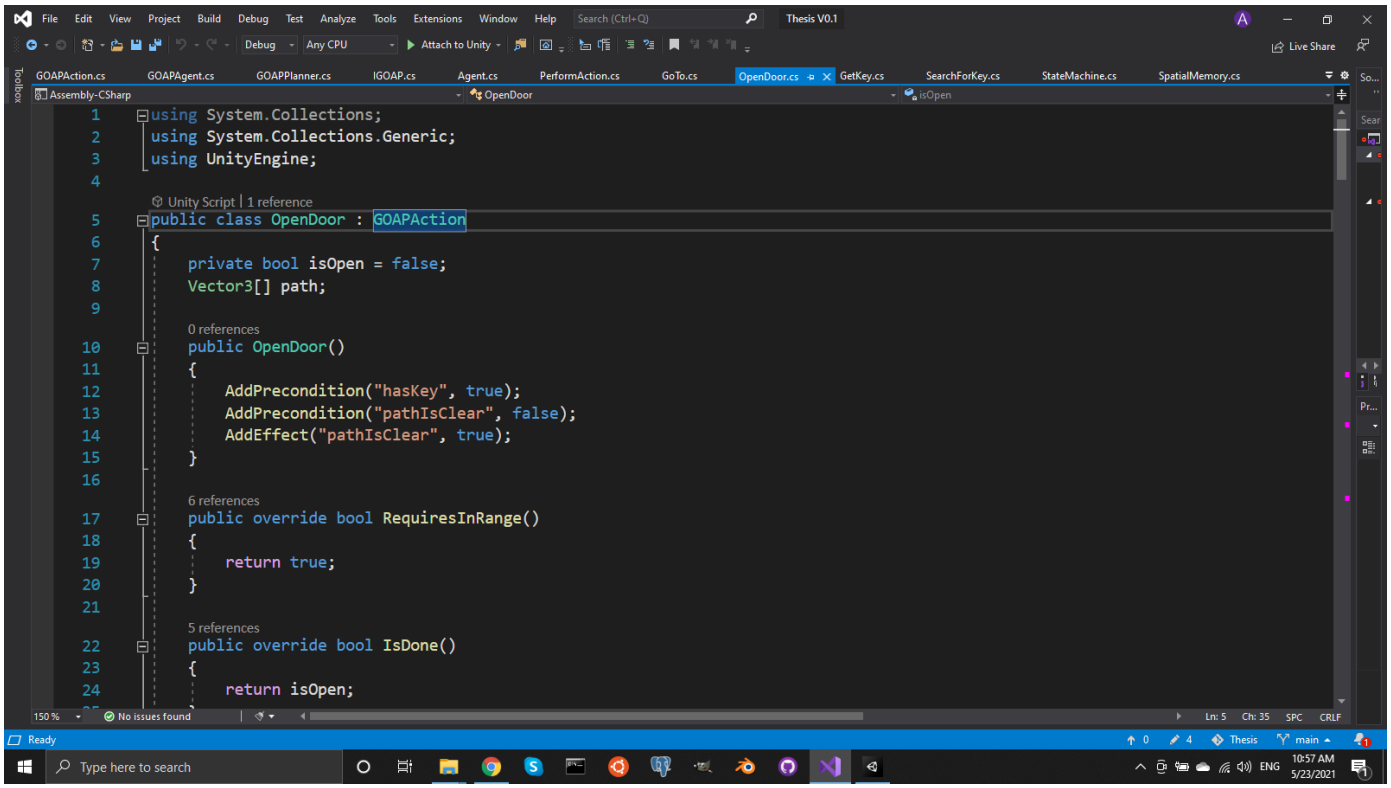
ΤΟ SCRIPT ΔΡΑΣΗΣ GoTo



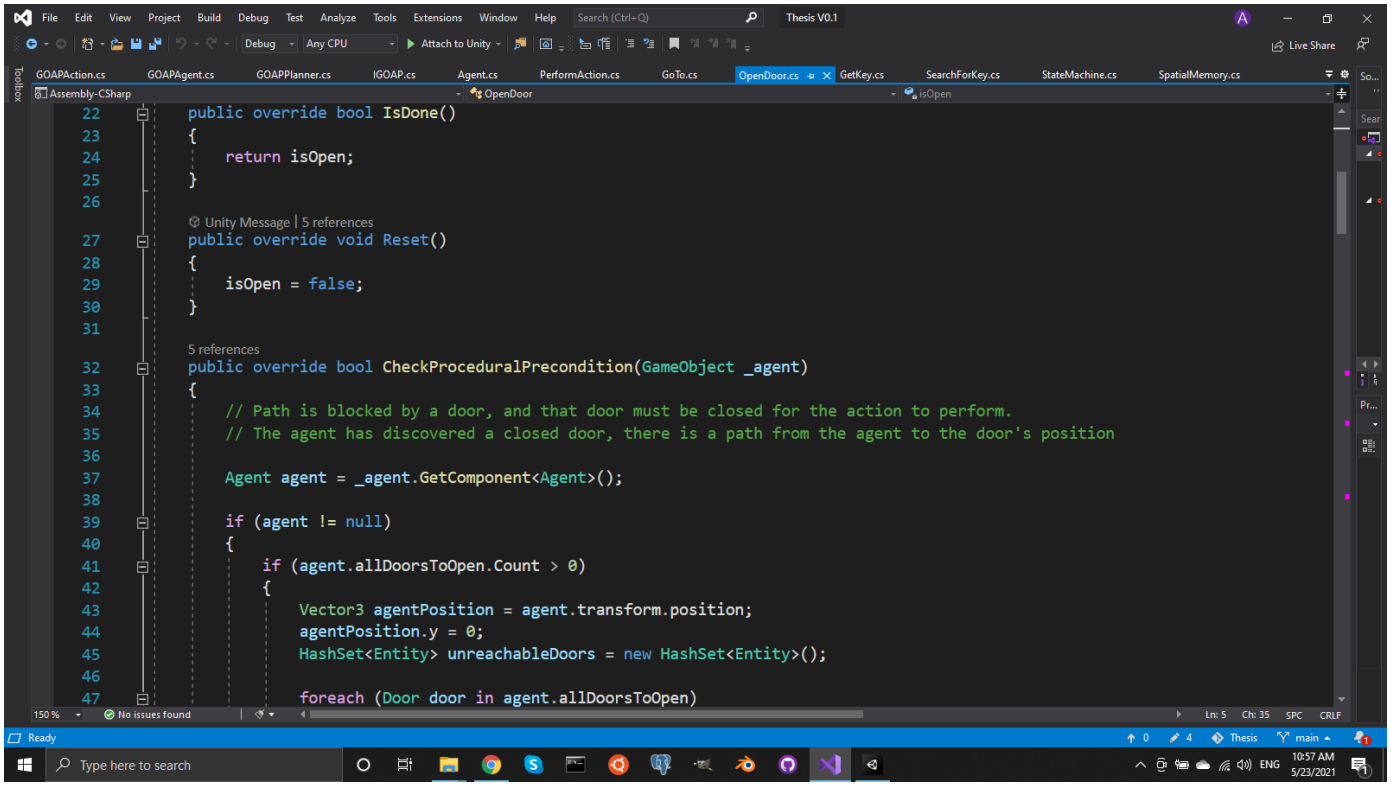
```
using UnityEngine;

Unity Script | 1 reference
public class GoTo : GOAPAction
{
    private bool arrivedAtGoToTarget = false;

    0 references
    public GoTo()
    {
        AddPrecondition("pathIsClear", true);
        AddPrecondition("arrivedAtGoToTarget", false);
        AddEffect("arrivedAtGoToTarget", true);
    }

    6 references
    public override bool RequiresInRange()
    {
        return true;
    }

    5 references
    public override bool IsDone()
    {
        return arrivedAtGoToTarget;
    }

    Unity Message | 5 references
    public override void Reset()
    {
        arrivedAtGoToTarget = false;
    }

    5 references
    public override bool CheckProceduralPrecondition(GameObject _agent)
    {
        Agent agent = _agent.GetComponent<Agent>();

        if (agent != null)
        {
            if (agent.goToTarget == null)
            {
                return false;
            }

            if (agent.UsablePath(agent.goToTarget.transform.position) != null)
            {
                target = agent.goToTarget.gameObject;
                return true;
            }
        }

        return false;
    }
```


File Edit View Project Build Debug Test Analyze Tools Extensions Window Help
Thesis V0.1
Debug
Any CPU
Attach to Unity
Live Share
GOAPAction.cs
GOAPAgent.cs
GOAPPlanner.cs
IGOAP.cs
Agent.cs
PerformAction.cs
GoTo.cs
EntityClassifier.cs
GridCustom.cs
Entity.cs
StateMachine.cs
SpatialMemory.cs
Assembly-CSharp
GoTo
arrivedAtGoToTarget
5 references
public override void AttemptToGetTarget(GameObject _agent) { }
5 references
public override bool Perform(GameObject _agent)
{
Agent agent = _agent.GetComponent<Agent>();
if (agent != null)
{
if (agent.ArrivedAt(target.transform.position))
{
arrivedAtGoToTarget = true;
print("Arrived at go to target!");
return true;
}
}
return false;
}
}
150 %
No issues found
Ln: 3 Ch: 31 SPC CRLF
Ready
Thesis
main
Type here to search
10:46 AM
5/23/2021

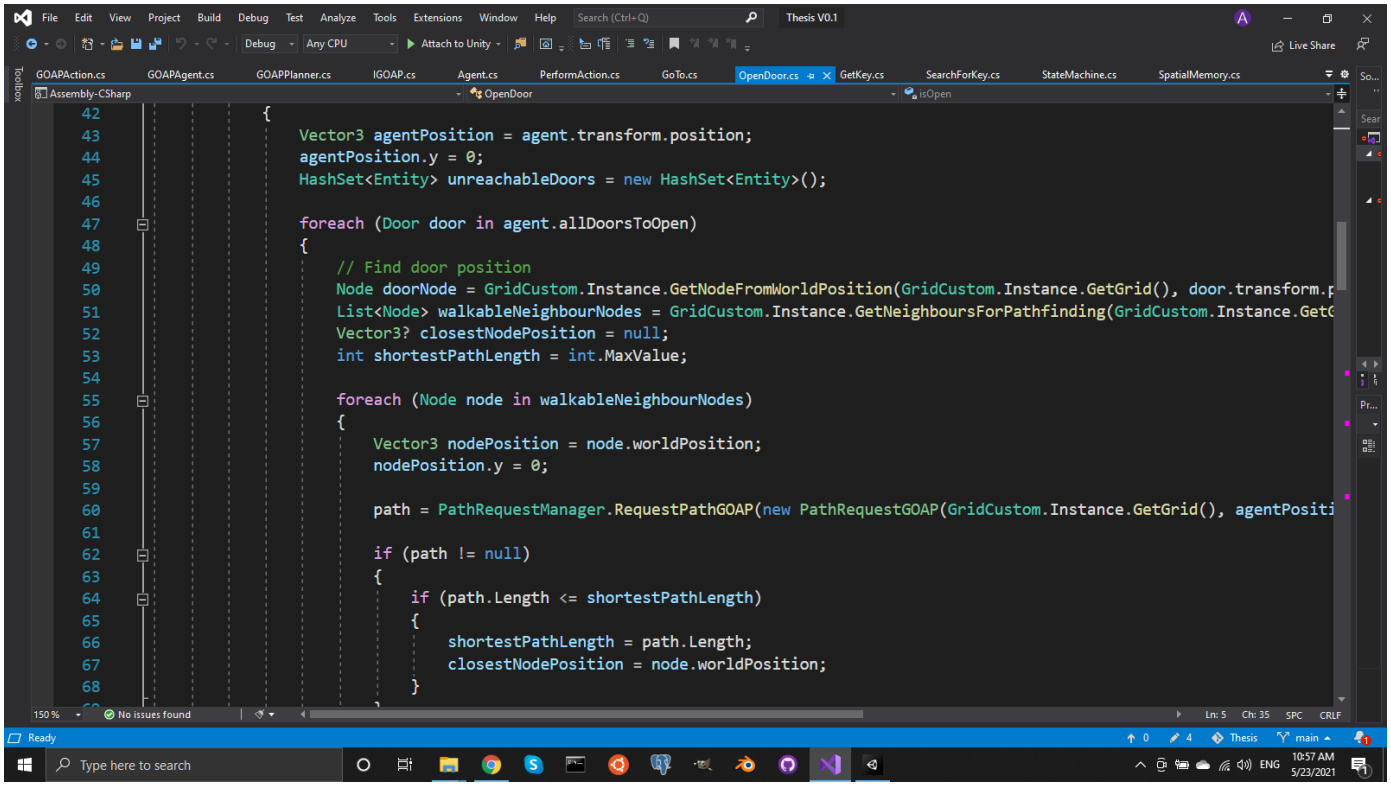
ΤΟ SCRIPT ΔΡΑΣΗΣ OpenDoor



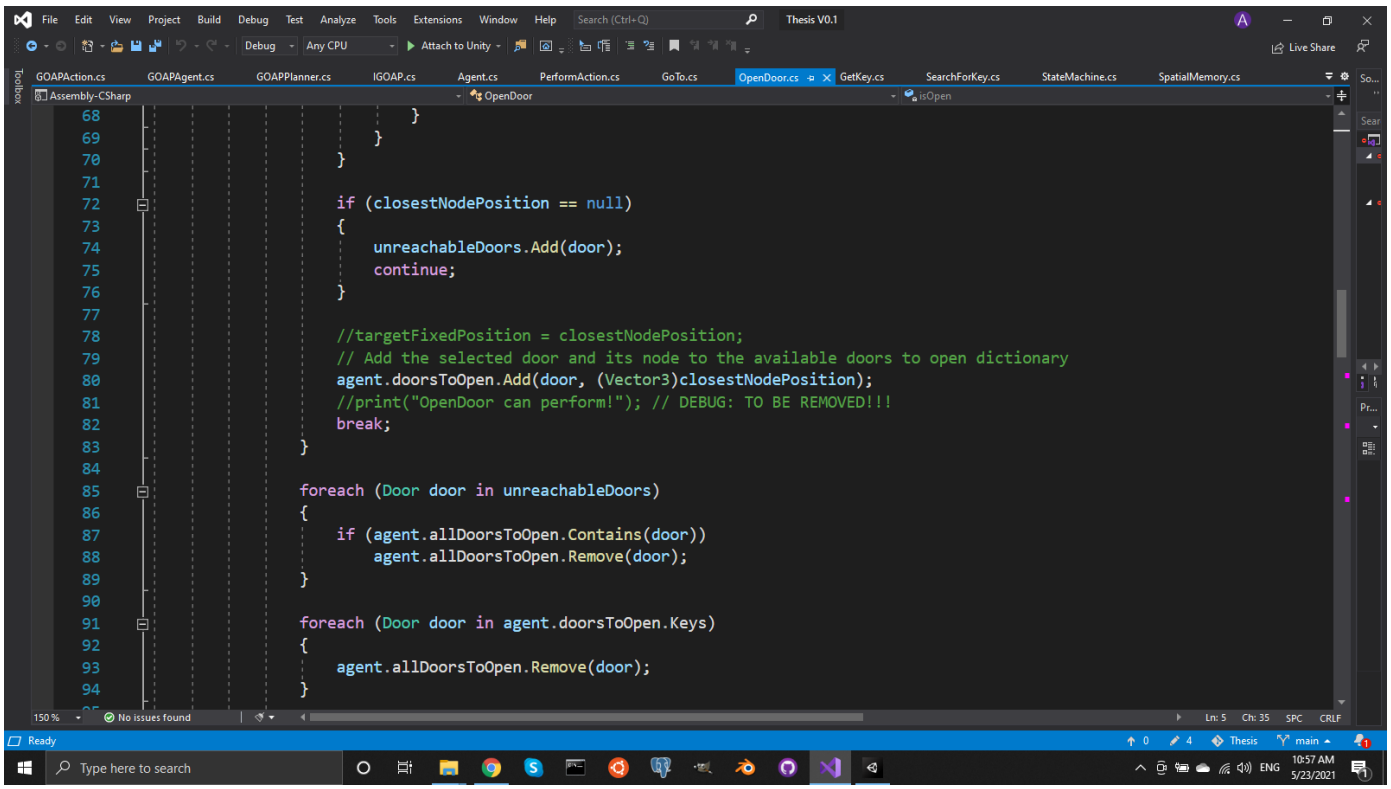
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

Unity Script | 1 reference
public class OpenDoor : GOAPAction
{
    private bool isOpen = false;
    Vector3[] path;

    0 references
    public OpenDoor()
    {
        AddPrecondition("hasKey", true);
        AddPrecondition("pathIsClear", false);
        AddEffect("pathIsClear", true);
    }

    6 references
    public override bool RequiresInRange()
    {
        return true;
    }

    5 references
    public override bool IsDone()
    {
        return isOpen;
    }
```



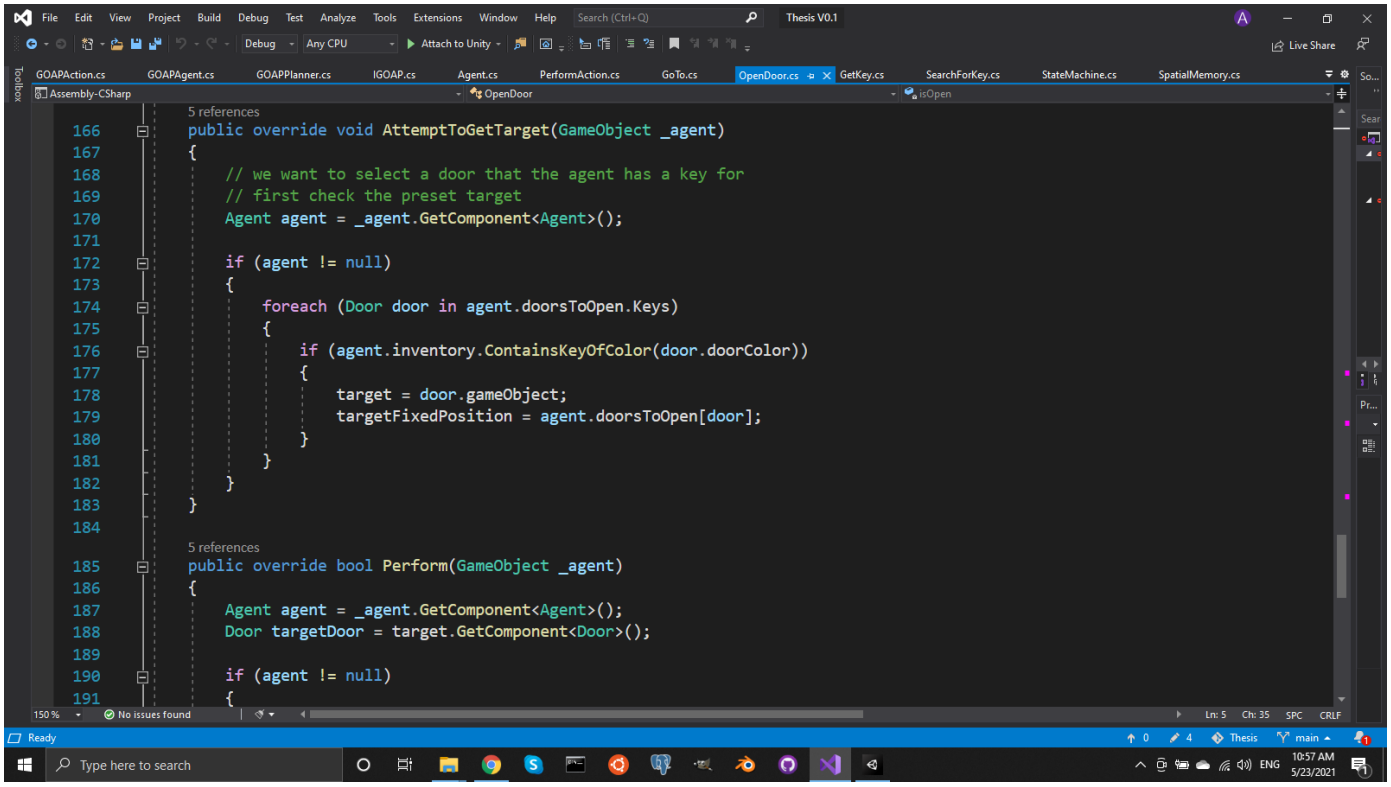
```
    public override bool IsDone()
    {
        return isOpen;
    }

    Unity Message | 5 references
    public override void Reset()
    {
        isOpen = false;
    }

    5 references
    public override bool CheckProceduralPrecondition(GameObject _agent)
    {
        // Path is blocked by a door, and that door must be closed for the action to perform.
        // The agent has discovered a closed door, there is a path from the agent to the door's position

        Agent agent = _agent.GetComponent<Agent>();

        if (agent != null)
        {
            if (agent.allDoorsToOpen.Count > 0)
            {
                Vector3 agentPosition = agent.transform.position;
                agentPosition.y = 0;
                HashSet<Entity> unreachableDoors = new HashSet<Entity>();

                foreach (Door door in agent.allDoorsToOpen)
```



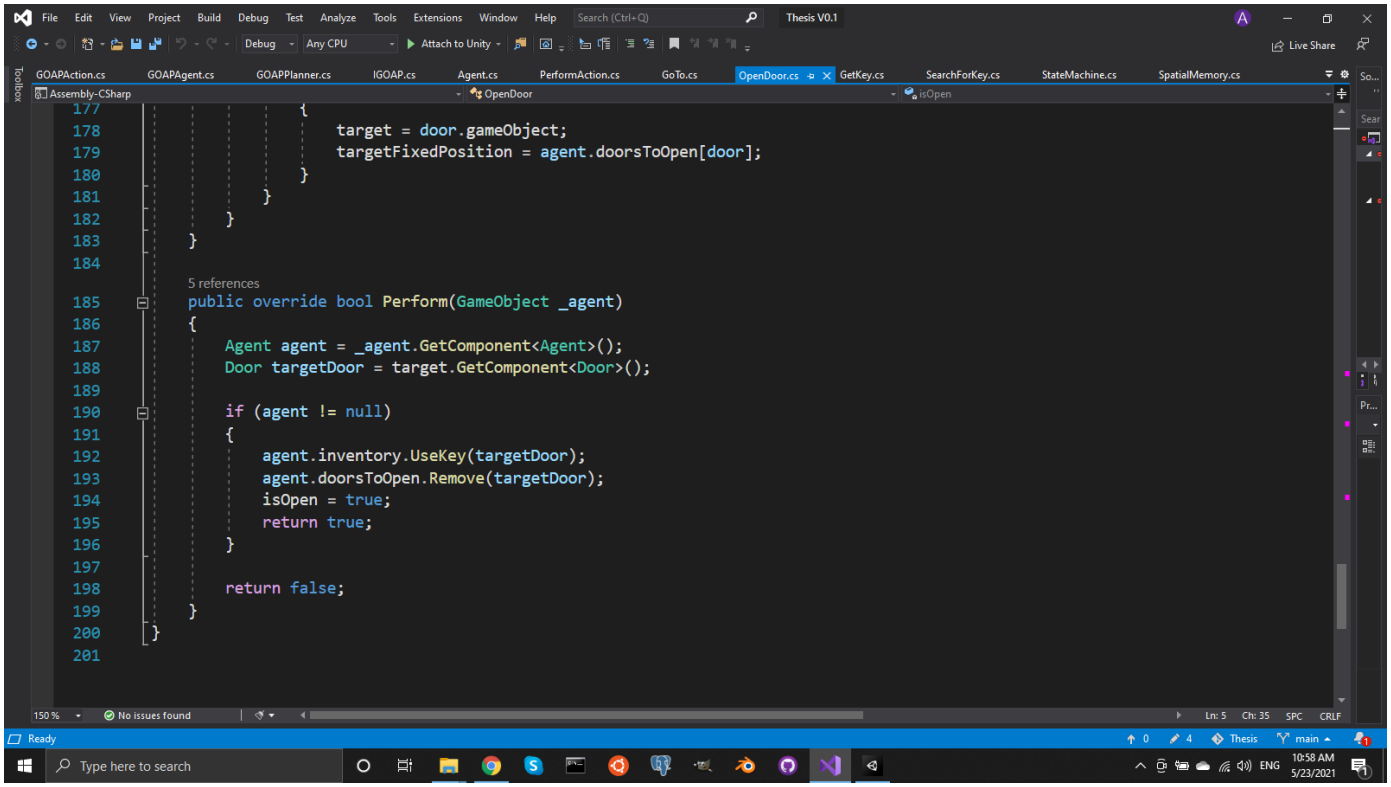
```
{
    Vector3 agentPosition = agent.transform.position;
    agentPosition.y = 0;
    HashSet<Entity> unreachableDoors = new HashSet<Entity>();

    foreach (Door door in agent.allDoorsToOpen)
    {
        // Find door position
        Node doorNode = GridCustom.Instance.GetNodeFromWorldPosition(GridCustom.Instance.GetGrid(), door.transform.p
        List<Node> walkableNeighbourNodes = GridCustom.Instance.GetNeighboursForPathfinding(GridCustom.Instance.GetG
        Vector3? closestNodePosition = null;
        int shortestPathLength = int.MaxValue;

        foreach (Node node in walkableNeighbourNodes)
        {
            Vector3 nodePosition = node.worldPosition;
            nodePosition.y = 0;

            path = PathRequestManager.RequestPathGOAP(new PathRequestGOAP(GridCustom.Instance.GetGrid(), agentPositi

            if (path != null)
            {
                if (path.Length <= shortestPathLength)
                {
                    shortestPathLength = path.Length;
                    closestNodePosition = node.worldPosition;
                }
```



```
                }
            }
        }

        if (closestNodePosition == null)
        {
            unreachableDoors.Add(door);
            continue;
        }

        //targetFixedPosition = closestNodePosition;
        // Add the selected door and its node to the available doors to open dictionary
        agent.doorsToOpen.Add(door, (Vector3)closestNodePosition);
        //print("OpenDoor can perform!"); // DEBUG: TO BE REMOVED!!!
        break;
    }

    foreach (Door door in unreachableDoors)
    {
        if (agent.allDoorsToOpen.Contains(door))
            agent.allDoorsToOpen.Remove(door);
    }

    foreach (Door door in agent.doorsToOpen.Keys)
    {
        agent.allDoorsToOpen.Remove(door);
    }
```

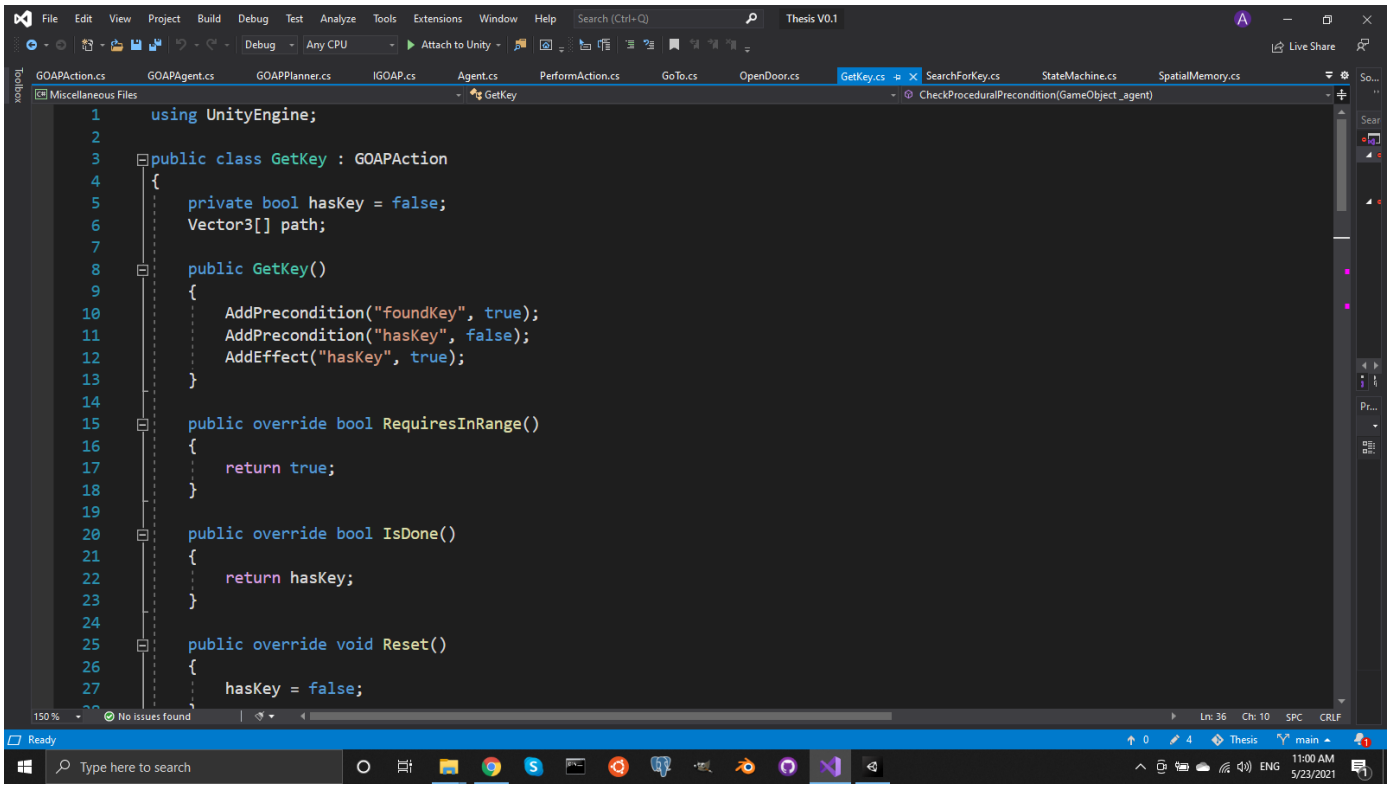


```
5 references
public override void AttemptToGetTarget(GameObject _agent)
{
    // we want to select a door that the agent has a key for
    // first check the preset target
    Agent agent = _agent.GetComponent<Agent>();

    if (agent != null)
    {
        foreach (Door door in agent.doorsToOpen.Keys)
        {
            if (agent.inventory.ContainsKeyOfColor(door.doorColor))
            {
                target = door.gameObject;
                targetFixedPosition = agent.doorsToOpen[door];
            }
        }
    }
}

5 references
public override bool Perform(GameObject _agent)
{
    Agent agent = _agent.GetComponent<Agent>();
    Door targetDoor = target.GetComponent<Door>();

    if (agent != null)
    {
```

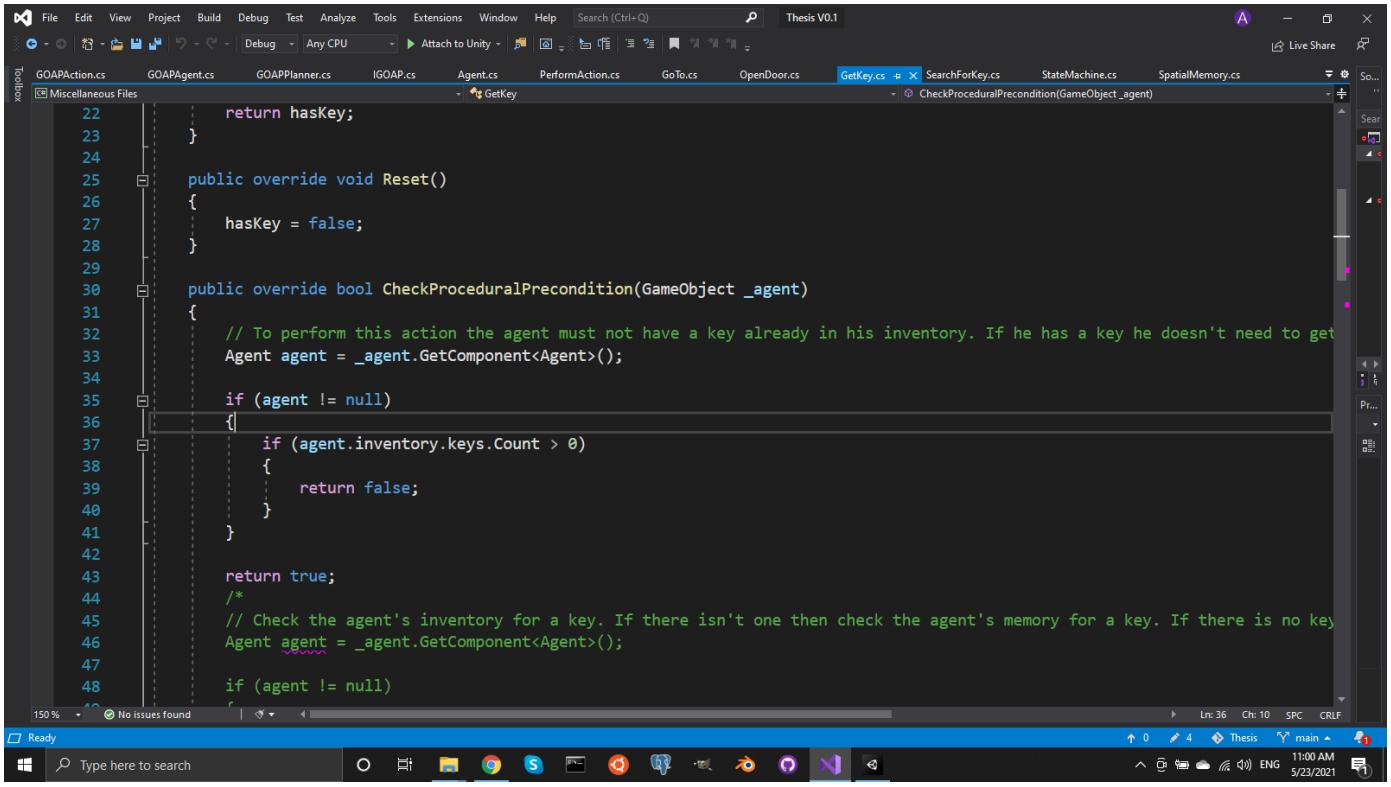


```
                target = door.gameObject;
                targetFixedPosition = agent.doorsToOpen[door];
            }
        }
    }
}

5 references
public override bool Perform(GameObject _agent)
{
    Agent agent = _agent.GetComponent<Agent>();
    Door targetDoor = target.GetComponent<Door>();

    if (agent != null)
    {
        agent.inventory.UseKey(targetDoor);
        agent.doorsToOpen.Remove(targetDoor);
        isOpen = true;
        return true;
    }

    return false;
}
}
```

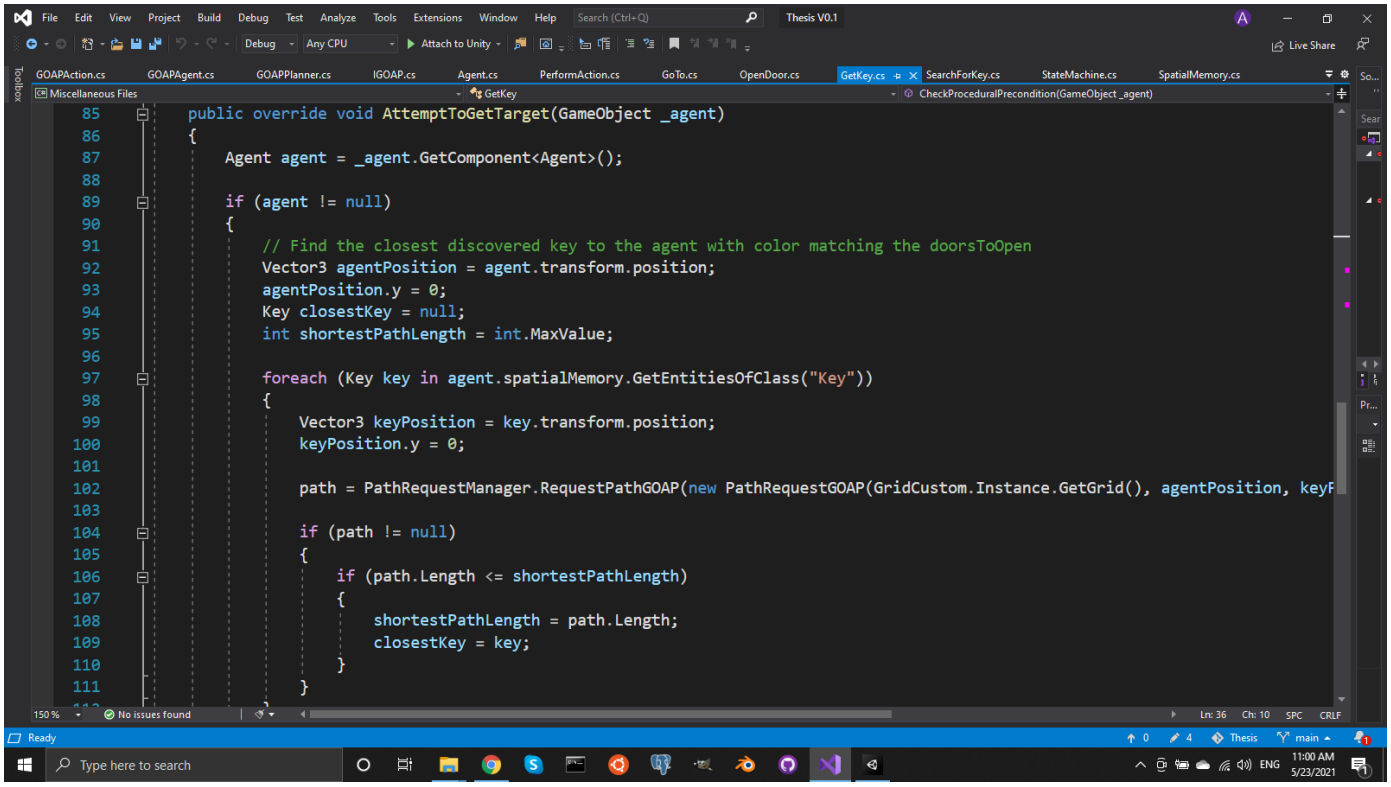
ΤΟ SCRIPT ΔΡΑΣΗΣ GetKey



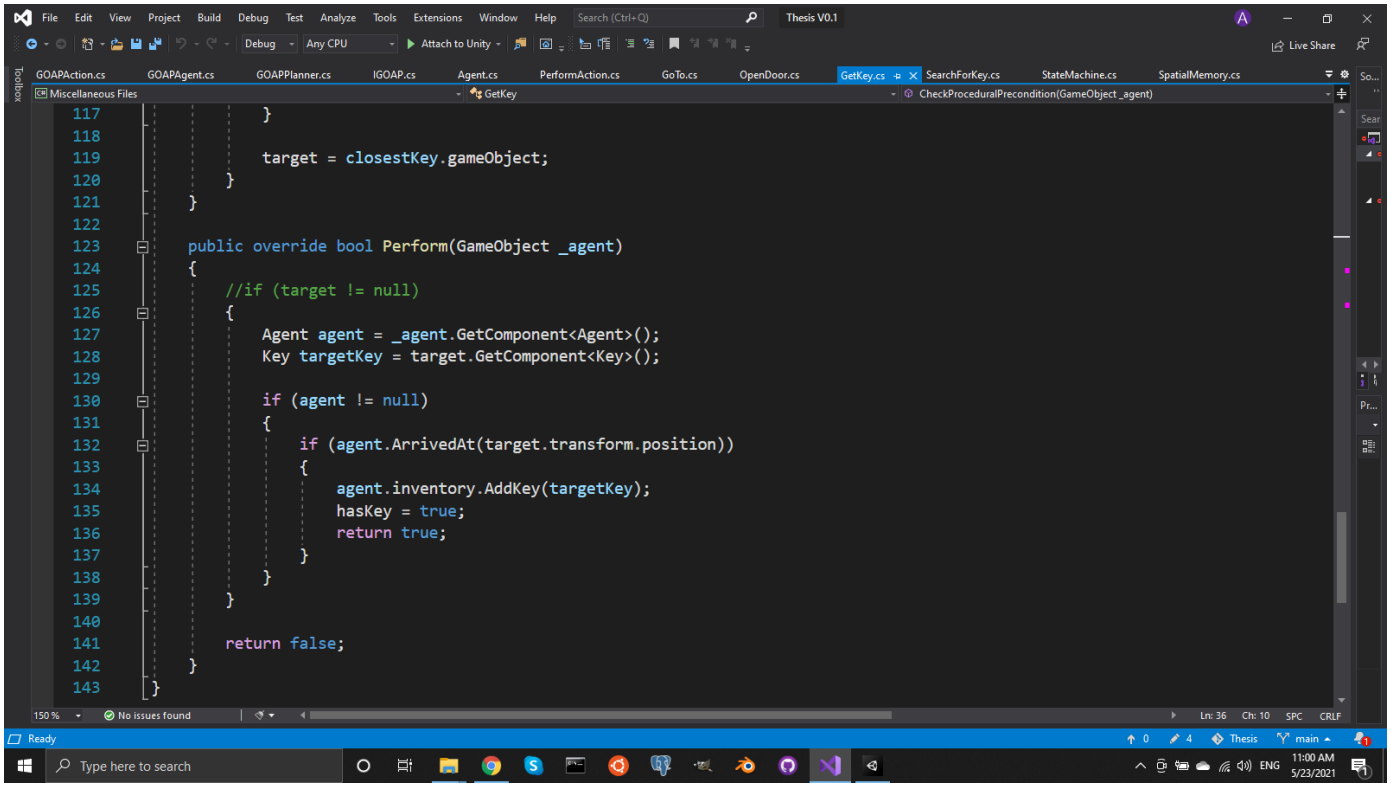
```
using UnityEngine;

public class GetKey : GOAPAction
{
    private bool hasKey = false;
    Vector3[] path;

    public GetKey()
    {
        AddPrecondition("foundKey", true);
        AddPrecondition("hasKey", false);
        AddEffect("hasKey", true);
    }

    public override bool RequiresInRange()
    {
        return true;
    }

    public override bool IsDone()
    {
        return hasKey;
    }

    public override void Reset()
    {
        hasKey = false;
```



```
        return hasKey;
    }

    public override void Reset()
    {
        hasKey = false;
    }

    public override bool CheckProceduralPrecondition(GameObject _agent)
    {
        // To perform this action the agent must not have a key already in his inventory. If he has a key he doesn't need to get
        Agent agent = _agent.GetComponent<Agent>();

        if (agent != null)
        {
            if (agent.inventory.keys.Count > 0)
            {
                return false;
            }
        }

        return true;
        /*
        // Check the agent's inventory for a key. If there isn't one then check the agent's memory for a key. If there is no key
        Agent agent = _agent.GetComponent<Agent>();

        if (agent != null)
```

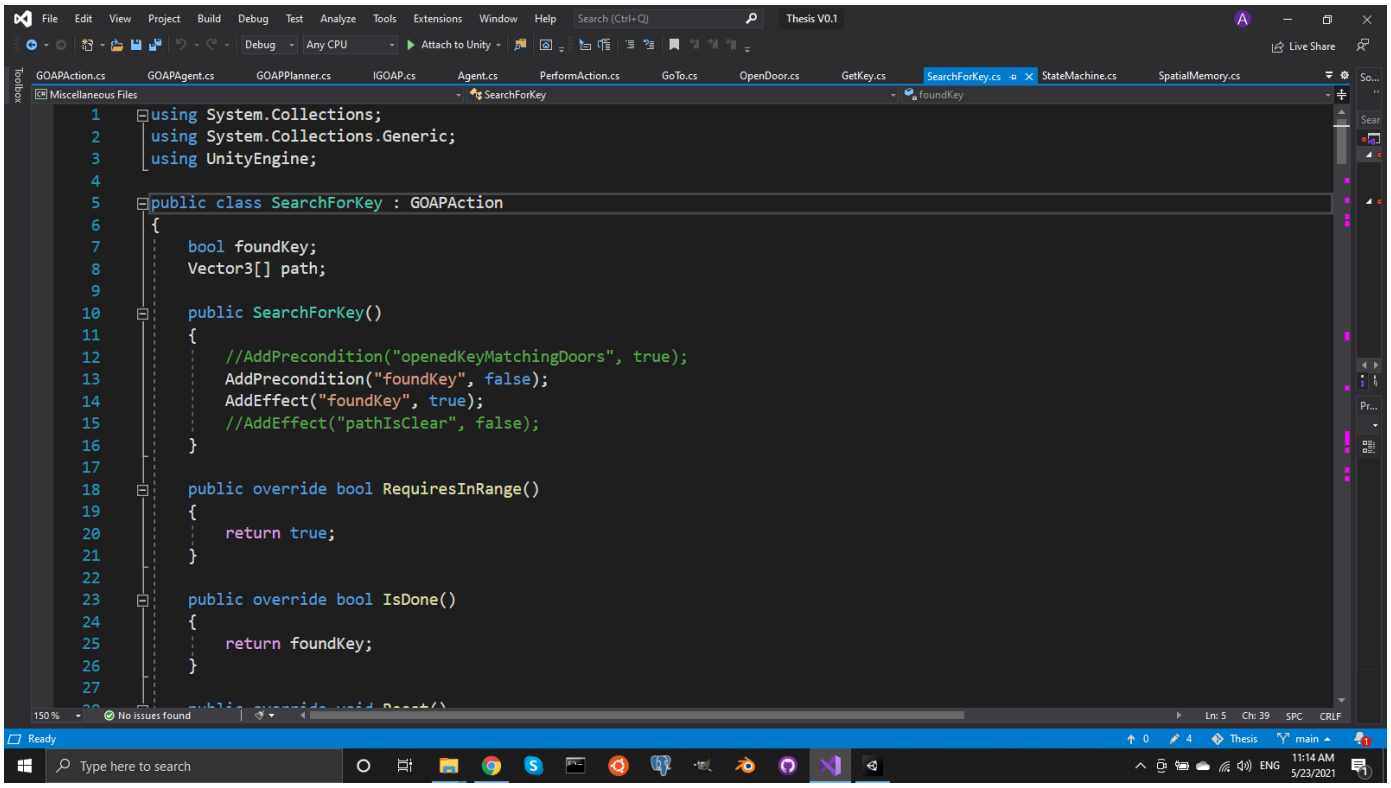


```
85      public override void AttemptToGetTarget(GameObject _agent)
86      {
87          Agent agent = _agent.GetComponent<Agent>();
88
89          if (agent != null)
90          {
91              // Find the closest discovered key to the agent with color matching the doorsToOpen
92              Vector3 agentPosition = agent.transform.position;
93              agentPosition.y = 0;
94              Key closestKey = null;
95              int shortestPathLength = int.MaxValue;
96
97              foreach (Key key in agent.spatialMemory.GetEntitiesOfClass("Key"))
98              {
99                  Vector3 keyPosition = key.transform.position;
100                 keyPosition.y = 0;
101
102                 path = PathRequestManager.RequestPathGOAP(new PathRequestGOAP(GridCustom.Instance.GetGrid(), agentPosition, keyF
103
104                 if (path != null)
105                 {
106                     if (path.Length <= shortestPathLength)
107                     {
108                         shortestPathLength = path.Length;
109                         closestKey = key;
110                     }
111                 }
```

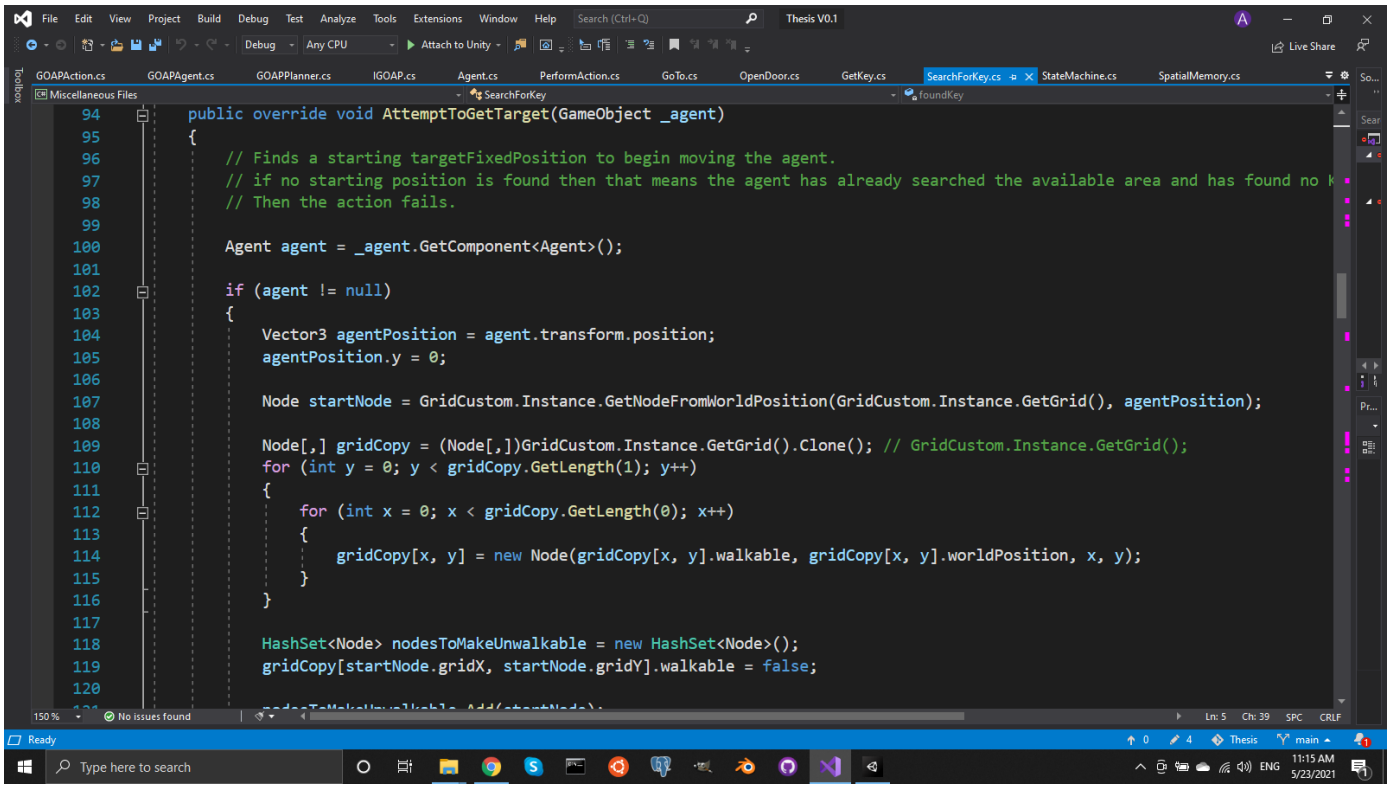


```
117             }
118
119             target = closestKey.gameObject;
120         }
121     }
122
123     public override bool Perform(GameObject _agent)
124     {
125         //if (target != null)
126         {
127             Agent agent = _agent.GetComponent<Agent>();
128             Key targetKey = target.GetComponent<Key>();
129
130             if (agent != null)
131             {
132                 if (agent.ArrivedAt(target.transform.position))
133                 {
134                     agent.inventory.AddKey(targetKey);
135                     hasKey = true;
136                     return true;
137                 }
138             }
139         }
140
141         return false;
142     }
143 }
```

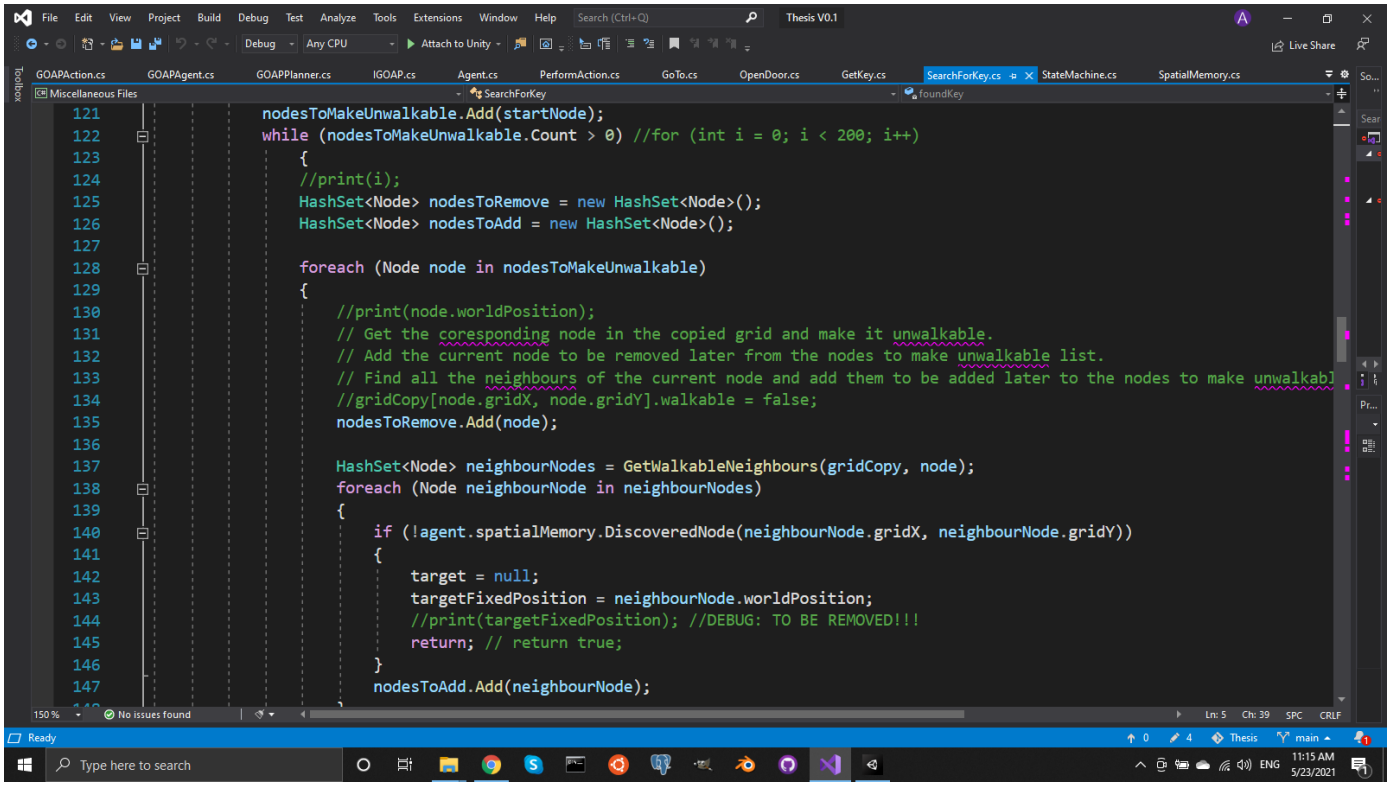
ΤΟ SCRIPT ΔΡΑΣΗΣ SearchForKey



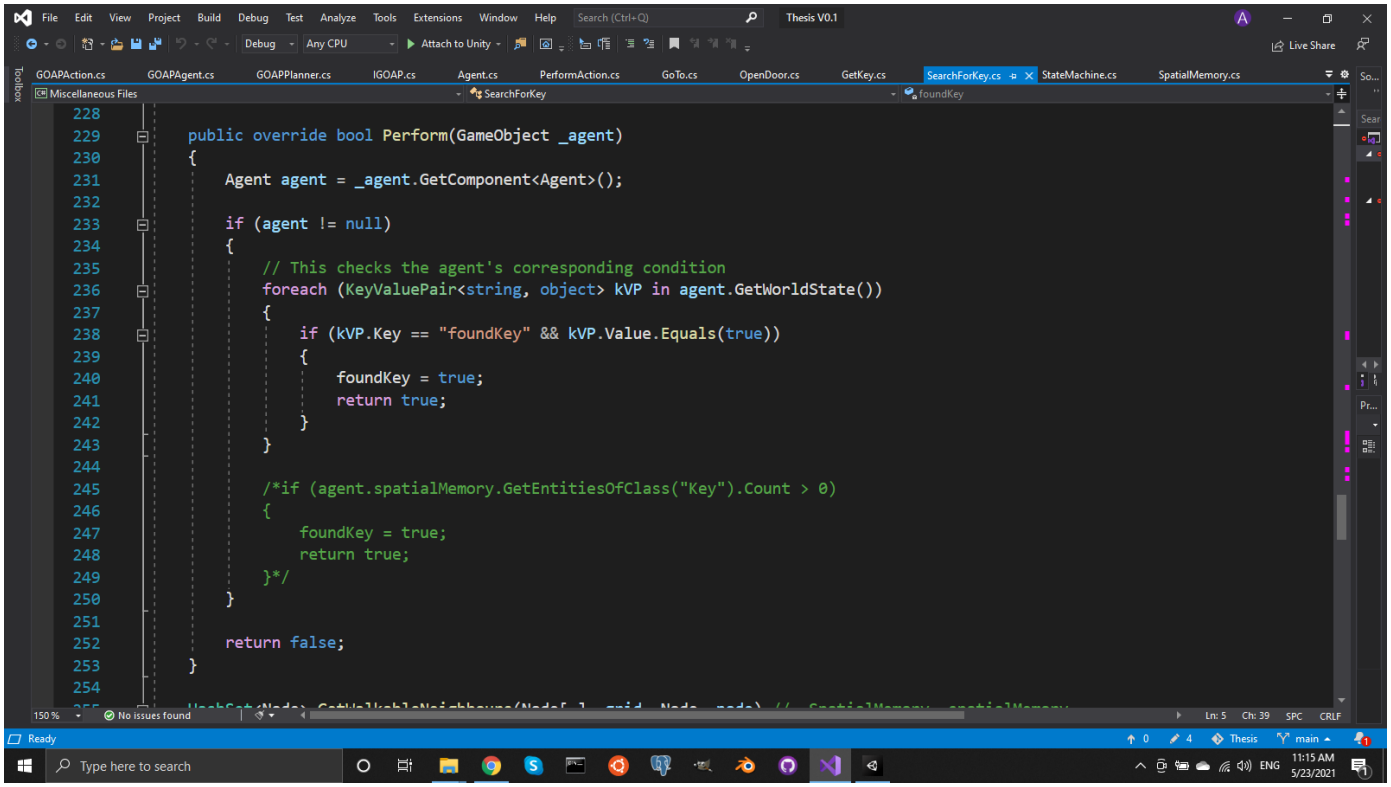
```
using System.Collections;
using System.Collections.Generic;
using UnityEngine;

public class SearchForKey : GOAPAction
{
    bool foundKey;
    Vector3[] path;

    public SearchForKey()
    {
        //AddPrecondition("openedKeyMatchingDoors", true);
        AddPrecondition("foundKey", false);
        AddEffect("foundKey", true);
        //AddEffect("pathIsClear", false);
    }

    public override bool RequiresInRange()
    {
        return true;
    }

    public override bool IsDone()
    {
        return foundKey;
    }
```



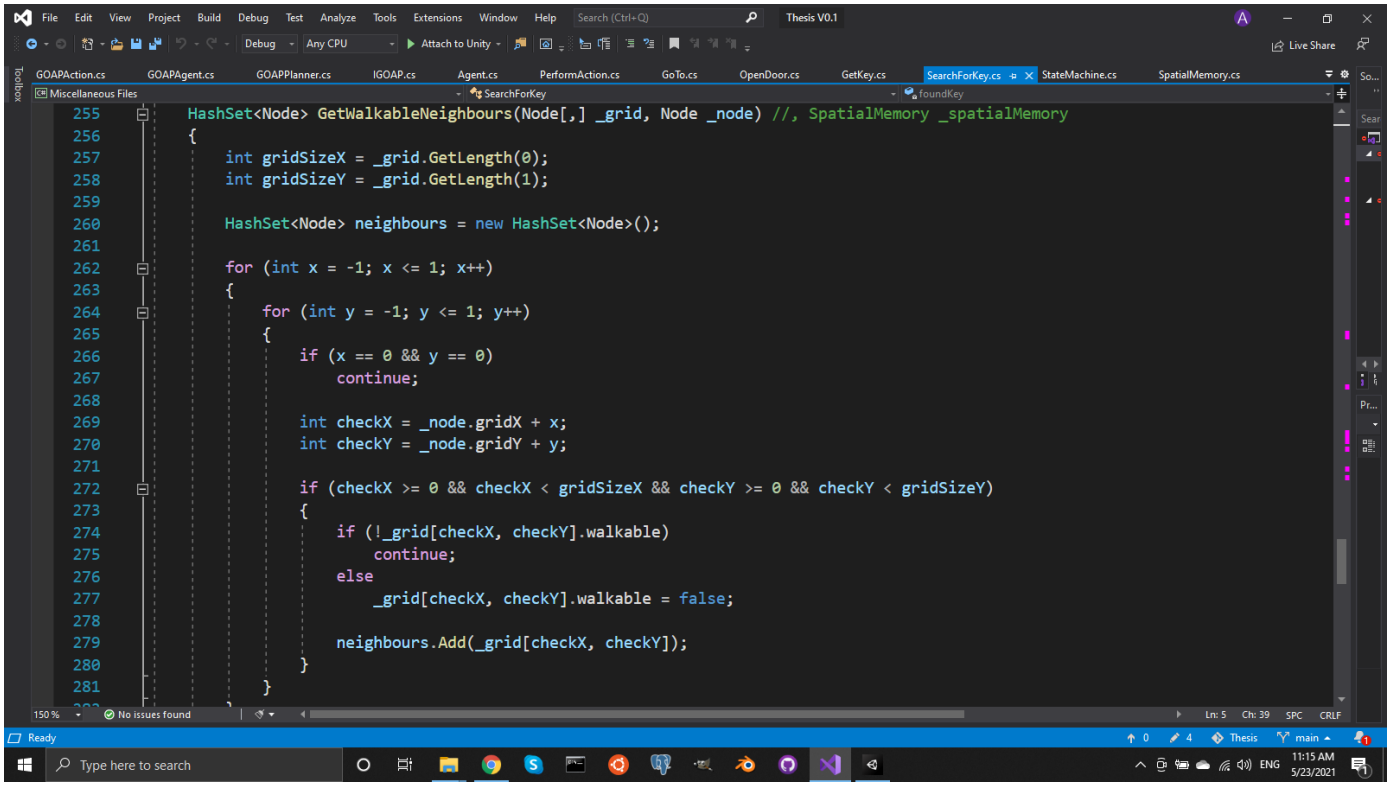
```
    public override void AttemptToGetTarget(GameObject _agent)
    {
        // Finds a starting targetFixedPosition to begin moving the agent.
        // if no starting position is found then that means the agent has already searched the available area and has found no k
        // Then the action fails.

        Agent agent = _agent.GetComponent<Agent>();

        if (agent != null)
        {
            Vector3 agentPosition = agent.transform.position;
            agentPosition.y = 0;

            Node startNode = GridCustom.Instance.GetNodeFromWorldPosition(GridCustom.Instance.GetGrid(), agentPosition);

            Node[,] gridCopy = (Node[,])GridCustom.Instance.GetGrid().Clone(); // GridCustom.Instance.GetGrid();
            for (int y = 0; y < gridCopy.GetLength(1); y++)
            {
                for (int x = 0; x < gridCopy.GetLength(0); x++)
                {
                    gridCopy[x, y] = new Node(gridCopy[x, y].walkable, gridCopy[x, y].worldPosition, x, y);
                }
            }

            HashSet<Node> nodesToMakeUnwalkable = new HashSet<Node>();
            gridCopy[startNode.gridX, startNode.gridY].walkable = false;
```



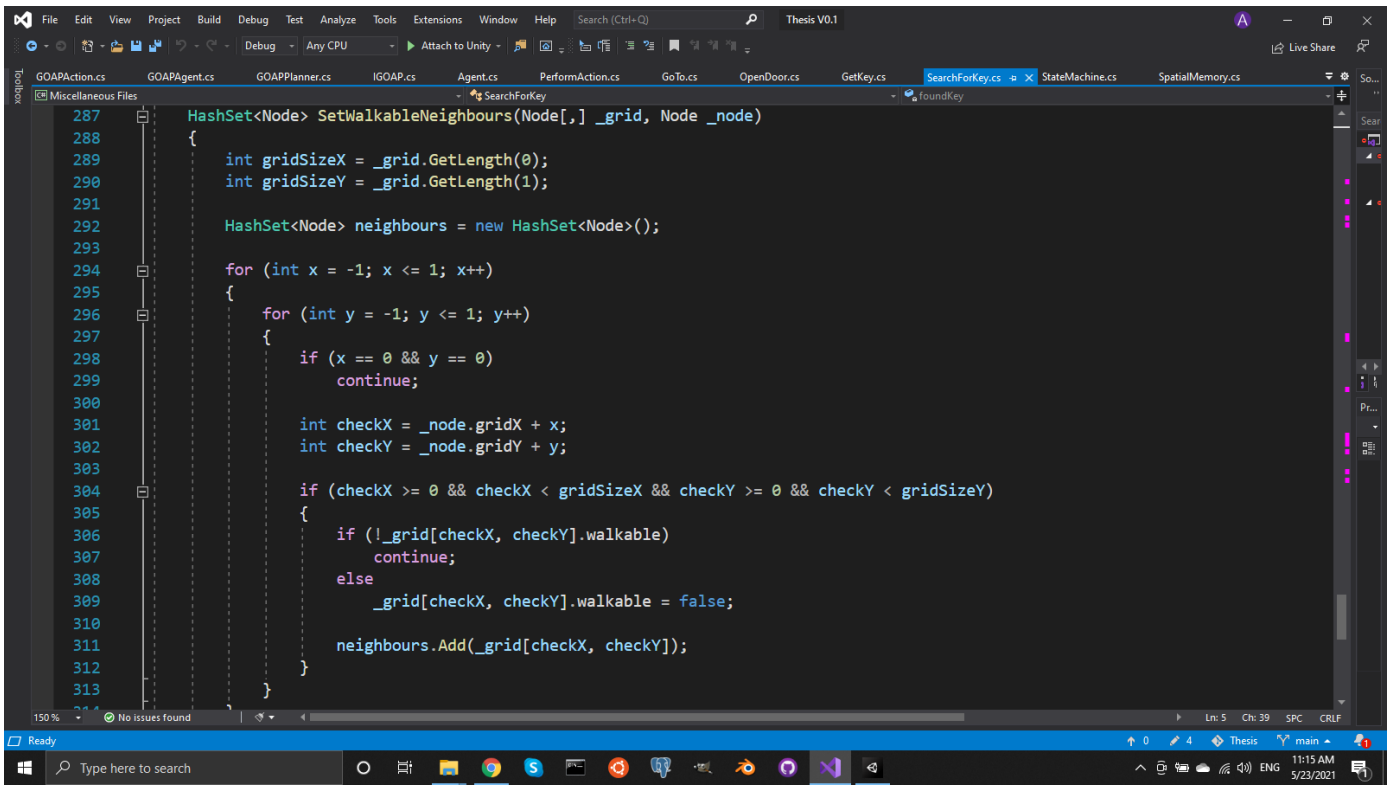
```
121             nodesToMakeUnwalkable.Add(startNode);
122             while (nodesToMakeUnwalkable.Count > 0) //for (int i = 0; i < 200; i++)
123                 {
124                 //print(i);
125                 HashSet<Node> nodesToRemove = new HashSet<Node>();
126                 HashSet<Node> nodesToAdd = new HashSet<Node>();
127
128                 foreach (Node node in nodesToMakeUnwalkable)
129                 {
130                     //print(node.worldPosition);
131                     // Get the coresponding node in the copied grid and make it unwalkable.
132                     // Add the current node to be removed later from the nodes to make unwalkable list.
133                     // Find all the neighbours of the current node and add them to be added later to the nodes to make unwalkabl
134                     //gridCopy[node.gridX, node.gridY].walkable = false;
135                     nodesToRemove.Add(node);
136
137                     HashSet<Node> neighbourNodes = GetWalkableNeighbours(gridCopy, node);
138                     foreach (Node neighbourNode in neighbourNodes)
139                     {
140                         if (!agent.spatialMemory.DiscoveredNode(neighbourNode.gridX, neighbourNode.gridY))
141                         {
142                             target = null;
143                             targetFixedPosition = neighbourNode.worldPosition;
144                             //print(targetFixedPosition); //DEBUG: TO BE REMOVED!!!
145                             return; // return true;
146                         }
147                         nodesToAdd.Add(neighbourNode);
```



```
228
229     public override bool Perform(GameObject _agent)
230     {
231         Agent agent = _agent.GetComponent<Agent>();
232
233         if (agent != null)
234         {
235             // This checks the agent's corresponding condition
236             foreach (KeyValuePair<string, object> kVP in agent.GetWorldState())
237             {
238                 if (kVP.Key == "foundKey" && kVP.Value.Equals(true))
239                 {
240                     foundKey = true;
241                     return true;
242                 }
243             }
244
245             /*if (agent.spatialMemory.GetEntitiesOfClass("Key").Count > 0)
246             {
247                 foundKey = true;
248                 return true;
249             }*/
250         }
251
252         return false;
253     }
254
```



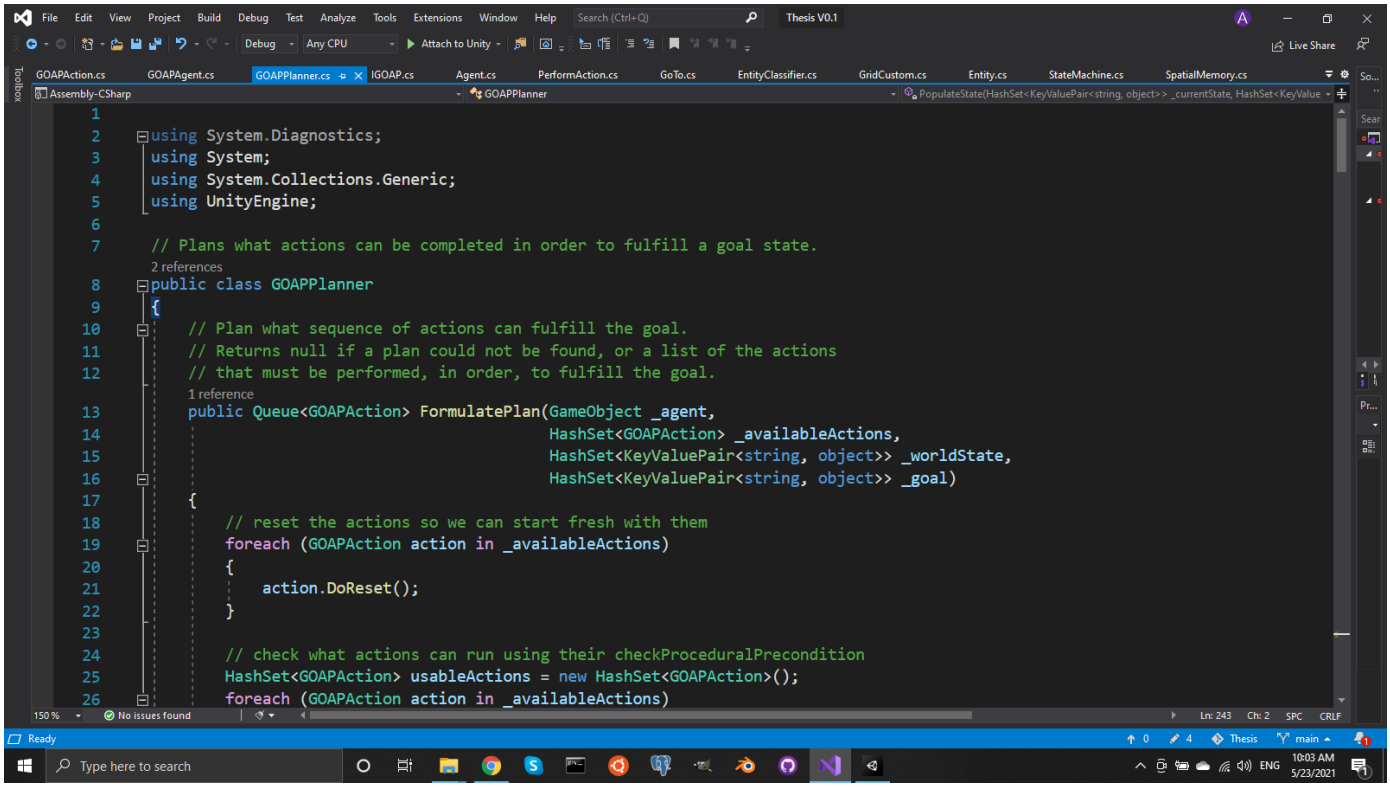
```
HashSet<Node> GetWalkableNeighbours(Node[,] _grid, Node _node) //, SpatialMemory _spatialMemory
{
    int gridSizeX = _grid.GetLength(0);
    int gridSizeY = _grid.GetLength(1);

    HashSet<Node> neighbours = new HashSet<Node>();

    for (int x = -1; x <= 1; x++)
    {
        for (int y = -1; y <= 1; y++)
        {
            if (x == 0 && y == 0)
                continue;

            int checkX = _node.gridX + x;
            int checkY = _node.gridY + y;

            if (checkX >= 0 && checkX < gridSizeX && checkY >= 0 && checkY < gridSizeY)
            {
                if (!_grid[checkX, checkY].walkable)
                    continue;
                else
                    _grid[checkX, checkY].walkable = false;

                neighbours.Add(_grid[checkX, checkY]);
            }
        }
```

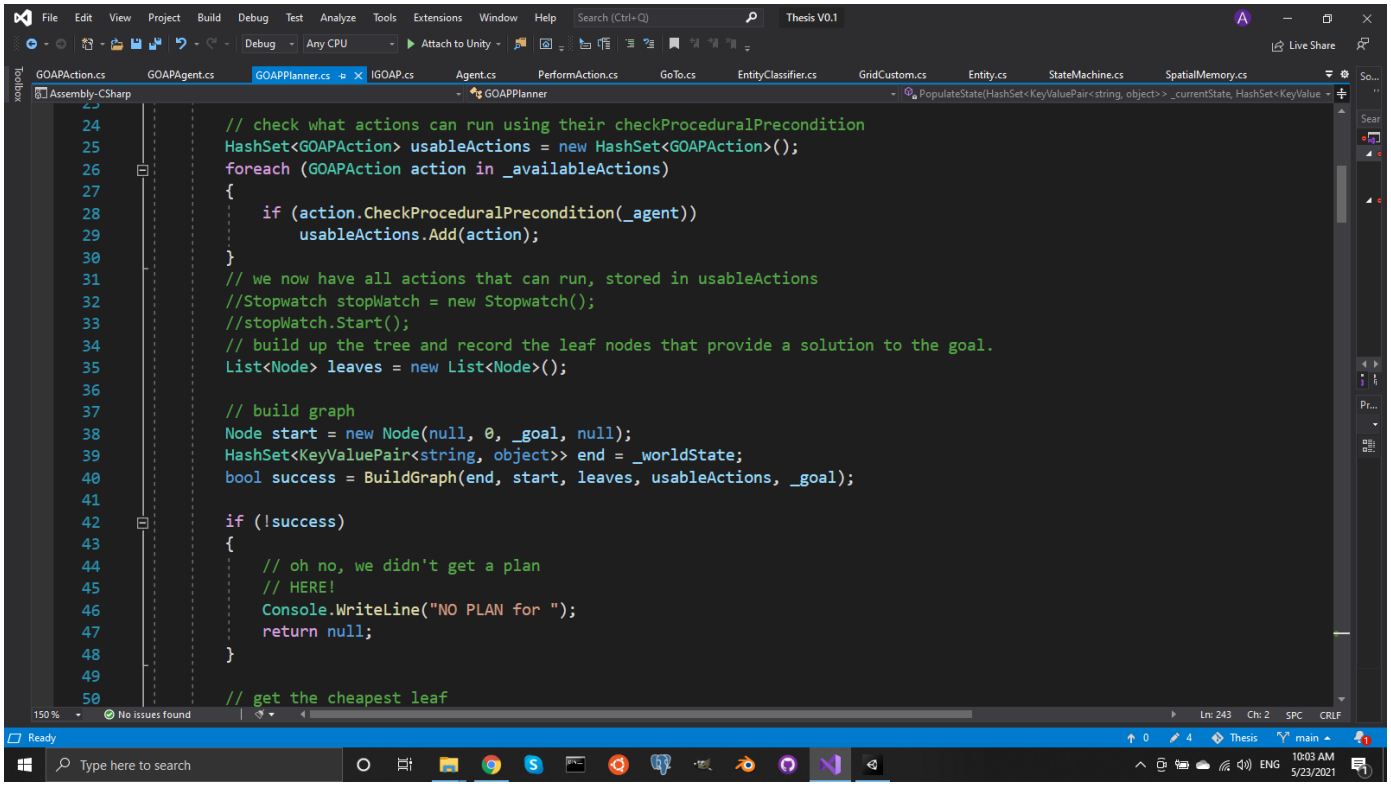


```
HashSet<Node> SetWalkableNeighbours(Node[,] _grid, Node _node)
{
    int gridSizeX = _grid.GetLength(0);
    int gridSizeY = _grid.GetLength(1);

    HashSet<Node> neighbours = new HashSet<Node>();

    for (int x = -1; x <= 1; x++)
    {
        for (int y = -1; y <= 1; y++)
        {
            if (x == 0 && y == 0)
                continue;

            int checkX = _node.gridX + x;
            int checkY = _node.gridY + y;

            if (checkX >= 0 && checkX < gridSizeX && checkY >= 0 && checkY < gridSizeY)
            {
                if (!_grid[checkX, checkY].walkable)
                    continue;
                else
                    _grid[checkX, checkY].walkable = false;

                neighbours.Add(_grid[checkX, checkY]);
            }
        }
```

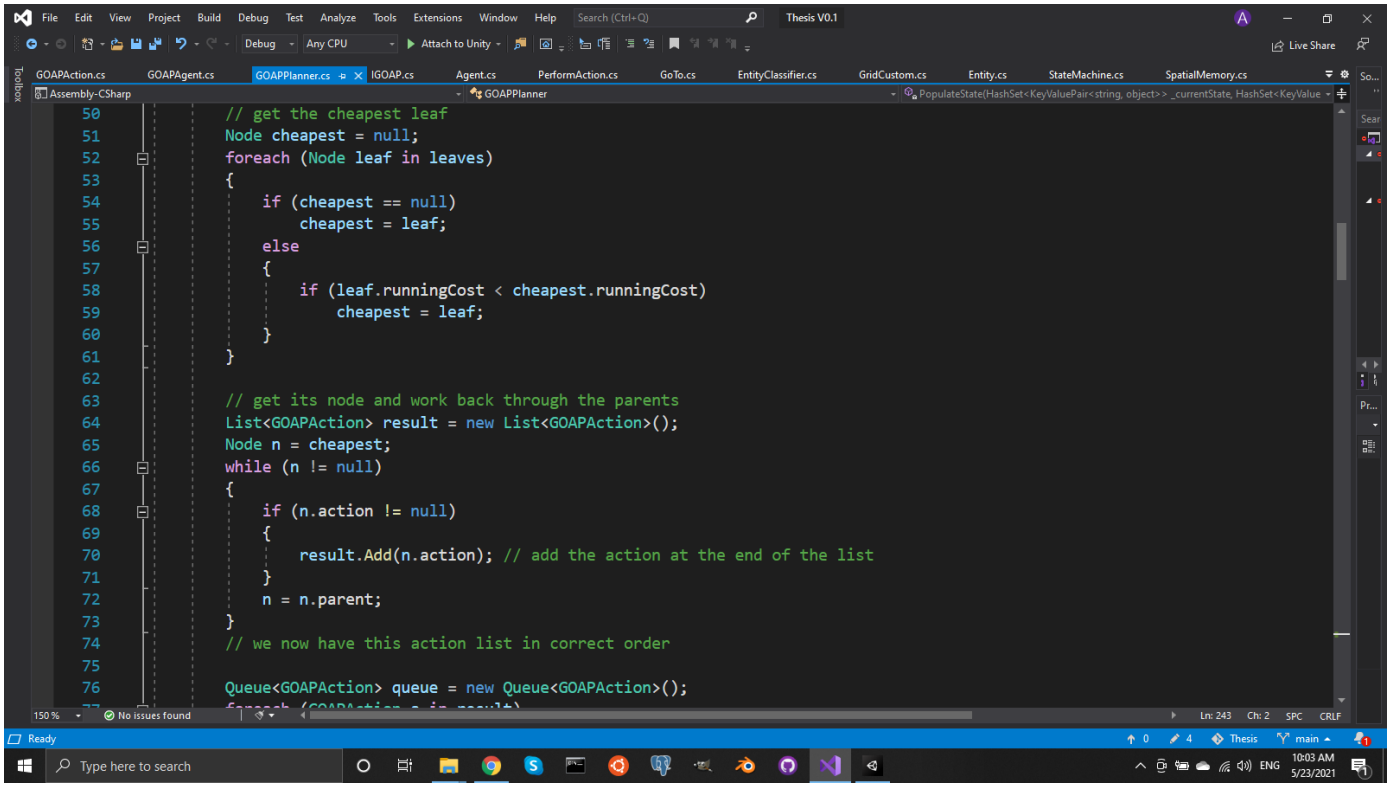
ΔΗΜΙΟΥΡΓΙΑ ΕΝΟΣ ΣΧΕΔΙΟΥ (GOAPPlanner)

Το **GOAPPlanner** είναι ένα script που εξετάζει τις προϋποθέσεις και τα αποτελέσματα των δράσεων και δημιουργεί ουρές (queues) δράσεων που θα επιτύχουν έναν στόχο, δηλαδή ένα σχέδιο. Αυτός ο στόχος παρέχεται από τον πράκτορα, μαζί με μία κατάσταση του κόσμου και μια λίστα δράσεων που μπορεί να εκτελέσει ο πράκτορας. Με αυτές τις πληροφορίες, το **GOAPPlanner** μπορεί να εξετάσει τις δράσεις, να δει ποιές μπορούν να εκτελεστούν και ποιές δεν μπορούν, και στη συνέχεια να αποφασίσει ποιά ακολουθία δράσεων είναι η καλύτερη.

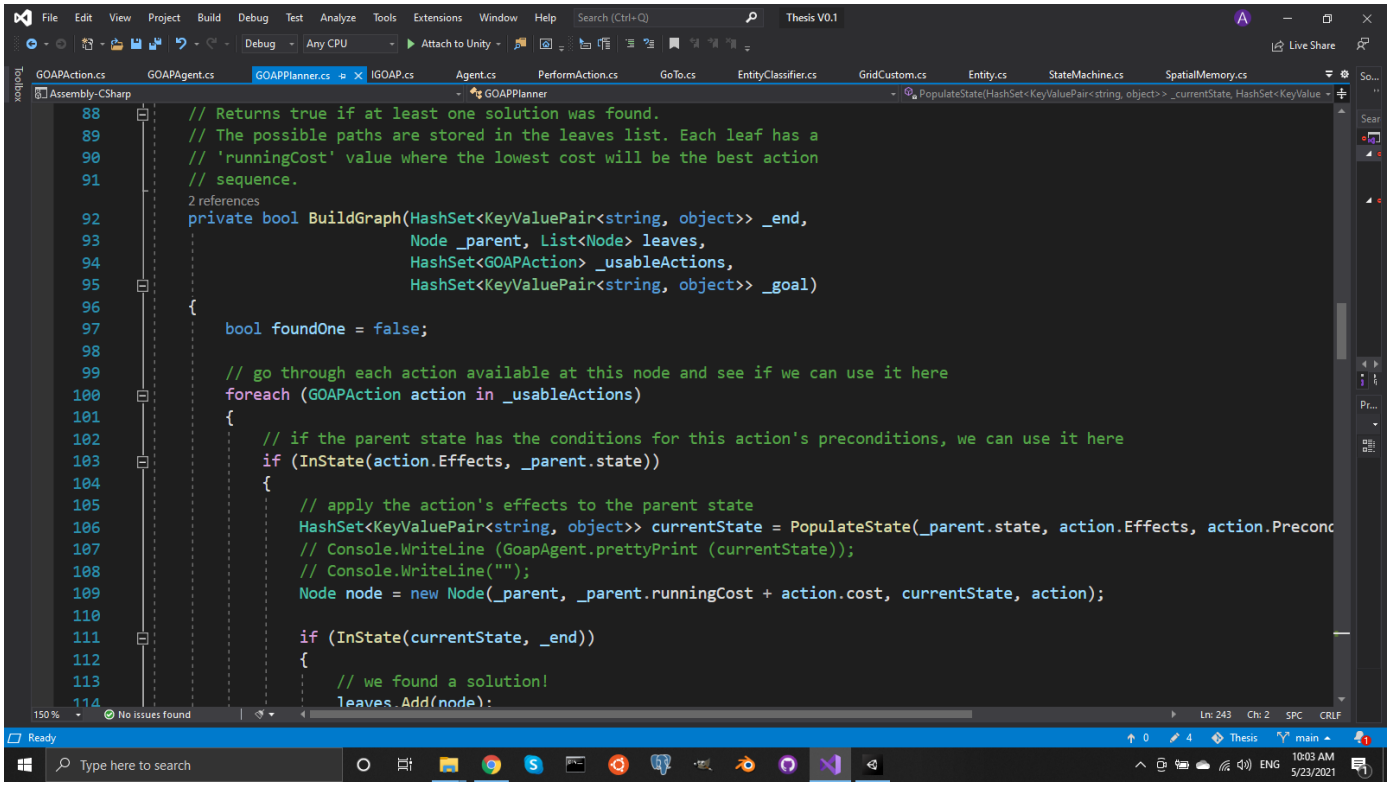




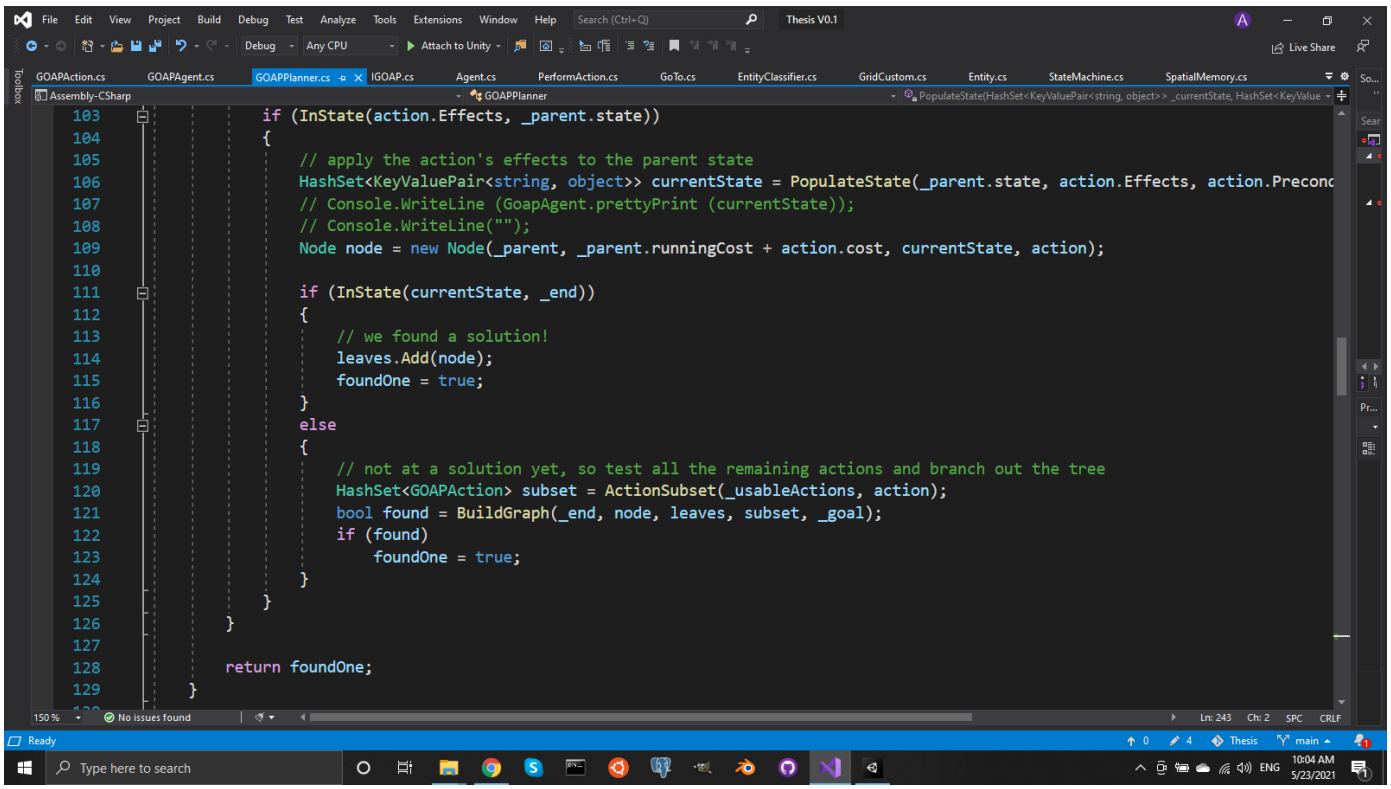
```
// check what actions can run using their checkProceduralPrecondition
HashSet<GOAPAction> usableActions = new HashSet<GOAPAction>();
foreach (GOAPAction action in _availableActions)
{
    if (action.CheckProceduralPrecondition(_agent))
        usableActions.Add(action);
}
// we now have all actions that can run, stored in usableActions
//Stopwatch stopWatch = new Stopwatch();
//stopWatch.Start();
// build up the tree and record the leaf nodes that provide a solution to the goal.
List<Node> leaves = new List<Node>();

// build graph
Node start = new Node(null, 0, _goal, null);
HashSet<KeyValuePair<string, object>> end = _worldState;
bool success = BuildGraph(end, start, leaves, usableActions, _goal);

if (!success)
{
    // oh no, we didn't get a plan
    // HERE!
    Console.WriteLine("NO PLAN for ");
    return null;
}

// get the cheapest leaf
```



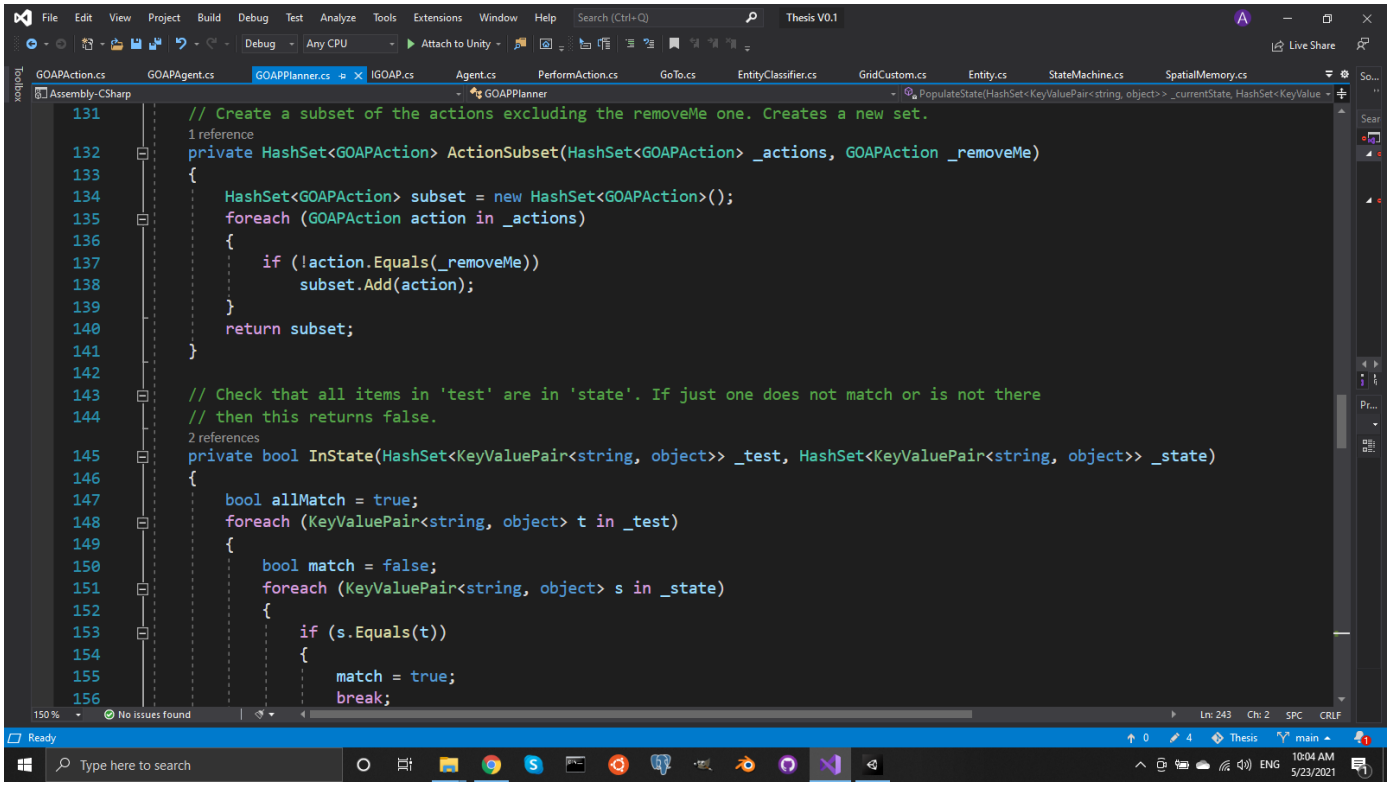
```
// get the cheapest leaf
Node cheapest = null;
foreach (Node leaf in leaves)
{
    if (cheapest == null)
        cheapest = leaf;
    else
    {
        if (leaf.runningCost < cheapest.runningCost)
            cheapest = leaf;
    }
}

// get its node and work back through the parents
List<GOAPAction> result = new List<GOAPAction>();
Node n = cheapest;
while (n != null)
{
    if (n.action != null)
    {
        result.Add(n.action); // add the action at the end of the list
    }
    n = n.parent;
}
// we now have this action list in correct order

Queue<GOAPAction> queue = new Queue<GOAPAction>();
```



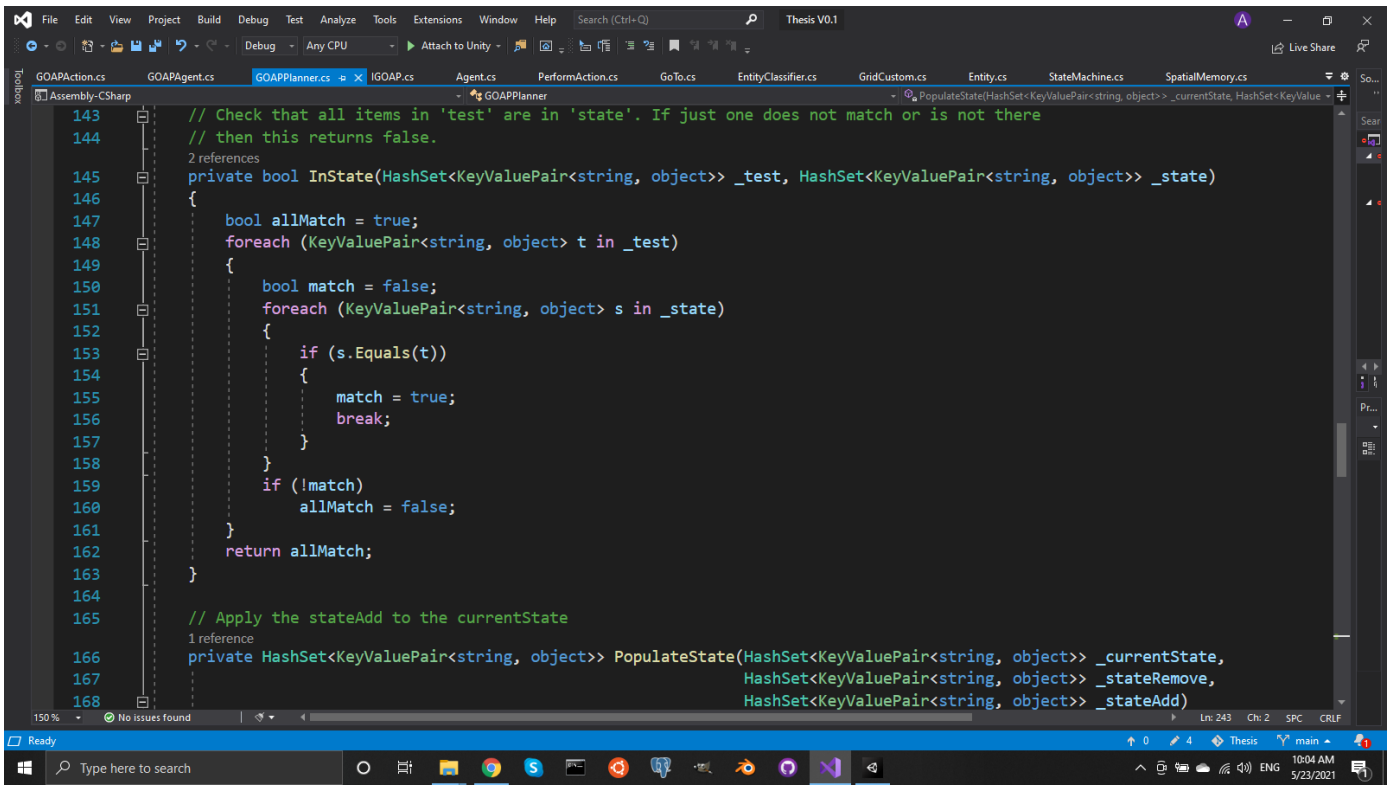
```
88      // Returns true if at least one solution was found.
89      // The possible paths are stored in the leaves list. Each leaf has a
90      // 'runningCost' value where the lowest cost will be the best action
91      // sequence.
        2 references
92      private bool BuildGraph(HashSet<KeyValuePair<string, object>> _end,
93                              Node _parent, List<Node> leaves,
94                              HashSet<GOAPAction> _usableActions,
95                              HashSet<KeyValuePair<string, object>> _goal)
96      {
97          bool foundOne = false;
98
99          // go through each action available at this node and see if we can use it here
100         foreach (GOAPAction action in _usableActions)
101         {
102             // if the parent state has the conditions for this action's preconditions, we can use it here
103             if (InState(action.Effects, _parent.state))
104             {
105                 // apply the action's effects to the parent state
106                 HashSet<KeyValuePair<string, object>> currentState = PopulateState(_parent.state, action.Effects, action.Preconc
107                 // Console.WriteLine (GoapAgent.prettyPrint (currentState));
108                 // Console.WriteLine("");
109                 Node node = new Node(_parent, _parent.runningCost + action.cost, currentState, action);
110
111                 if (InState(currentState, _end))
112                 {
113                     // we found a solution!
114                     leaves.Add(node);
```



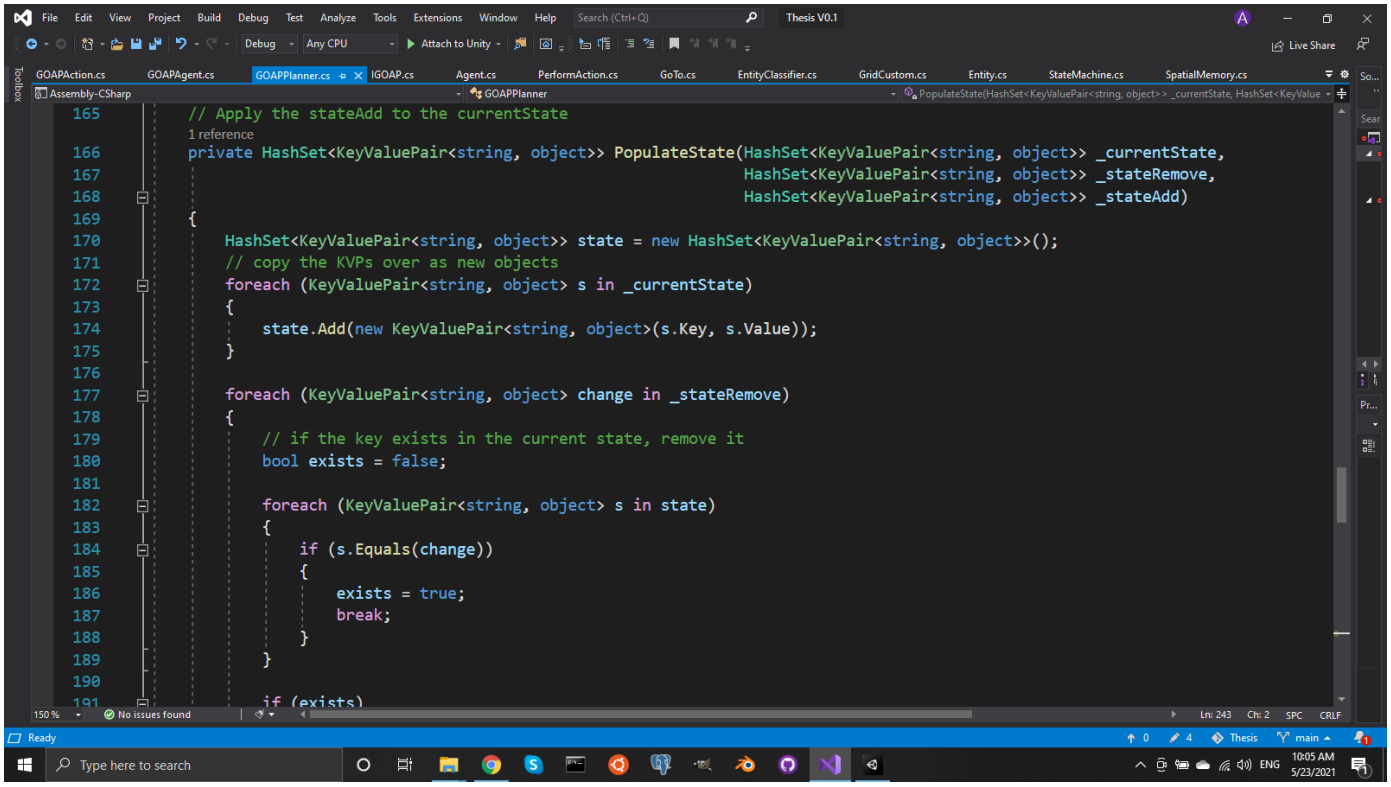
```
103             if (InState(action.Effects, _parent.state))
104             {
105                 // apply the action's effects to the parent state
106                 HashSet<KeyValuePair<string, object>> currentState = PopulateState(_parent.state, action.Effects, action.Preconc
107                 // Console.WriteLine (GoapAgent.prettyPrint (currentState));
108                 // Console.WriteLine("");
109                 Node node = new Node(_parent, _parent.runningCost + action.cost, currentState, action);
110
111                 if (InState(currentState, _end))
112                 {
113                     // we found a solution!
114                     leaves.Add(node);
115                     foundOne = true;
116                 }
117                 else
118                 {
119                     // not at a solution yet, so test all the remaining actions and branch out the tree
120                     HashSet<GOAPAction> subset = ActionSubset(_usableActions, action);
121                     bool found = BuildGraph(_end, node, leaves, subset, _goal);
122                     if (found)
123                         foundOne = true;
124                 }
125             }
126         }
127
128         return foundOne;
129     }
```



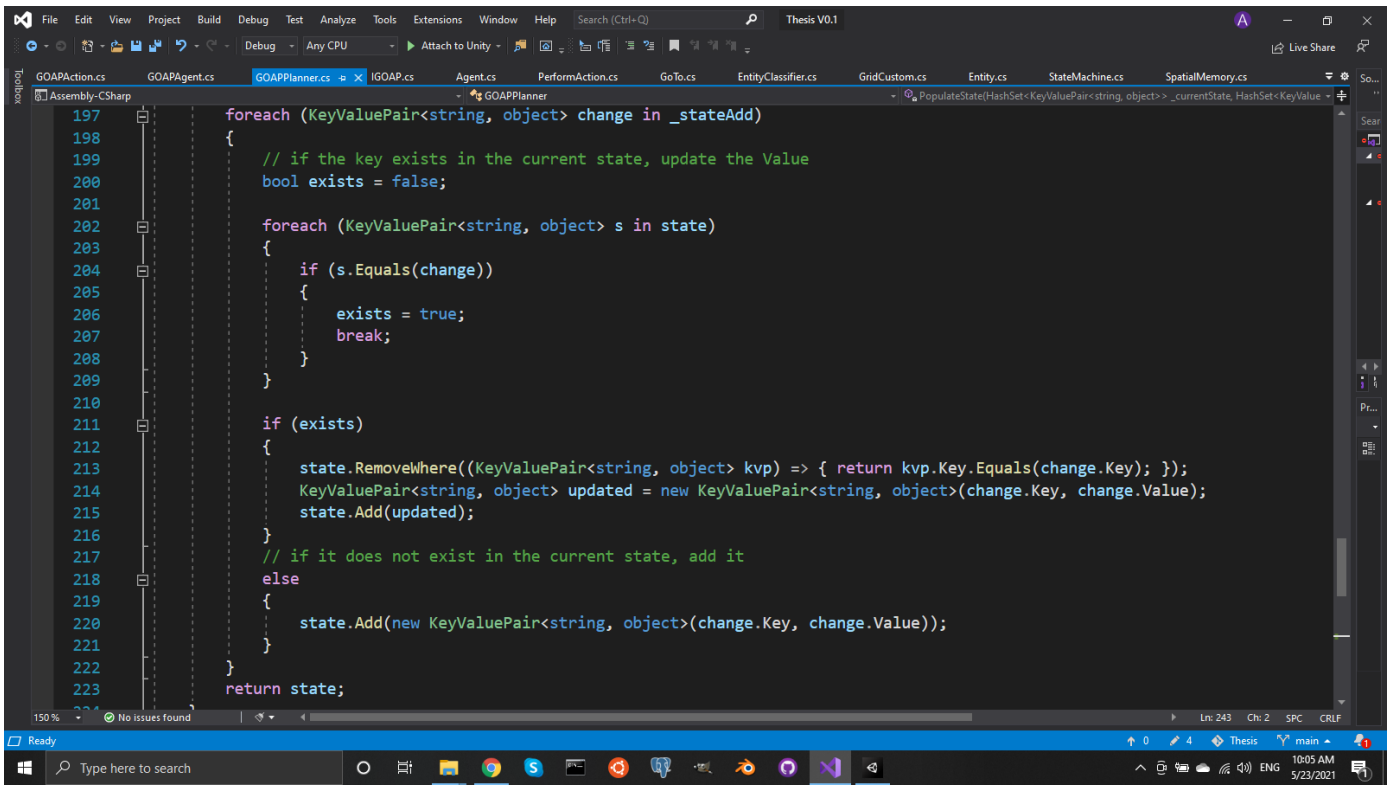
```
131     // Create a subset of the actions excluding the removeMe one. Creates a new set.
        1 reference
132     private HashSet<GOAPAction> ActionSubset(HashSet<GOAPAction> _actions, GOAPAction _removeMe)
133     {
134         HashSet<GOAPAction> subset = new HashSet<GOAPAction>();
135         foreach (GOAPAction action in _actions)
136         {
137             if (!action.Equals(_removeMe))
138                 subset.Add(action);
139         }
140         return subset;
141     }
142
143     // Check that all items in 'test' are in 'state'. If just one does not match or is not there
144     // then this returns false.
        2 references
145     private bool InState(HashSet<KeyValuePair<string, object>> _test, HashSet<KeyValuePair<string, object>> _state)
146     {
147         bool allMatch = true;
148         foreach (KeyValuePair<string, object> t in _test)
149         {
150             bool match = false;
151             foreach (KeyValuePair<string, object> s in _state)
152             {
153                 if (s.Equals(t))
154                 {
155                     match = true;
156                     break;
```



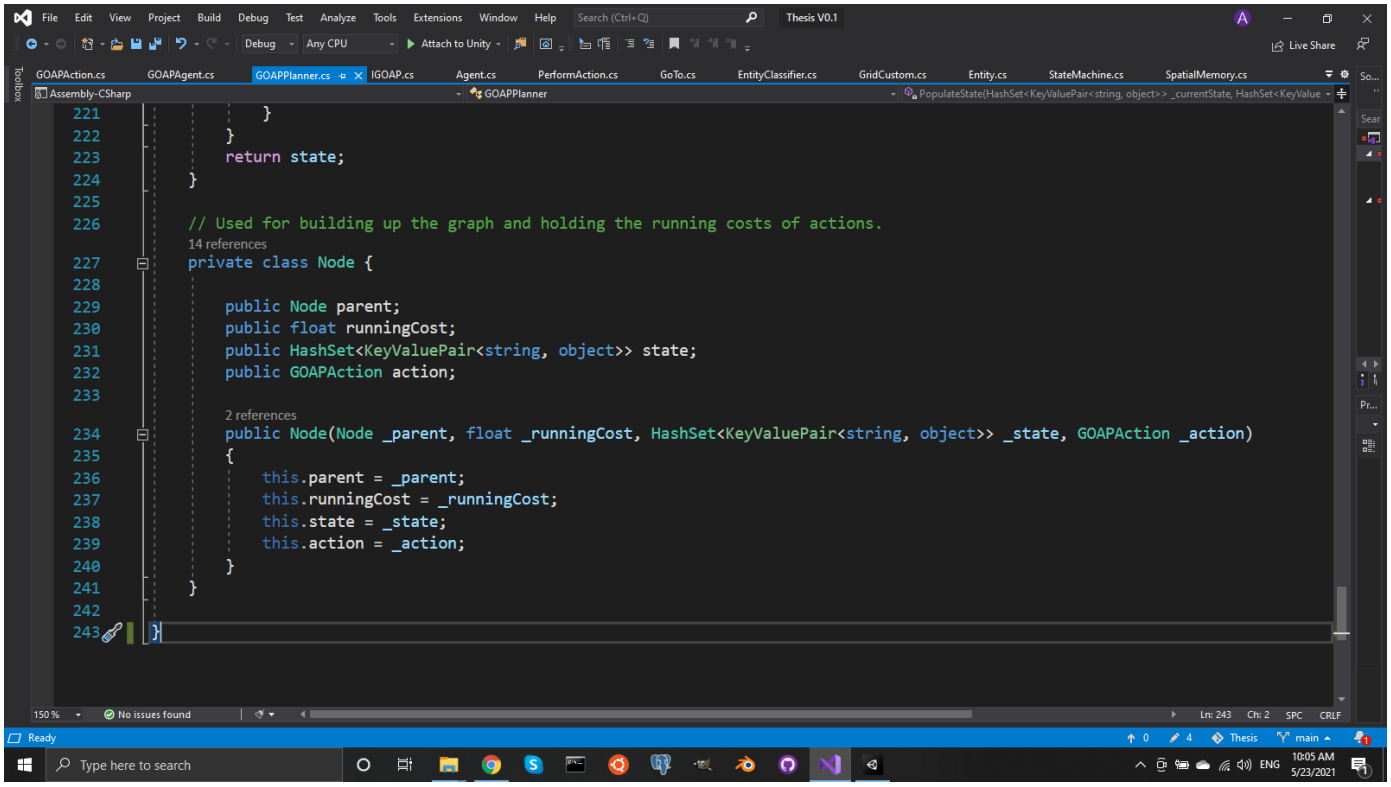
```
143     // Check that all items in 'test' are in 'state'. If just one does not match or is not there
144     // then this returns false.
        2 references
145     private bool InState(HashSet<KeyValuePair<string, object>> _test, HashSet<KeyValuePair<string, object>> _state)
146     {
147         bool allMatch = true;
148         foreach (KeyValuePair<string, object> t in _test)
149         {
150             bool match = false;
151             foreach (KeyValuePair<string, object> s in _state)
152             {
153                 if (s.Equals(t))
154                 {
155                     match = true;
156                     break;
157                 }
158             }
159             if (!match)
160                 allMatch = false;
161         }
162         return allMatch;
163     }
164
165     // Apply the stateAdd to the currentState
        1 reference
166     private HashSet<KeyValuePair<string, object>> PopulateState(HashSet<KeyValuePair<string, object>> _currentState,
167                                                                 HashSet<KeyValuePair<string, object>> _stateRemove,
168                                                                 HashSet<KeyValuePair<string, object>> _stateAdd)
```



```
165     // Apply the stateAdd to the currentState
        1 reference
166     private HashSet<KeyValuePair<string, object>> PopulateState(HashSet<KeyValuePair<string, object>> _currentState,
167                                                                 HashSet<KeyValuePair<string, object>> _stateRemove,
168                                                                 HashSet<KeyValuePair<string, object>> _stateAdd)
169     {
170         HashSet<KeyValuePair<string, object>> state = new HashSet<KeyValuePair<string, object>>();
171         // copy the KVPs over as new objects
172         foreach (KeyValuePair<string, object> s in _currentState)
173         {
174             state.Add(new KeyValuePair<string, object>(s.Key, s.Value));
175         }
176
177         foreach (KeyValuePair<string, object> change in _stateRemove)
178         {
179             // if the key exists in the current state, remove it
180             bool exists = false;
181
182             foreach (KeyValuePair<string, object> s in state)
183             {
184                 if (s.Equals(change))
185                 {
186                     exists = true;
187                     break;
188                 }
189             }
190
191             if (exists)
```



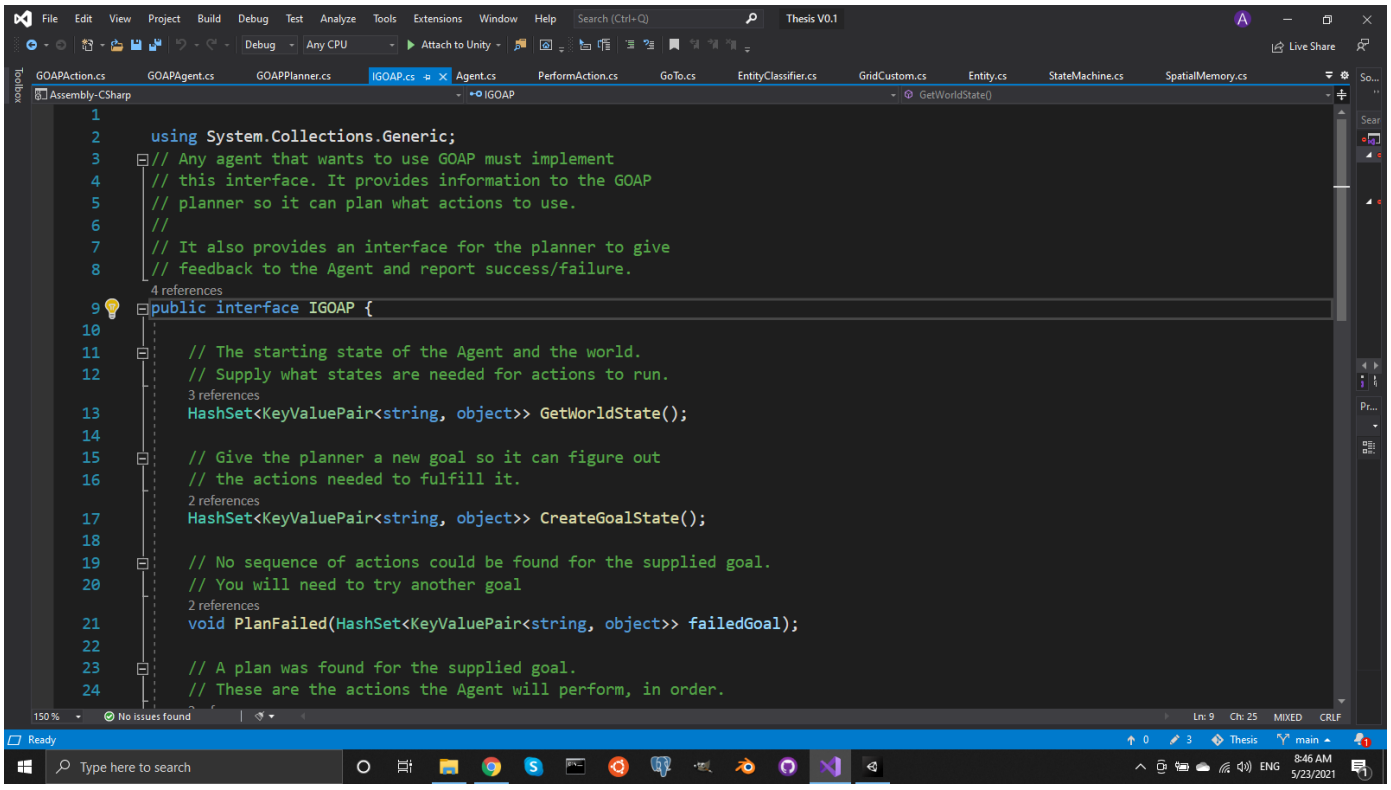
```
197         foreach (KeyValuePair<string, object> change in _stateAdd)
198         {
199             // if the key exists in the current state, update the Value
200             bool exists = false;
201
202             foreach (KeyValuePair<string, object> s in state)
203             {
204                 if (s.Equals(change))
205                 {
206                     exists = true;
207                     break;
208                 }
209             }
210
211             if (exists)
212             {
213                 state.RemoveWhere((KeyValuePair<string, object> kvp) => { return kvp.Key.Equals(change.Key); });
214                 KeyValuePair<string, object> updated = new KeyValuePair<string, object>(change.Key, change.Value);
215                 state.Add(updated);
216             }
217             // if it does not exist in the current state, add it
218             else
219             {
220                 state.Add(new KeyValuePair<string, object>(change.Key, change.Value));
221             }
222         }
223         return state;
```

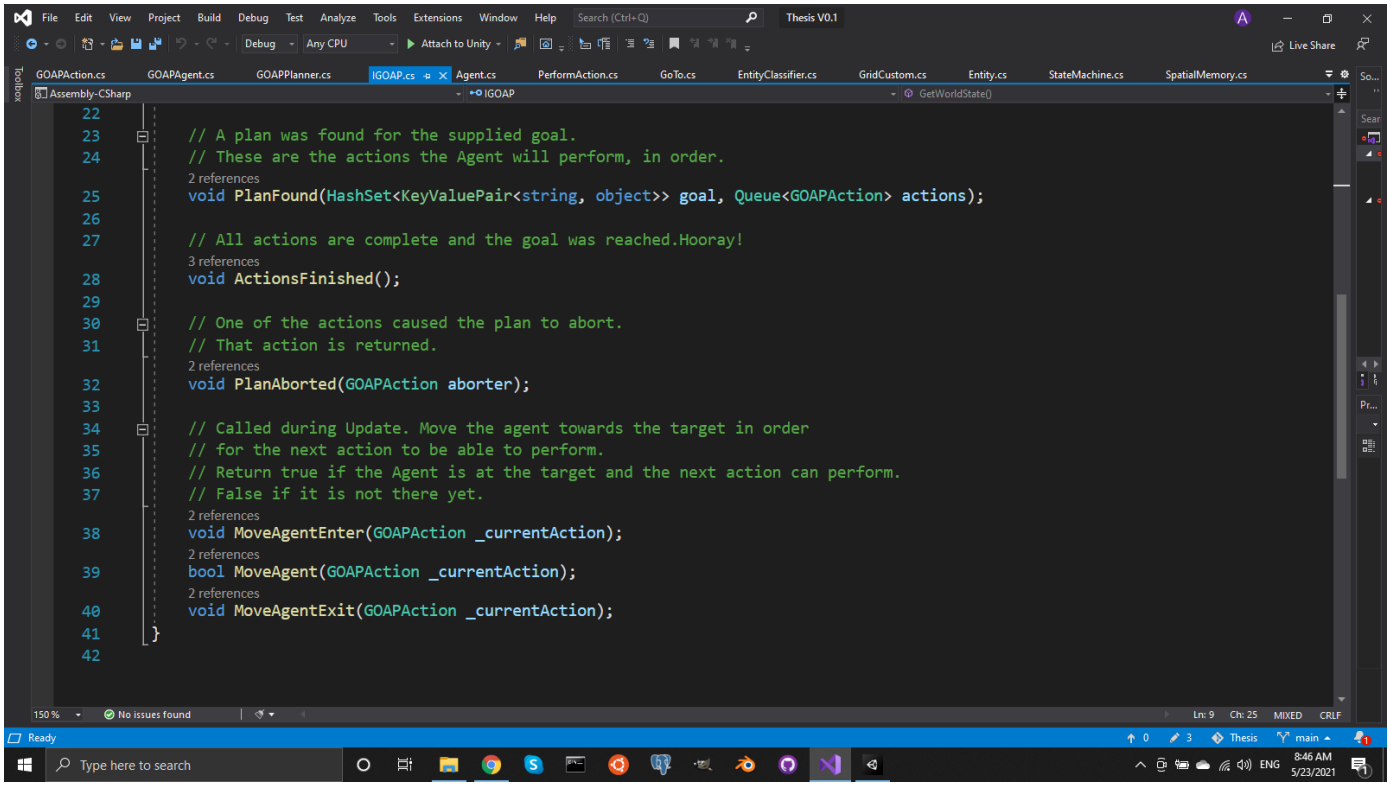

File Edit View Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
Thesis V0.1
Debug
Any CPU
Attach to Unity
Live Share
GOAPAction.cs
GOAPAgent.cs
GOAPPlanner.cs
IGOAP.cs
Agent.cs
PerformAction.cs
GoTo.cs
EntityClassifier.cs
GridCustom.cs
Entity.cs
StateMachine.cs
SpatialMemory.cs
Assembly-CSharp
GOAPPlanner
PopulateState(HashSet<KeyValuePair<string, object>> _currentState, HashSet<KeyValue
}
}
return state;
}
// Used for building up the graph and holding the running costs of actions.
14 references
private class Node {
public Node parent;
public float runningCost;
public HashSet<KeyValuePair<string, object>> state;
public GOAPAction action;
2 references
public Node(Node _parent, float _runningCost, HashSet<KeyValuePair<string, object>> _state, GOAPAction _action)
{
this.parent = _parent;
this.runningCost = _runningCost;
this.state = _state;
this.action = _action;
}
}
}
150 %
No issues found
Ln: 243 Ch: 2 SPC CRLF
Ready
Thesis
main
Type here to search
ENG
10:05 AM
5/23/2021

ΔΙΕΠΑΦΗ ΓΙΑ ΤΟ GOAP (IGOAP)

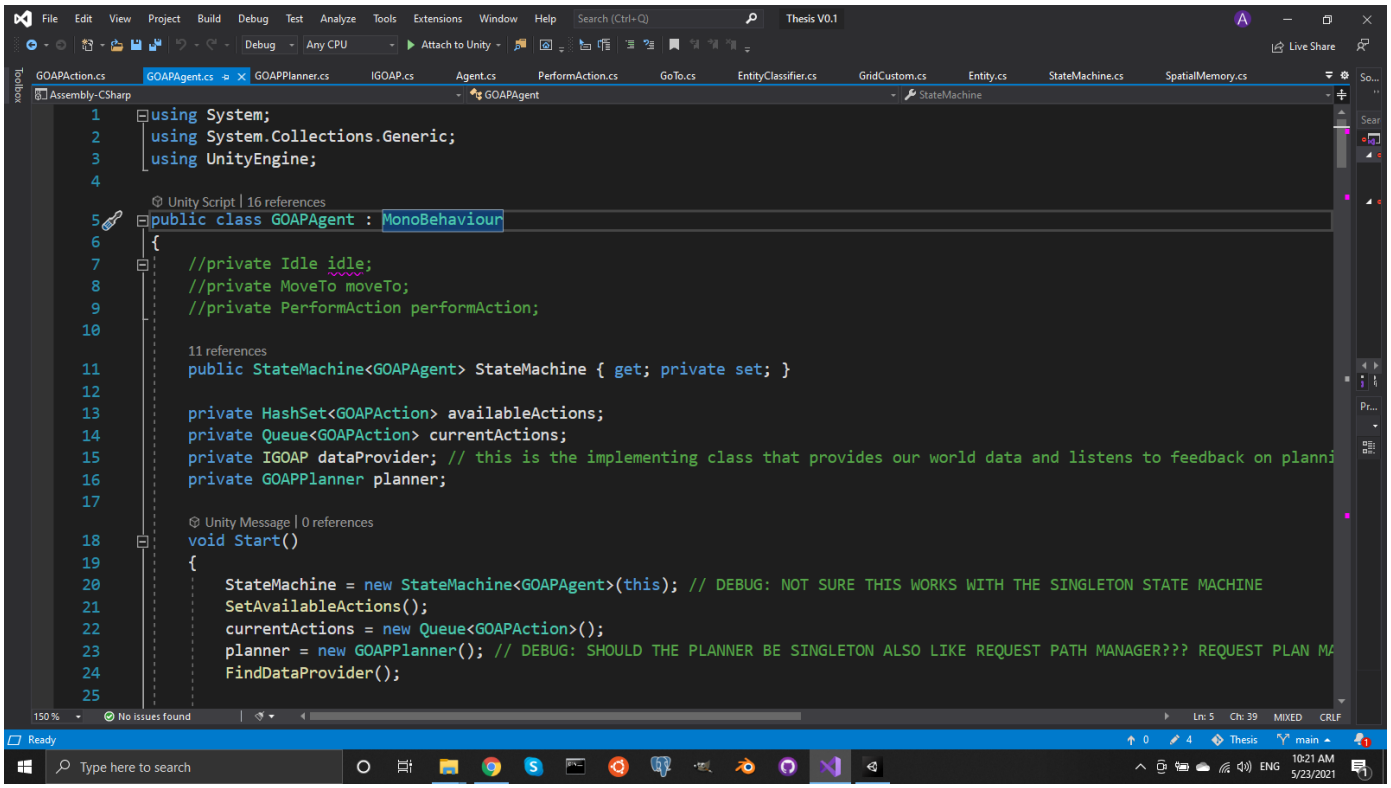
Το **IGOAP** script είναι ένα interface που πρέπει να εφαρμόζει ο κάθε πράκτορας (Agent) ώστε να χρησιμοποιήσει το σύστημα **GOAP**. Παρέχει πληροφορίες στο **GOAPPlanner** script και έτσι του δίνει την δυνατότητα να δημιουργήσει ένα σχέδιο. Παρέχει επίσης μια διεπαφή για το **GOAPPlanner** μέσω της οποίας δίνει αναφορά επιτυχίας ή αποτυχίας στον πράκτορα.

Δύο από τις σημαντικότερες μεθόδους που παρέχει το **IGOAP** script είναι η **GetWorldState** και η **CreateWorldState**. Με την **GetWorldState** παρέχει τις καταστάσεις που απαιτούνται για την εκτέλεση ενεργειών. Ενώ με την **CreateWorldState** δίνει στον planner έναν νέο στόχο, ώστε να βρει τις ενέργειες που απαιτούνται για την εκπλήρωσή του.

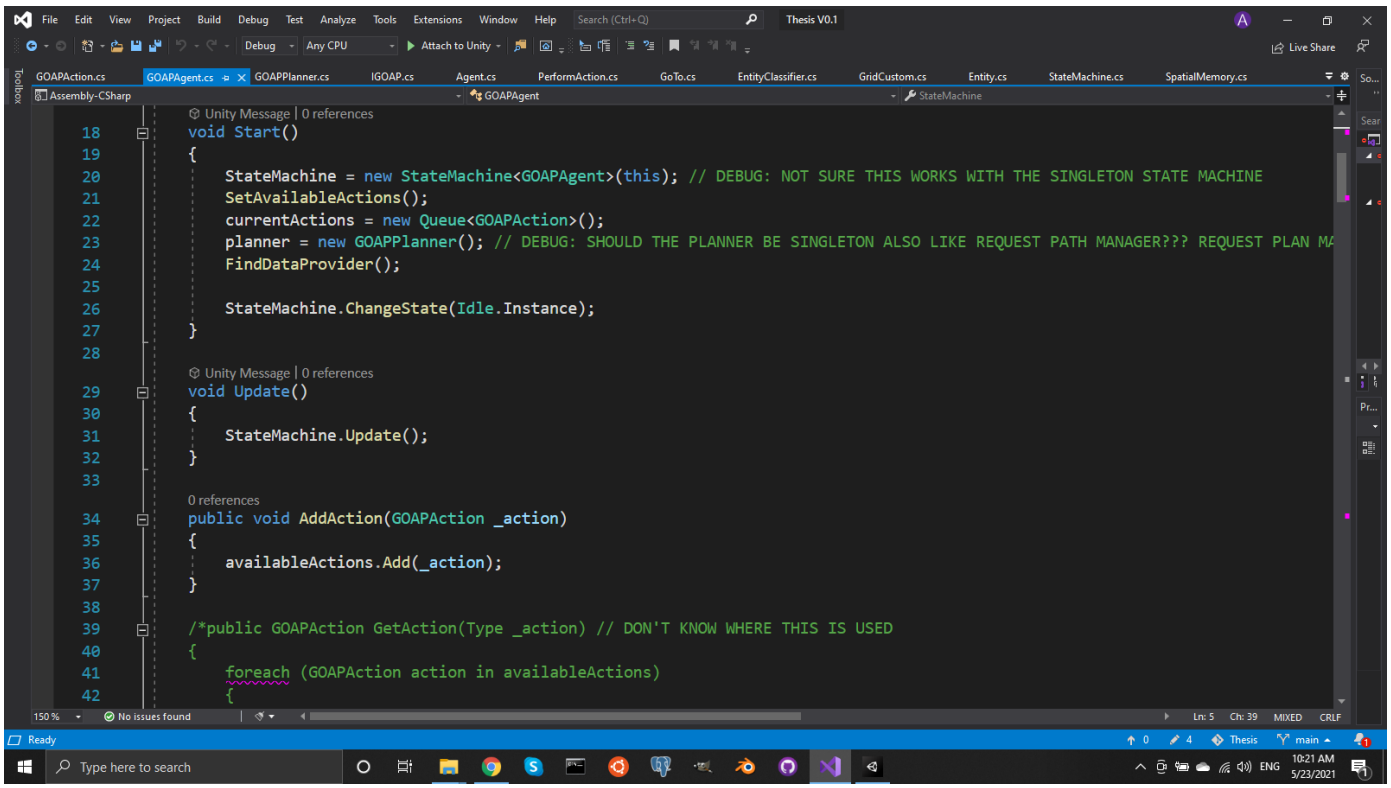



File Edit View Project Build Debug Test Analyze Tools Extensions Window Help
Thesis V0.1
Debug
Any CPU
Attach to Unity
Live Share
GOAPAction.cs
GOAPAgent.cs
GOAPPlanner.cs
IGOAP.cs
Agent.cs
PerformAction.cs
GoTo.cs
EntityClassifier.cs
GridCustom.cs
Entity.cs
StateMachine.cs
SpatialMemory.cs
Assembly-CSharp
IGOAP
GetWorldState()
// A plan was found for the supplied goal.
// These are the actions the Agent will perform, in order.
2 references
void PlanFound(HashSet<KeyValuePair<string, object>> goal, Queue<GOAPAction> actions);
// All actions are complete and the goal was reached.Hooray!
3 references
void ActionsFinished();
// One of the actions caused the plan to abort.
// That action is returned.
2 references
void PlanAborted(GOAPAction aborter);
// Called during Update. Move the agent towards the target in order
// for the next action to be able to perform.
// Return true if the Agent is at the target and the next action can perform.
// False if it is not there yet.
2 references
void MoveAgentEnter(GOAPAction _currentAction);
2 references
bool MoveAgent(GOAPAction _currentAction);
2 references
void MoveAgentExit(GOAPAction _currentAction);
}
150 %
No issues found
Ln: 9 Ch: 25 MIXED CRLF
Ready
Thesis
main
Type here to search
ENG
8:46 AM
5/23/2021

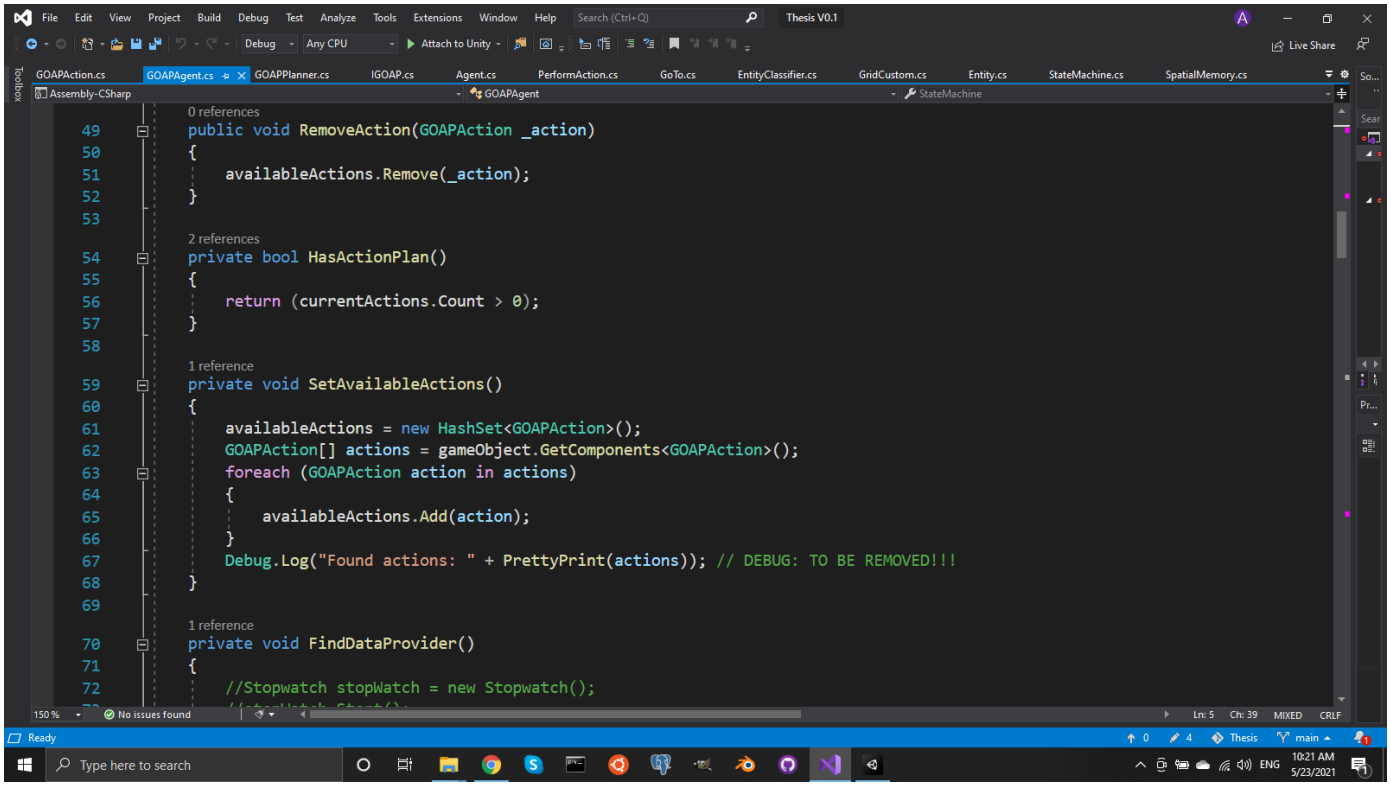
ΠΡΑΚΤΟΡΑΣ GOAP (GOAPAgent)



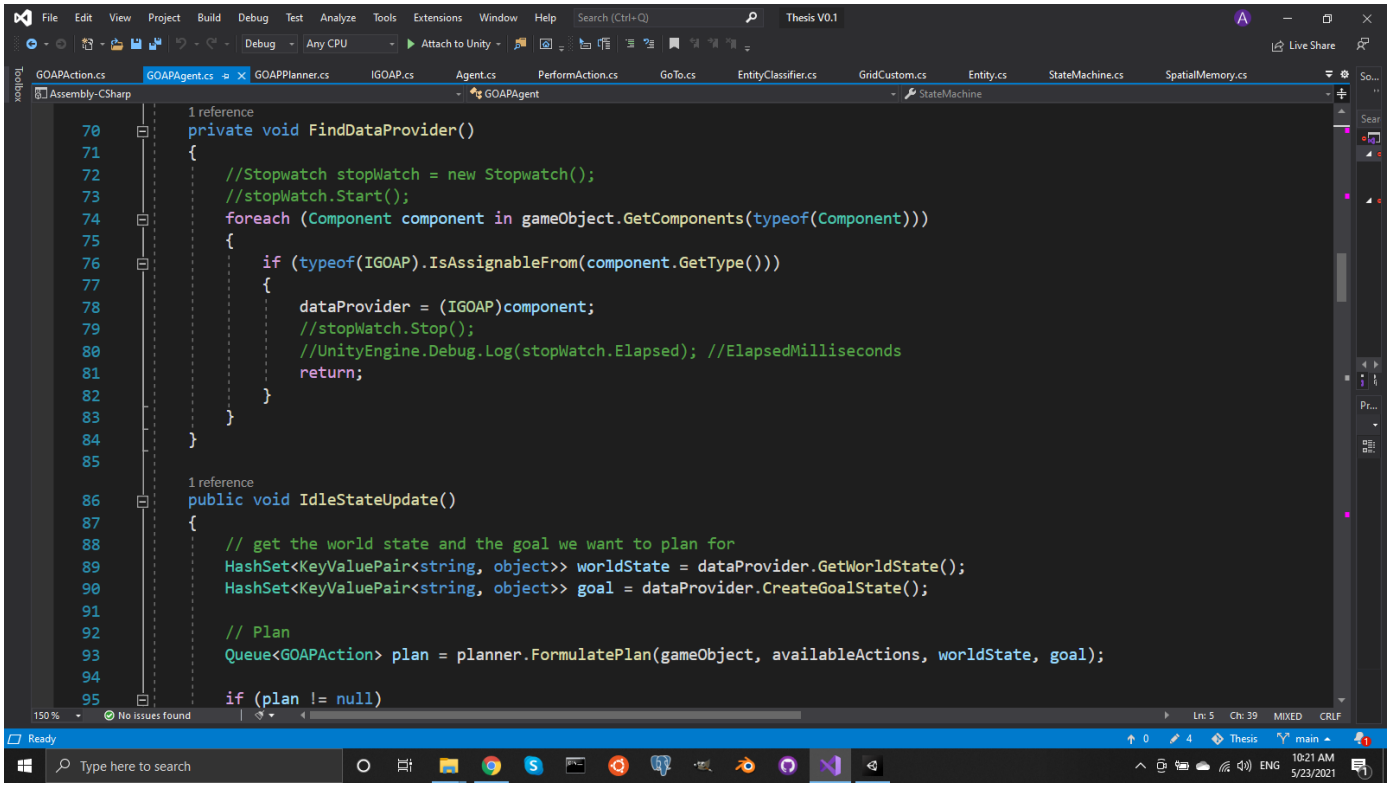
```
using System;
using System.Collections.Generic;
using UnityEngine;

Unity Script | 16 references
public class GOAPAgent : MonoBehaviour
{
    //private Idle idle;
    //private MoveTo moveTo;
    //private PerformAction performAction;

    11 references
    public StateMachine<GOAPAgent> StateMachine { get; private set; }

    private HashSet<GOAPAction> availableActions;
    private Queue<GOAPAction> currentActions;
    private IGOAP dataProvider; // this is the implementing class that provides our world data and listens to feedback on planni
    private GOAPPlanner planner;

    Unity Message | 0 references
    void Start()
    {
        StateMachine = new StateMachine<GOAPAgent>(this); // DEBUG: NOT SURE THIS WORKS WITH THE SINGLETON STATE MACHINE
        SetAvailableActions();
        currentActions = new Queue<GOAPAction>();
        planner = new GOAPPlanner(); // DEBUG: SHOULD THE PLANNER BE SINGLETON ALSO LIKE REQUEST PATH MANAGER??? REQUEST PLAN MA
        FindDataProvider();
```



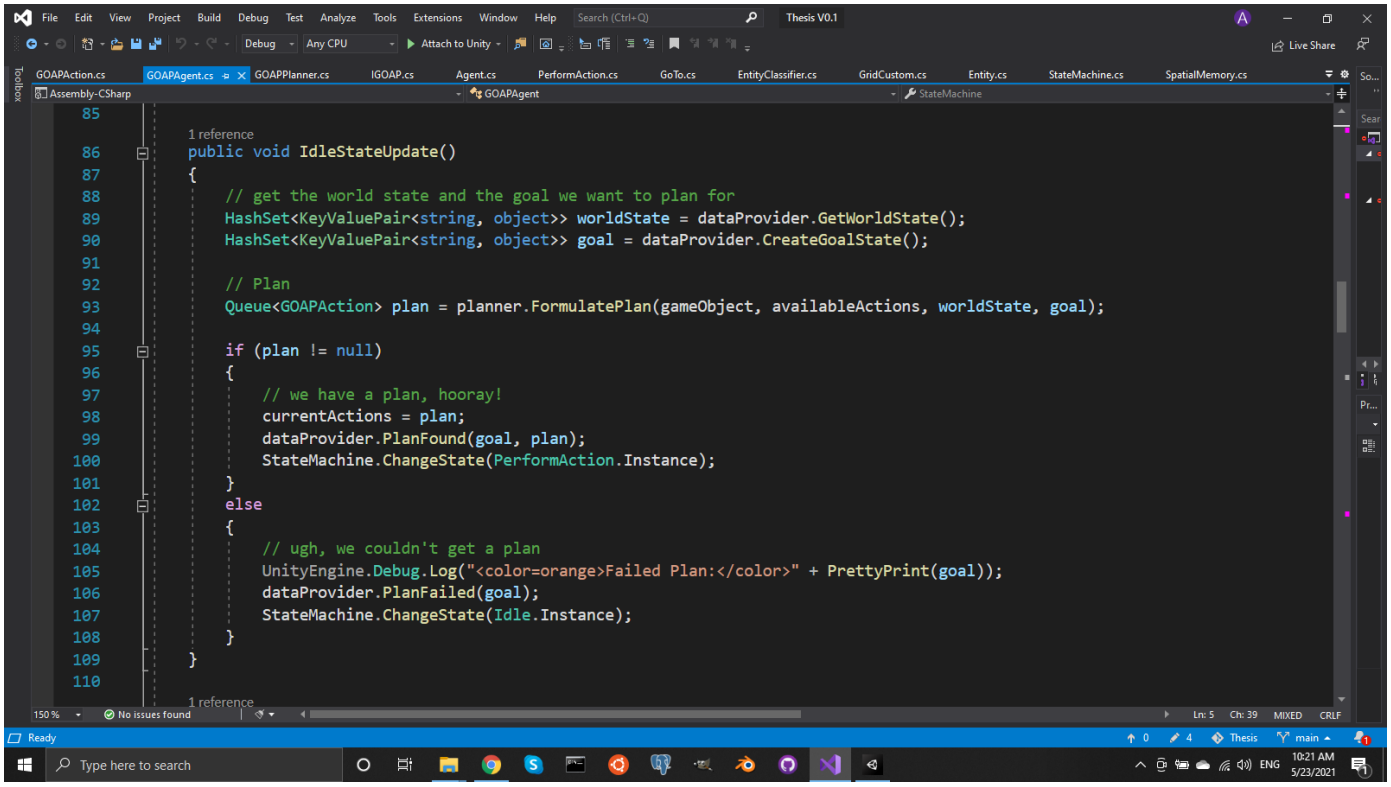
```
    Unity Message | 0 references
    void Start()
    {
        StateMachine = new StateMachine<GOAPAgent>(this); // DEBUG: NOT SURE THIS WORKS WITH THE SINGLETON STATE MACHINE
        SetAvailableActions();
        currentActions = new Queue<GOAPAction>();
        planner = new GOAPPlanner(); // DEBUG: SHOULD THE PLANNER BE SINGLETON ALSO LIKE REQUEST PATH MANAGER??? REQUEST PLAN MA
        FindDataProvider();

        StateMachine.ChangeState(Idle.Instance);
    }

    Unity Message | 0 references
    void Update()
    {
        StateMachine.Update();
    }

    0 references
    public void AddAction(GOAPAction _action)
    {
        availableActions.Add(_action);
    }

    /*public GOAPAction GetAction(Type _action) // DON'T KNOW WHERE THIS IS USED
    {
        foreach (GOAPAction action in availableActions)
        {
```



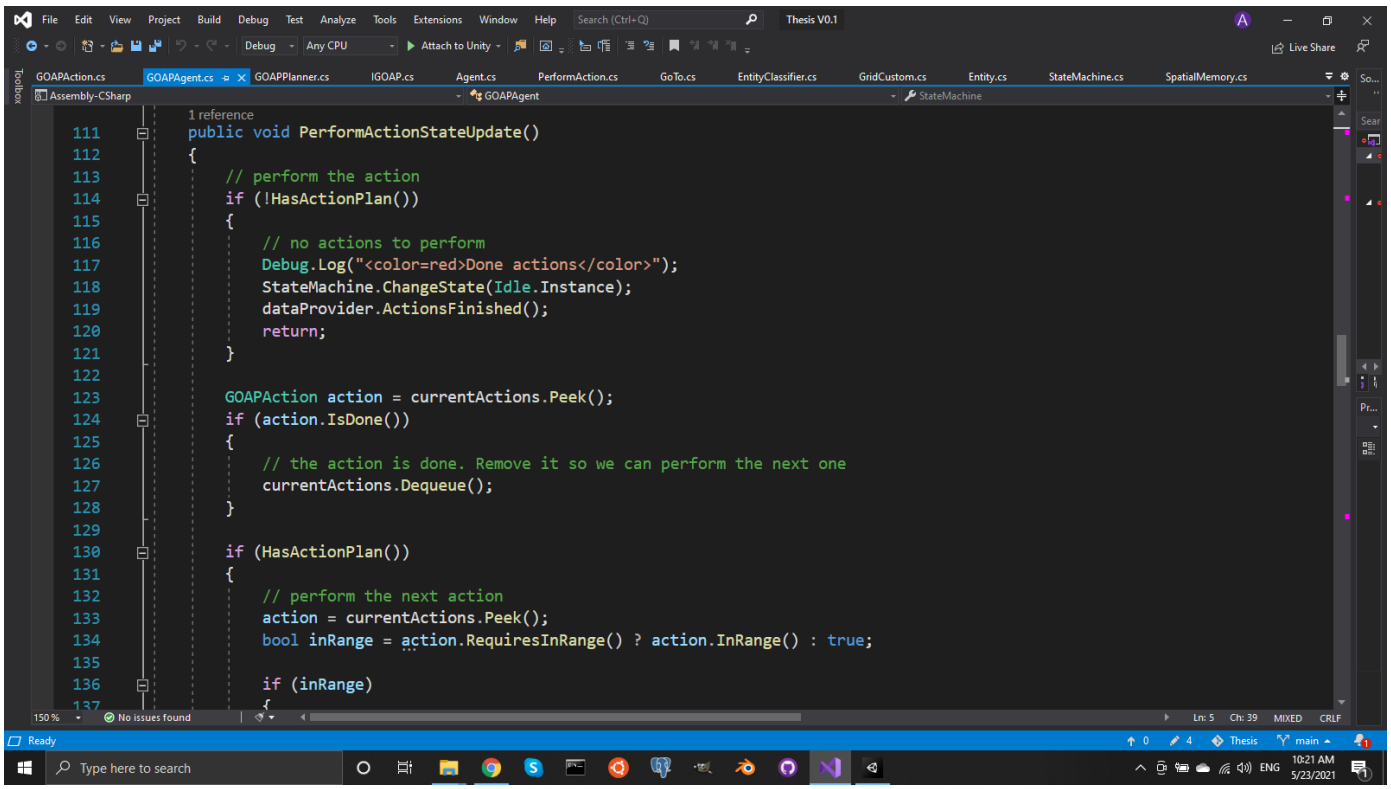
```
0 references
public void RemoveAction(GOAPAction _action)
{
    availableActions.Remove(_action);
}

2 references
private bool HasActionPlan()
{
    return (currentActions.Count > 0);
}

1 reference
private void SetAvailableActions()
{
    availableActions = new HashSet<GOAPAction>();
    GOAPAction[] actions = gameObject.GetComponents<GOAPAction>();
    foreach (GOAPAction action in actions)
    {
        availableActions.Add(action);
    }
    Debug.Log("Found actions: " + PrettyPrint(actions)); // DEBUG: TO BE REMOVED!!!
}

1 reference
private void FindDataProvider()
{
    //Stopwatch stopWatch = new Stopwatch();
```



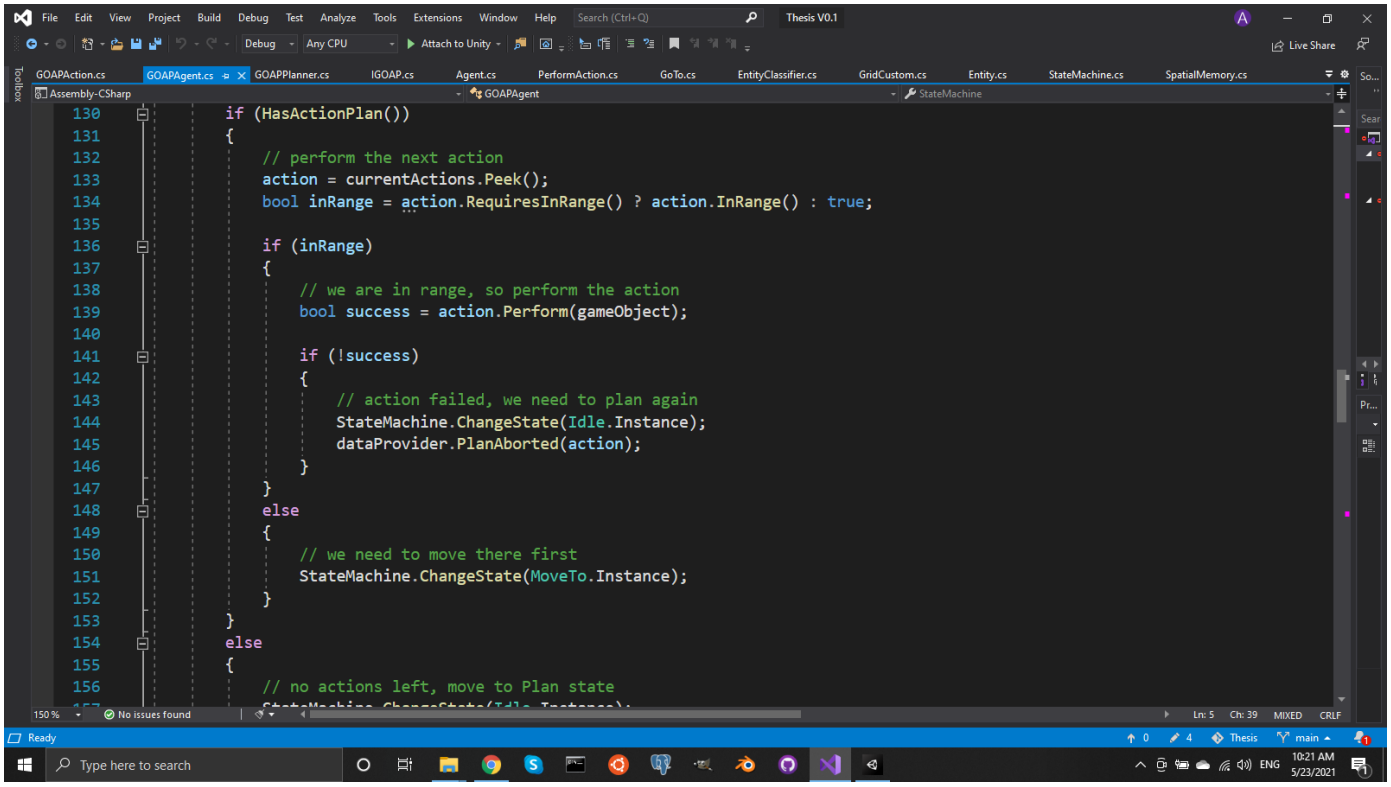
```
1 reference
private void FindDataProvider()
{
    //Stopwatch stopWatch = new Stopwatch();
    //stopWatch.Start();
    foreach (Component component in gameObject.GetComponents(typeof(Component)))
    {
        if (typeof(IGOAP).IsAssignableFrom(component.GetType()))
        {
            dataProvider = (IGOAP)component;
            //stopWatch.Stop();
            //UnityEngine.Debug.Log(stopWatch.Elapsed); //ElapsedMilliseconds
            return;
        }
    }
}

1 reference
public void IdleStateUpdate()
{
    // get the world state and the goal we want to plan for
    HashSet<KeyValuePair<string, object>> worldState = dataProvider.GetWorldState();
    HashSet<KeyValuePair<string, object>> goal = dataProvider.CreateGoalState();

    // Plan
    Queue<GOAPAction> plan = planner.FormulatePlan(gameObject, availableActions, worldState, goal);

    if (plan != null)
```



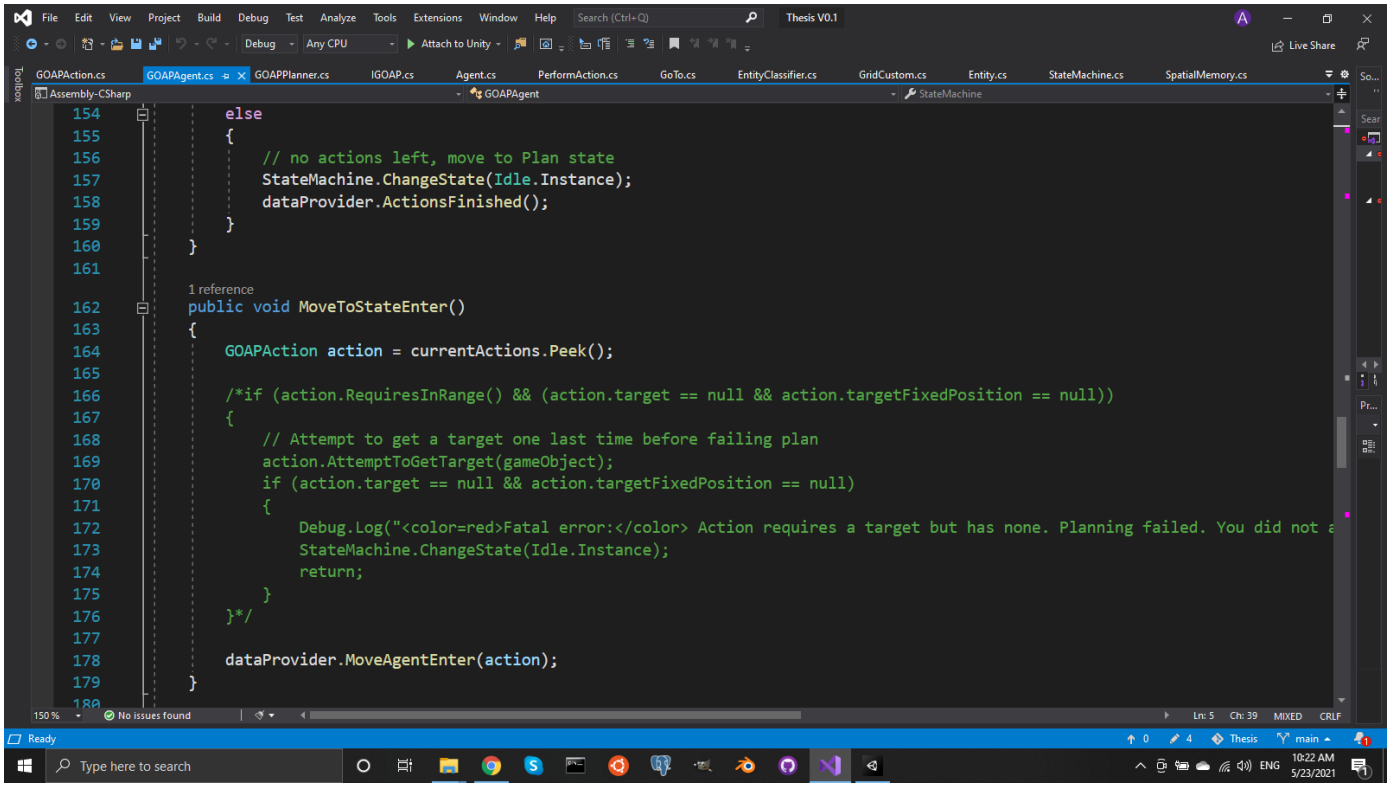
```
85
        1 reference
86      public void IdleStateUpdate()
87      {
88          // get the world state and the goal we want to plan for
89          HashSet<KeyValuePair<string, object>> worldState = dataProvider.GetWorldState();
90          HashSet<KeyValuePair<string, object>> goal = dataProvider.CreateGoalState();
91
92          // Plan
93          Queue<GOAPAction> plan = planner.FormulatePlan(gameObject, availableActions, worldState, goal);
94
95          if (plan != null)
96          {
97              // we have a plan, hooray!
98              currentActions = plan;
99              dataProvider.PlanFound(goal, plan);
100             StateMachine.ChangeState(PerformAction.Instance);
101         }
102         else
103         {
104             // ugh, we couldn't get a plan
105             UnityEngine.Debug.Log("<color=orange>Failed Plan:</color>" + PrettyPrint(goal));
106             dataProvider.PlanFailed(goal);
107             StateMachine.ChangeState(Idle.Instance);
108         }
109     }
110
        1 reference
```



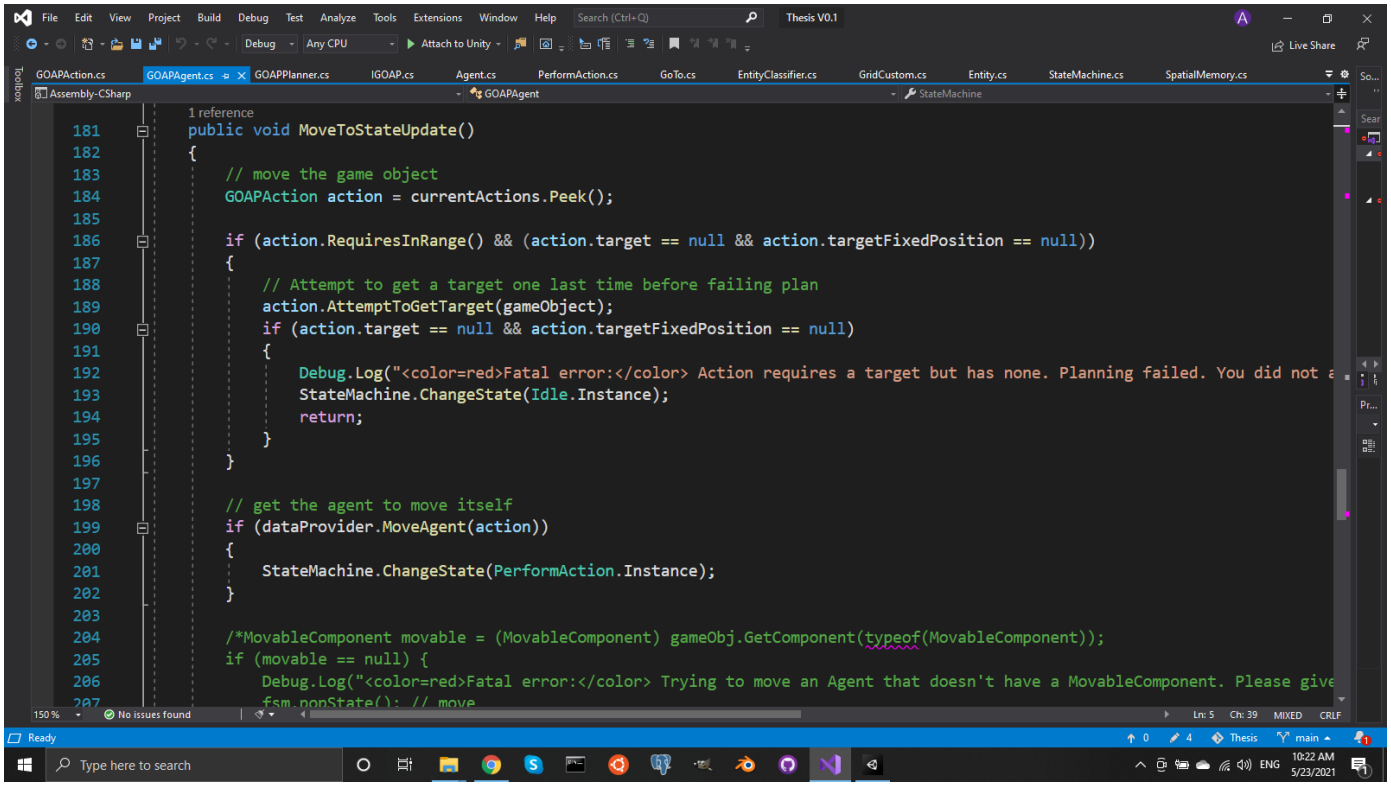
```
        1 reference
111     public void PerformActionStateUpdate()
112     {
113         // perform the action
114         if (!HasActionPlan())
115         {
116             // no actions to perform
117             Debug.Log("<color=red>Done actions</color>");
118             StateMachine.ChangeState(Idle.Instance);
119             dataProvider.ActionsFinished();
120             return;
121         }
122
123         GOAPAction action = currentActions.Peek();
124         if (action.IsDone())
125         {
126             // the action is done. Remove it so we can perform the next one
127             currentActions.Dequeue();
128         }
129
130         if (HasActionPlan())
131         {
132             // perform the next action
133             action = currentActions.Peek();
134             bool inRange = action.RequiresInRange() ? action.InRange() : true;
135
136             if (inRange)
137             {
```



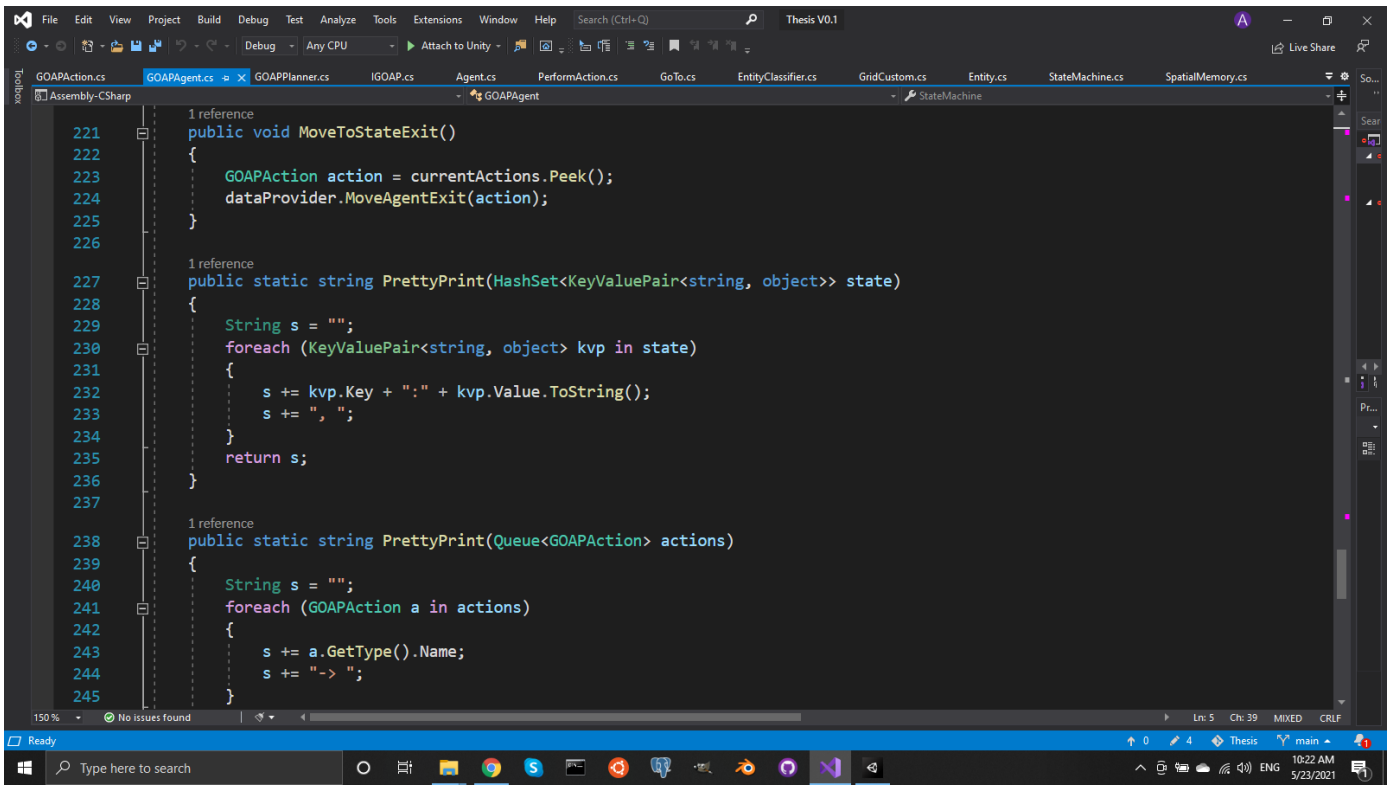
```
130         if (HasActionPlan())
131         {
132             // perform the next action
133             action = currentActions.Peek();
134             bool inRange = action.RequiresInRange() ? action.InRange() : true;
135
136             if (inRange)
137             {
138                 // we are in range, so perform the action
139                 bool success = action.Perform(gameObject);
140
141                 if (!success)
142                 {
143                     // action failed, we need to plan again
144                     StateMachine.ChangeState(Idle.Instance);
145                     dataProvider.PlanAborted(action);
146                 }
147             }
148             else
149             {
150                 // we need to move there first
151                 StateMachine.ChangeState(MoveTo.Instance);
152             }
153         }
154         else
155         {
156             // no actions left, move to Plan state
```



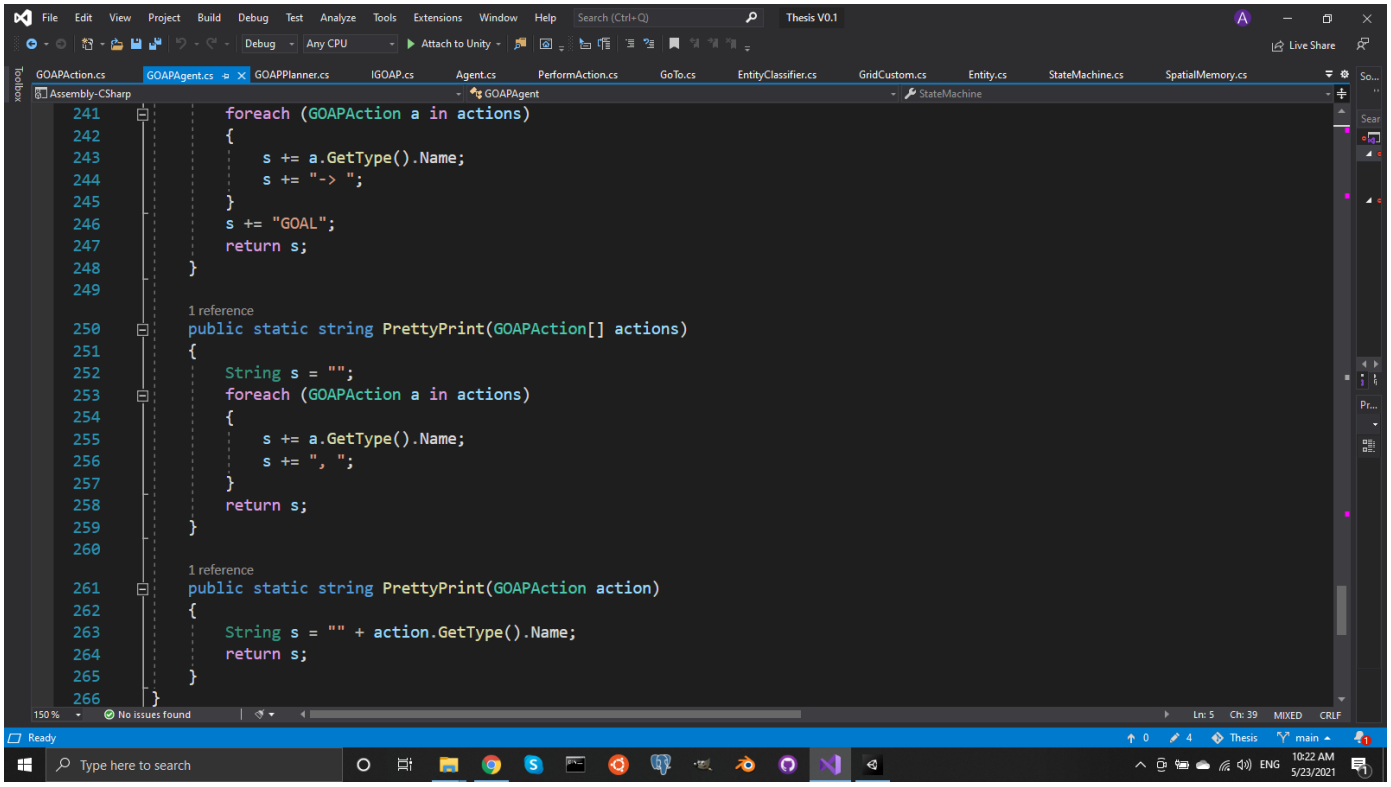
```
154         else
155         {
156             // no actions left, move to Plan state
157             StateMachine.ChangeState(Idle.Instance);
158             dataProvider.ActionsFinished();
159         }
160     }
161
        1 reference
162     public void MoveToStateEnter()
163     {
164         GOAPAction action = currentActions.Peek();
165
166         /*if (action.RequiresInRange() && (action.target == null && action.targetFixedPosition == null))
167         {
168             // Attempt to get a target one last time before failing plan
169             action.AttemptToGetTarget(gameObject);
170             if (action.target == null && action.targetFixedPosition == null)
171             {
172                 Debug.Log("<color=red>Fatal error:</color> Action requires a target but has none. Planning failed. You did not a
173                 StateMachine.ChangeState(Idle.Instance);
174                 return;
175             }
176         }*/
177
178         dataProvider.MoveAgentEnter(action);
179     }
```



```
1 reference
public void MoveToStateUpdate()
{
    // move the game object
    GOAPAction action = currentActions.Peek();

    if (action.RequiresInRange() && (action.target == null && action.targetFixedPosition == null))
    {
        // Attempt to get a target one last time before failing plan
        action.AttemptToGetTarget(gameObject);
        if (action.target == null && action.targetFixedPosition == null)
        {
            Debug.Log("<color=red>Fatal error:</color> Action requires a target but has none. Planning failed. You did not a
            StateMachine.ChangeState(Idle.Instance);
            return;
        }
    }

    // get the agent to move itself
    if (dataProvider.MoveAgent(action))
    {
        StateMachine.ChangeState(PerformAction.Instance);
    }

    /*MovableComponent movable = (MovableComponent) gameObj.GetComponent(typeof(MovableComponent));
    if (movable == null) {
        Debug.Log("<color=red>Fatal error:</color> Trying to move an Agent that doesn't have a MovableComponent. Please give
        fsm.popState(); // move
```



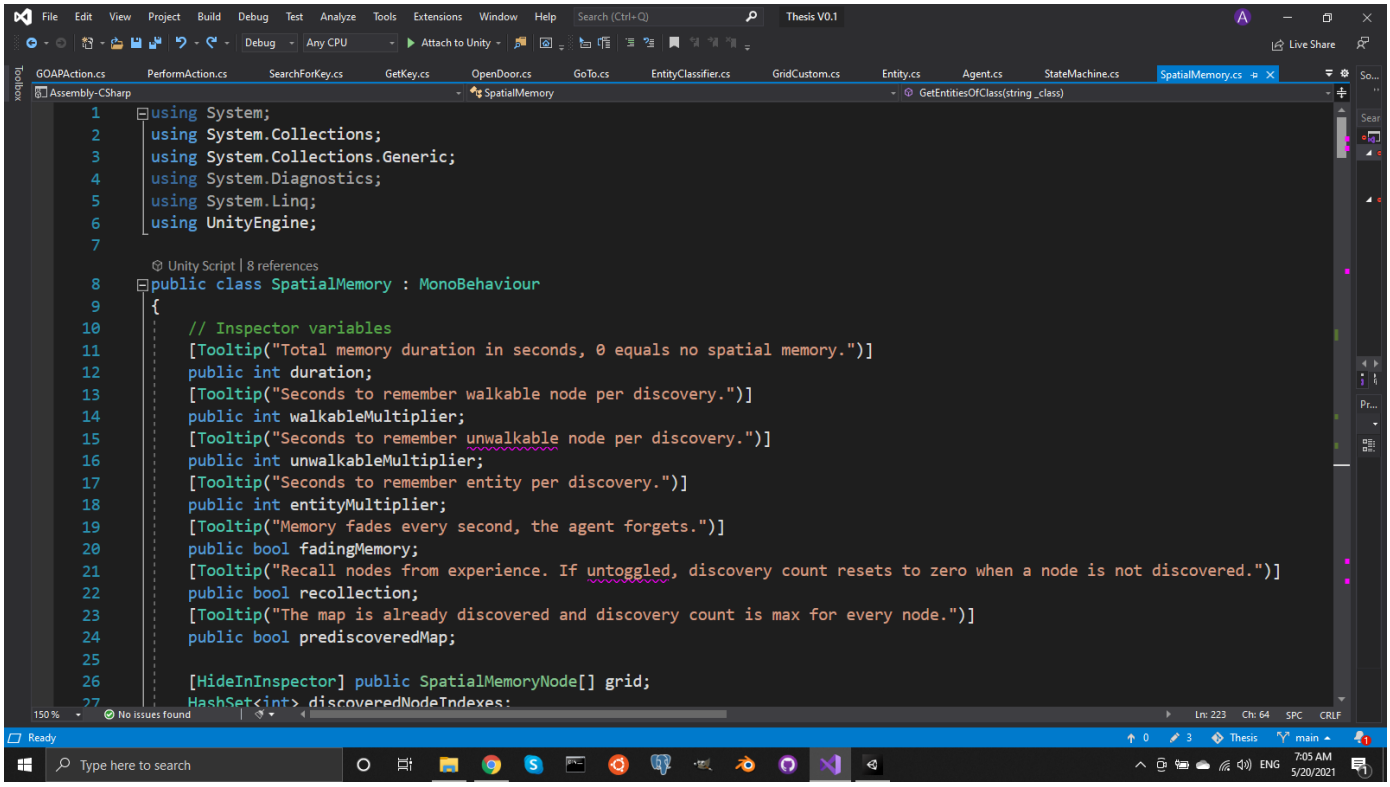
```
1 reference
public void MoveToStateExit()
{
    GOAPAction action = currentActions.Peek();
    dataProvider.MoveAgentExit(action);
}

1 reference
public static string PrettyPrint(HashSet<KeyValuePair<string, object>> state)
{
    String s = "";
    foreach (KeyValuePair<string, object> kvp in state)
    {
        s += kvp.Key + ":" + kvp.Value.ToString();
        s += ", ";
    }
    return s;
}

1 reference
public static string PrettyPrint(Queue<GOAPAction> actions)
{
    String s = "";
    foreach (GOAPAction a in actions)
    {
        s += a.GetType().Name;
        s += "-> ";
    }
```


File Edit View Project Build Debug Test Analyze Tools Extensions Window Help
Search (Ctrl+Q)
Thesis V0.1
Debug
Any CPU
Attach to Unity
Live Share
GOAPAction.cs
GOAPAgent.cs
GOAPPlanner.cs
IGOAP.cs
Agent.cs
PerformAction.cs
GoTo.cs
EntityClassifier.cs
GridCustom.cs
Entity.cs
StateMachine.cs
SpatialMemory.cs
Assembly-CSharp
GOAPAgent
StateMachine
foreach (GOAPAction a in actions)
{
s += a.GetType().Name;
s += "-> ";
}
s += "GOAL";
return s;
}
1 reference
public static string PrettyPrint(GOAPAction[] actions)
{
String s = "";
foreach (GOAPAction a in actions)
{
s += a.GetType().Name;
s += ", ";
}
return s;
}
1 reference
public static string PrettyPrint(GOAPAction action)
{
String s = "" + action.GetType().Name;
return s;
}
}
150 %
No issues found
Ln: 5 Ch: 39 MIXED CRLF
Ready
Thesis
main
Type here to search
ENG
10:22 AM
5/23/2021

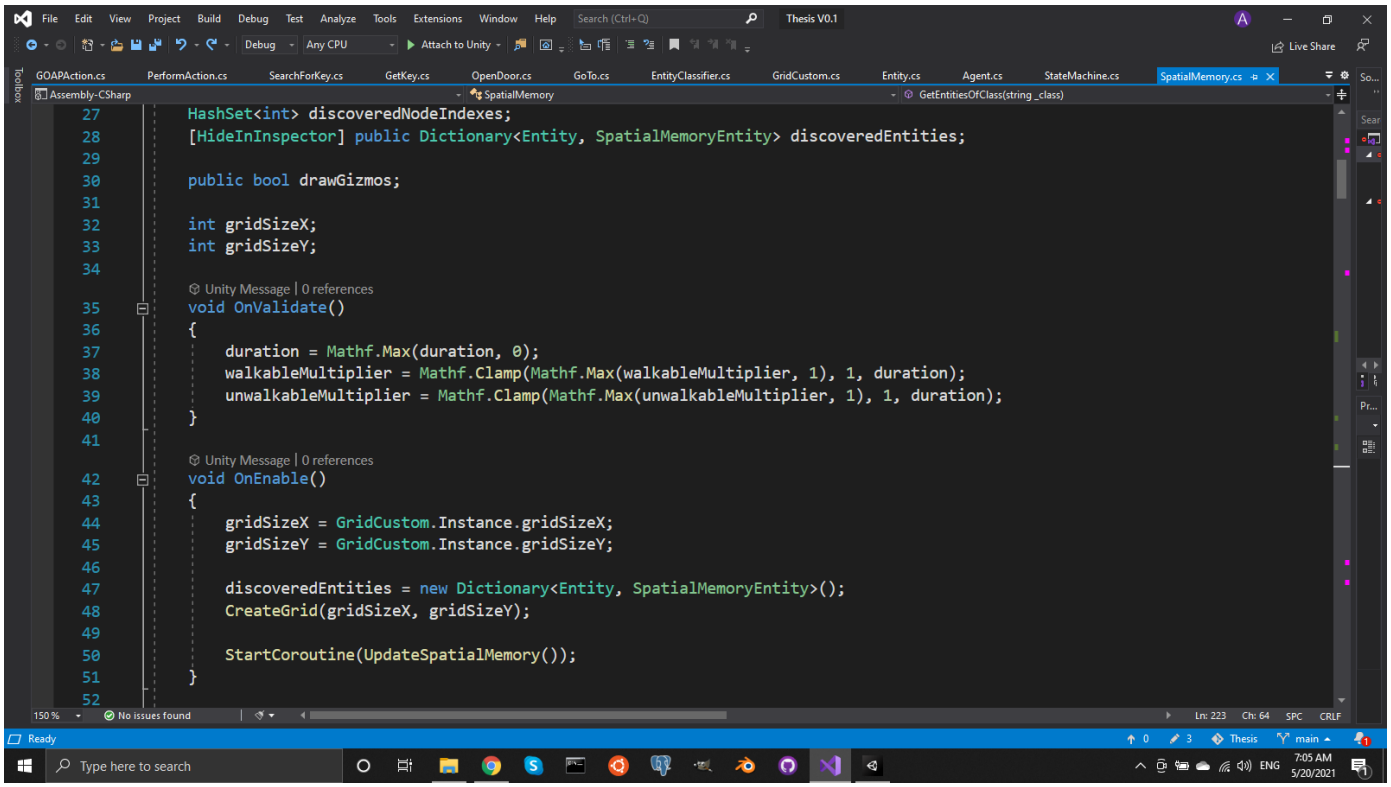
ΧΩΡΙΚΗ ΜΝΗΜΗ (SPATIAL MEMORY)

Για να προσομοιώσω την χωρική μνήμη του πράκτορα, η οποία του επιτρέπει να εξερευνά και να απομνημονεύει το λαβύρινθο καθώς και τα αντικείμενα που βρίσκει μέσα σε αυτόν, έφτιαξα την κλάση **Spatial Memory**. Η **Spatial Memory** δημιουργεί ένα πλέγμα (grid) το οποίο χωρίζει τον χώρο σε τετράγωνα κελιά τα οποία ο πράκτορας ανακαλύπτει χρησιμοποιώντας την όρασή του. Με αυτό τον τρόπο γνωρίζει ποιά τμήματα του λαβυρίνθου έχει επισκεφτεί και σε ποιά κελιά βρίσκονται τα αντικείμενα με τα οποία μπορεί να αλληλεπιδράσει καθώς και τα κελιά που έχουν εμπόδια. Μία ακόμα δυνατότητα της **Spatial Memory** είναι η ικανότητα να ξεχνά με την πάροδο του χρόνου.

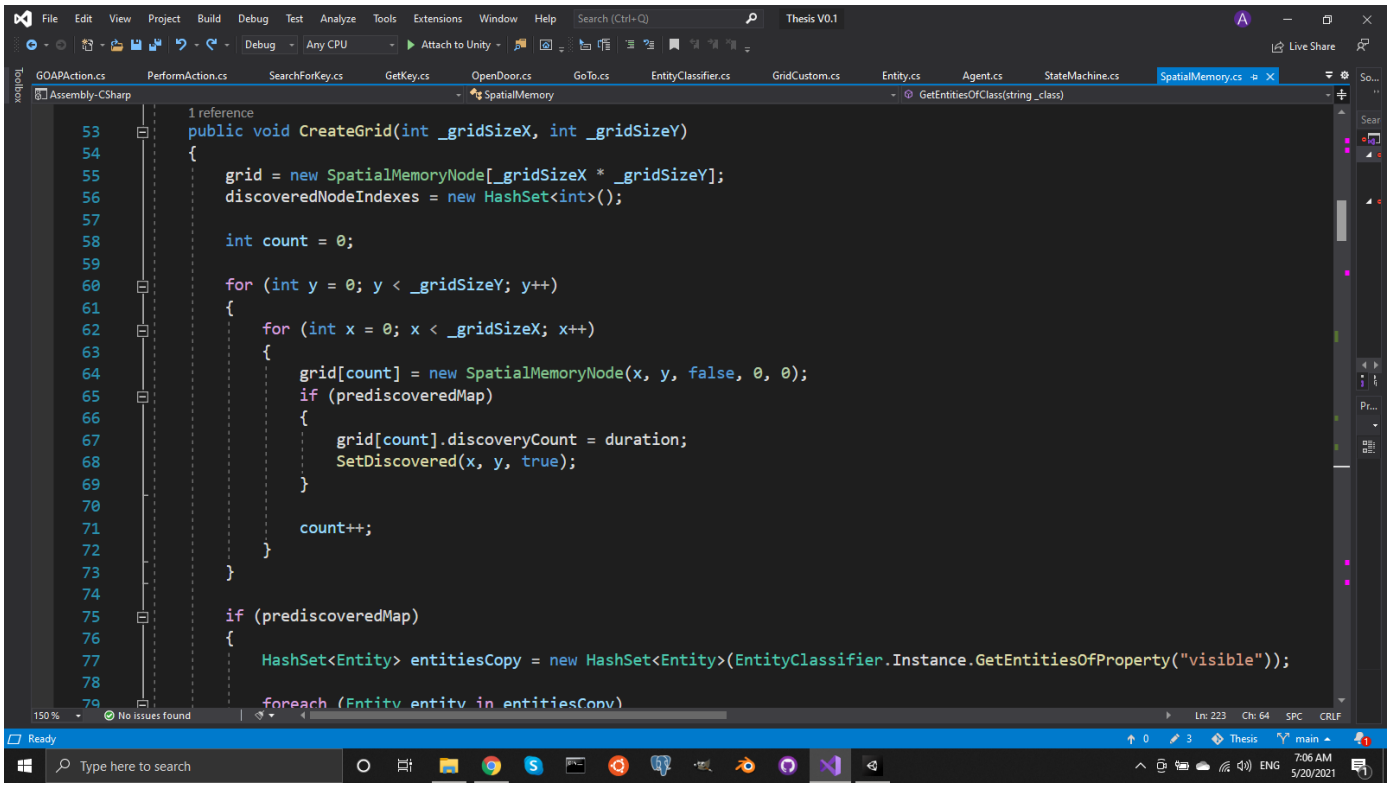
Στην παρακάτω screenshot βλέπουμε τις μεταβλητές της κλάσης που έχει πρόσβαση ο προγραμματιστής μέσω του Inspector της Unity και μπορεί να θέσει ο ίδιος της τιμές τους εκεί. Επίσης υπάρχουν τα tooltips τα οποία περιγράφουν τι αλλάζει με την κάθε μεταβλητή.



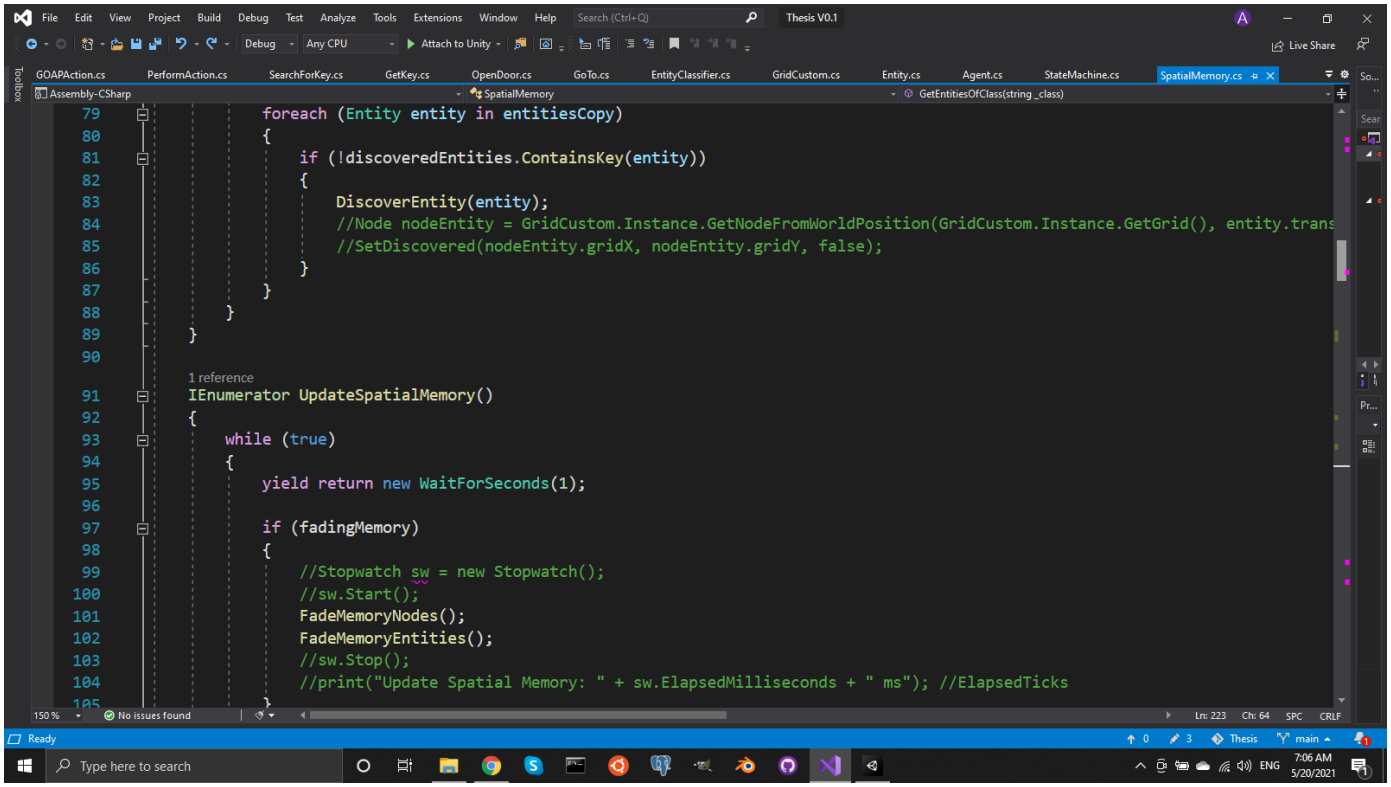
Χρησιμοποιώντας τη μέθοδο *OnEnable* της Unity που καλείται όταν ενεργοποιείται ένα game object η **Spatial Memory** καλώντας τη μέθοδο *CreateGrid* δημιουργεί ένα grid μεγέθους gridSizeX x gridSizeY.

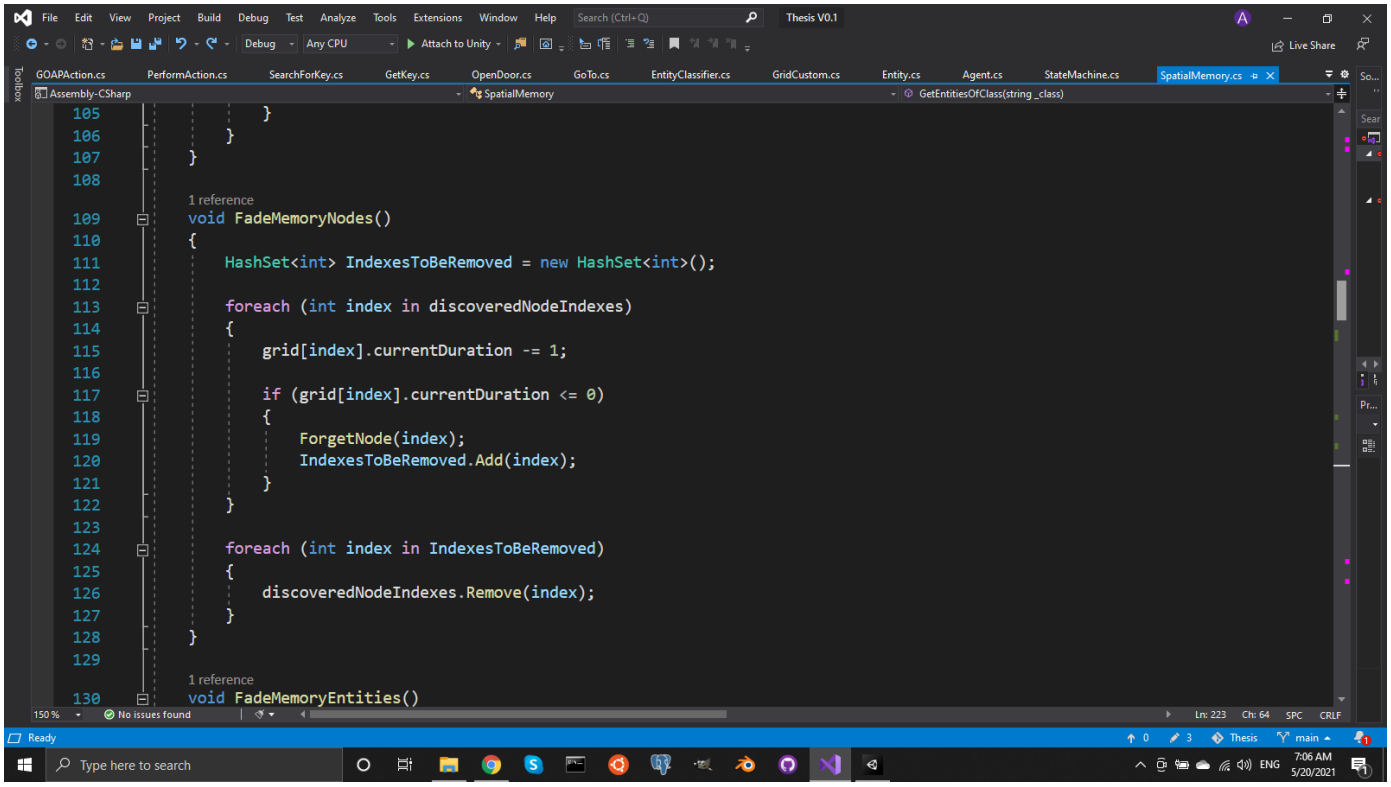


Αν ο προγραμματιστής έχει θέσει την μεταβλητή prediscoveredMap σε αληθή τότε όλα τα κελιά του grid καθώς και τα αντικείμενα που βρίσκονται σε αυτά είναι ήδη γνωστά στον πράκτορα κατά την αρχικοποίηση.

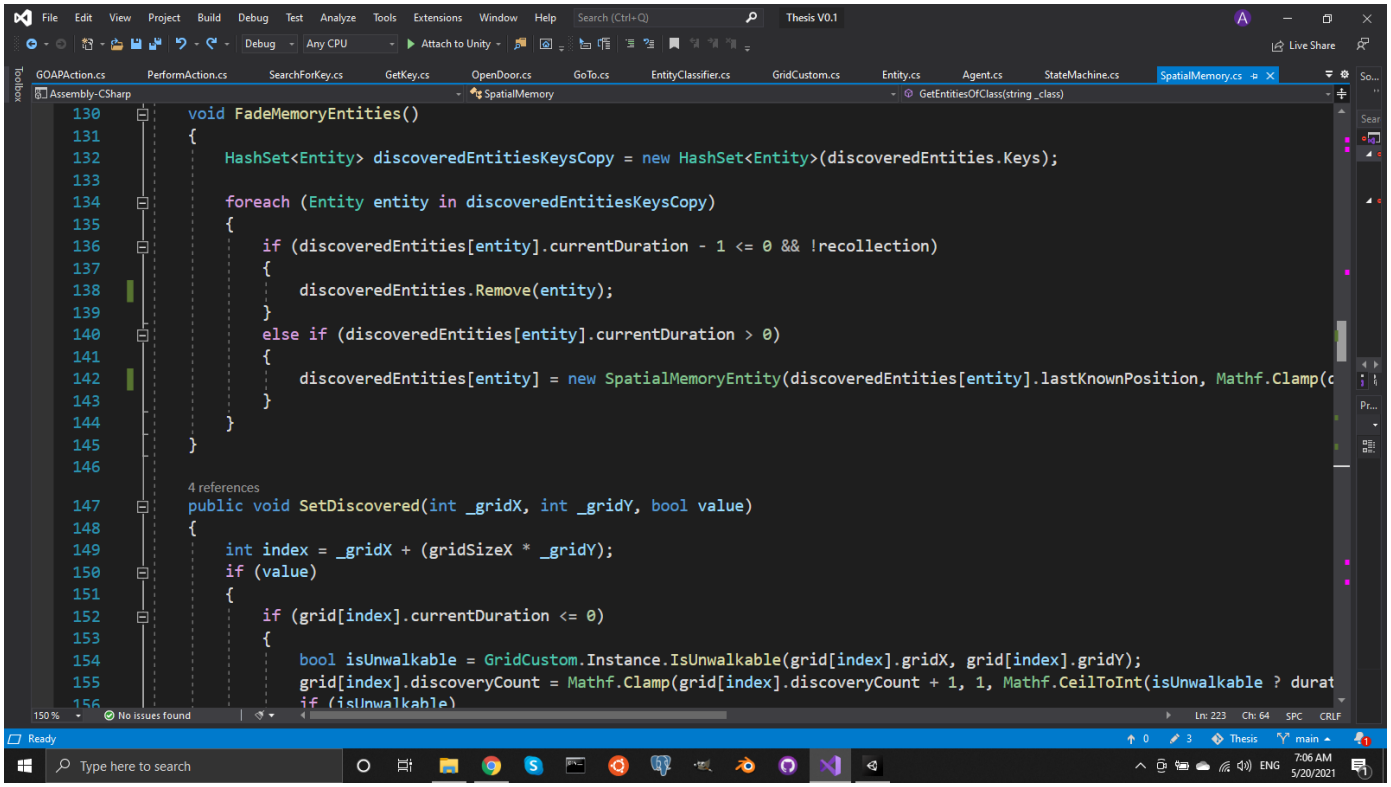


Μετά ξεκινάει να καλεί την μέθοδο *UpdateSpatialMemory* μέσω μιας coroutine η οποία κάθε δευτερόλεπτο που περνάει ανανεώνει την μνήμη αν ο προγραμματιστής έχει επιλέξει να ξεχνάει ο πράκτορας. Έτσι καλεί τις δύο μεθόδους *FadeMemoryNodes* και *FadeMemoryEntities* που είναι υπεύθυνες για να ξεχνάει τα κελιά και τα αντικείμενα αντίστοιχα.



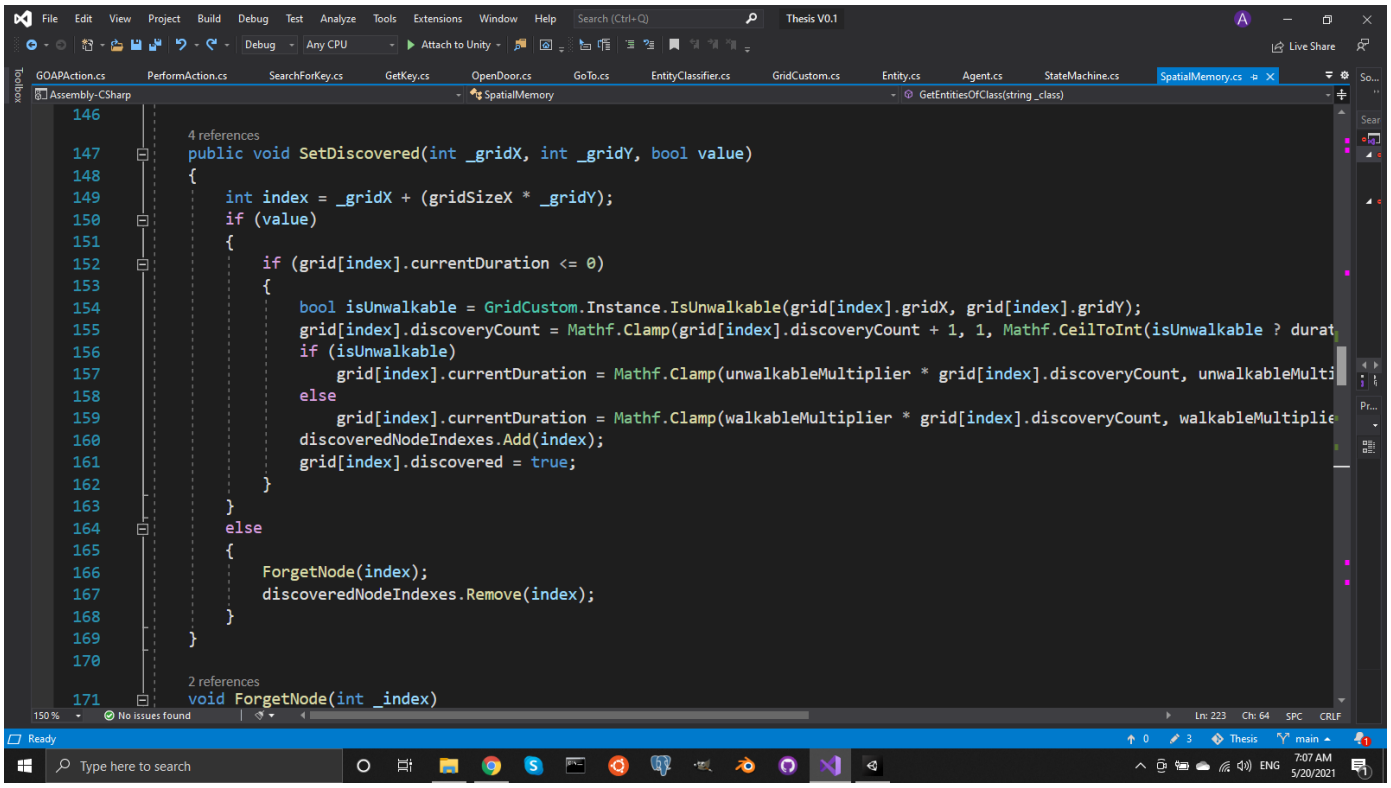


```
105             }
106         }
107     }
108
        1 reference
109     void FadeMemoryNodes()
110     {
111         HashSet<int> IndexesToBeRemoved = new HashSet<int>();
112
113         foreach (int index in discoveredNodeIndexes)
114         {
115             grid[index].currentDuration -= 1;
116
117             if (grid[index].currentDuration <= 0)
118             {
119                 ForgetNode(index);
120                 IndexesToBeRemoved.Add(index);
121             }
122         }
123
124         foreach (int index in IndexesToBeRemoved)
125         {
126             discoveredNodeIndexes.Remove(index);
127         }
128     }
129
        1 reference
130     void FadeMemoryEntities()
```

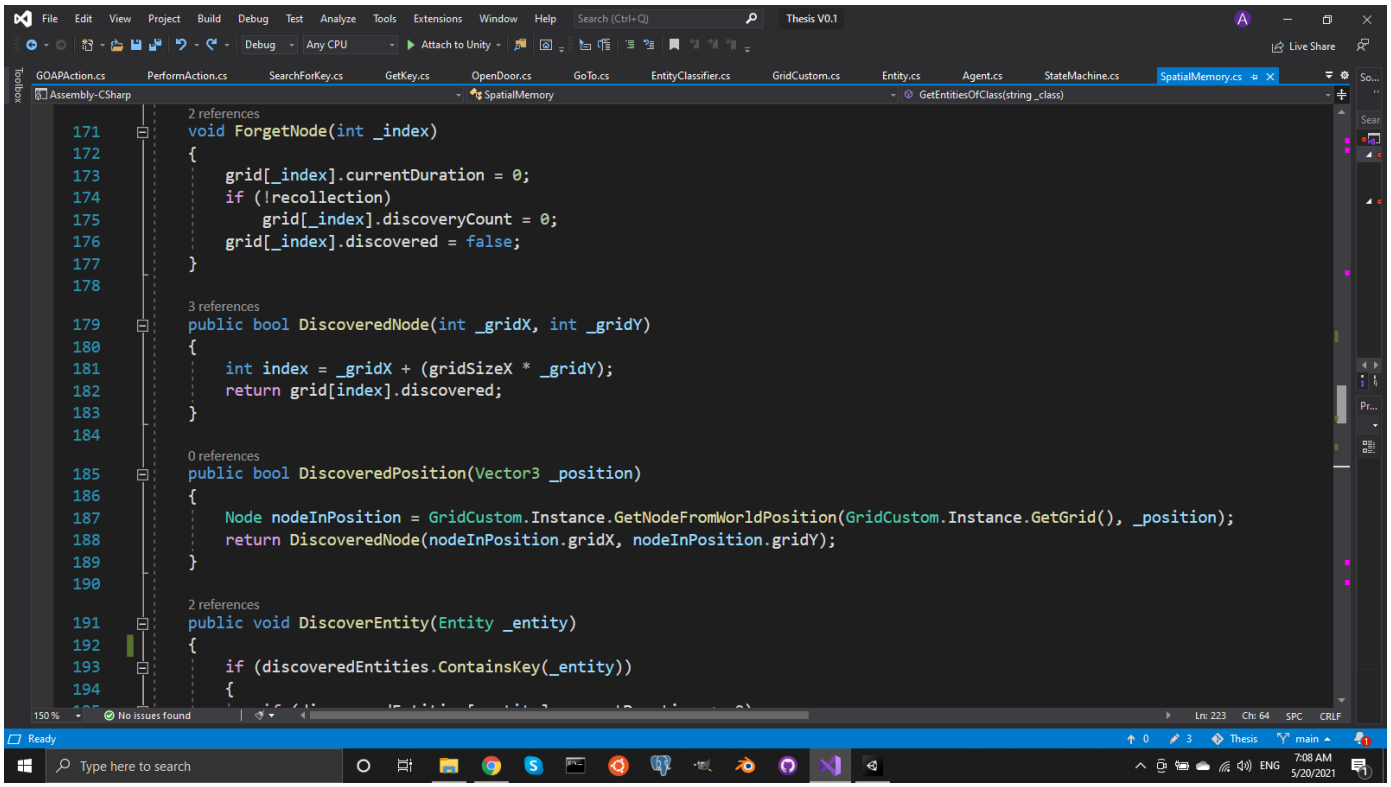


```
130     void FadeMemoryEntities()
131     {
132         HashSet<Entity> discoveredEntitiesKeysCopy = new HashSet<Entity>(discoveredEntities.Keys);
133
134         foreach (Entity entity in discoveredEntitiesKeysCopy)
135         {
136             if (discoveredEntities[entity].currentDuration - 1 <= 0 && !recollection)
137             {
138                 discoveredEntities.Remove(entity);
139             }
140             else if (discoveredEntities[entity].currentDuration > 0)
141             {
142                 discoveredEntities[entity] = new SpatialMemoryEntity(discoveredEntities[entity].lastKnownPosition, Mathf.Clamp(c
143             }
144         }
145     }
146
        4 references
147     public void SetDiscovered(int _gridX, int _gridY, bool value)
148     {
149         int index = _gridX + (gridSizeX * _gridY);
150         if (value)
151         {
152             if (grid[index].currentDuration <= 0)
153             {
154                 bool isUnwalkable = GridCustom.Instance.IsUnwalkable(grid[index].gridX, grid[index].gridY);
155                 grid[index].discoveryCount = Mathf.Clamp(grid[index].discoveryCount + 1, 1, Mathf.CeilToInt(isUnwalkable ? durat
156                 if (isUnwalkable)
```

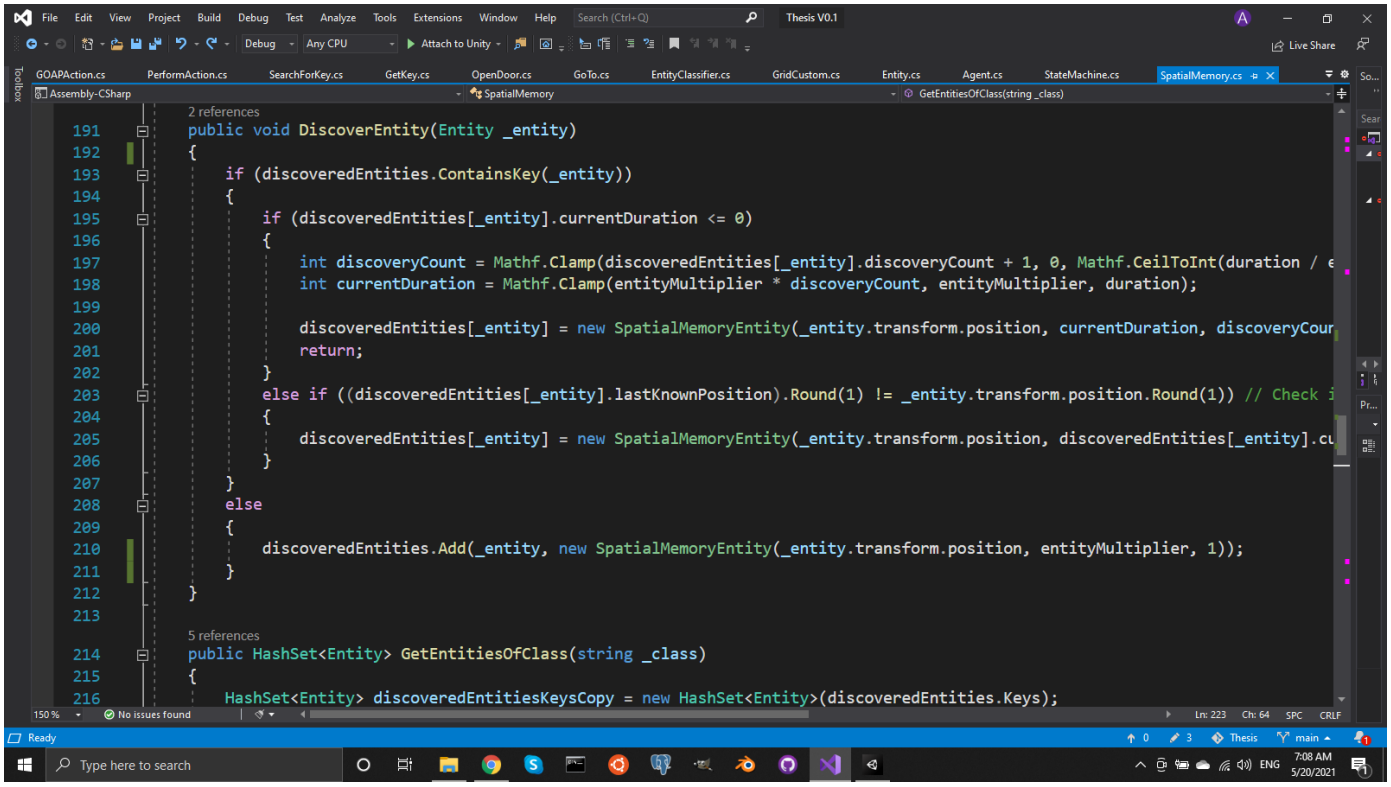
Όταν ο πράκτορας ανακαλύπτει με την όραση του ένα νέο κελί τότε αυτό το κελί προστίθεται σε μία λίστα με κελιά που έχει ανακαλύψει χρησιμοποιώντας την μέθοδο *SetDiscovered*.



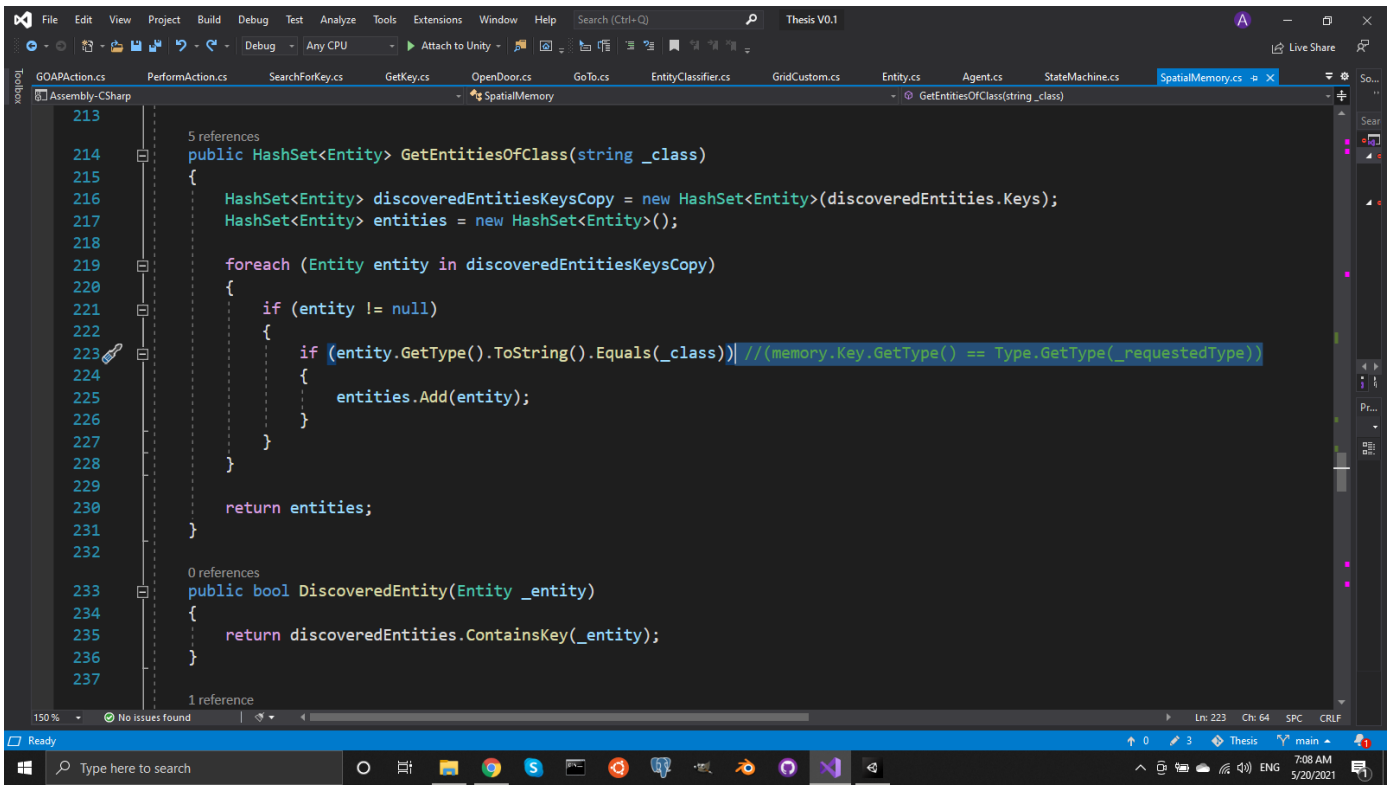
```
146
147     4 references
        public void SetDiscovered(int _gridX, int _gridY, bool value)
148     {
149         int index = _gridX + (gridSizeX * _gridY);
150         if (value)
151         {
152             if (grid[index].currentDuration <= 0)
153             {
154                 bool isUnwalkable = GridCustom.Instance.IsUnwalkable(grid[index].gridX, grid[index].gridY);
155                 grid[index].discoveryCount = Mathf.Clamp(grid[index].discoveryCount + 1, 1, Mathf.CeilToInt(isUnwalkable ? durat
156                 if (isUnwalkable)
157                     grid[index].currentDuration = Mathf.Clamp(unwalkableMultiplier * grid[index].discoveryCount, unwalkableMulti
158                 else
159                     grid[index].currentDuration = Mathf.Clamp(walkableMultiplier * grid[index].discoveryCount, walkableMultiplie
160                 discoveredNodeIndexes.Add(index);
161                 grid[index].discovered = true;
162             }
163         }
164         else
165         {
166             ForgetNode(index);
167             discoveredNodeIndexes.Remove(index);
168         }
169     }
170
171     2 references
        void ForgetNode(int _index)
```



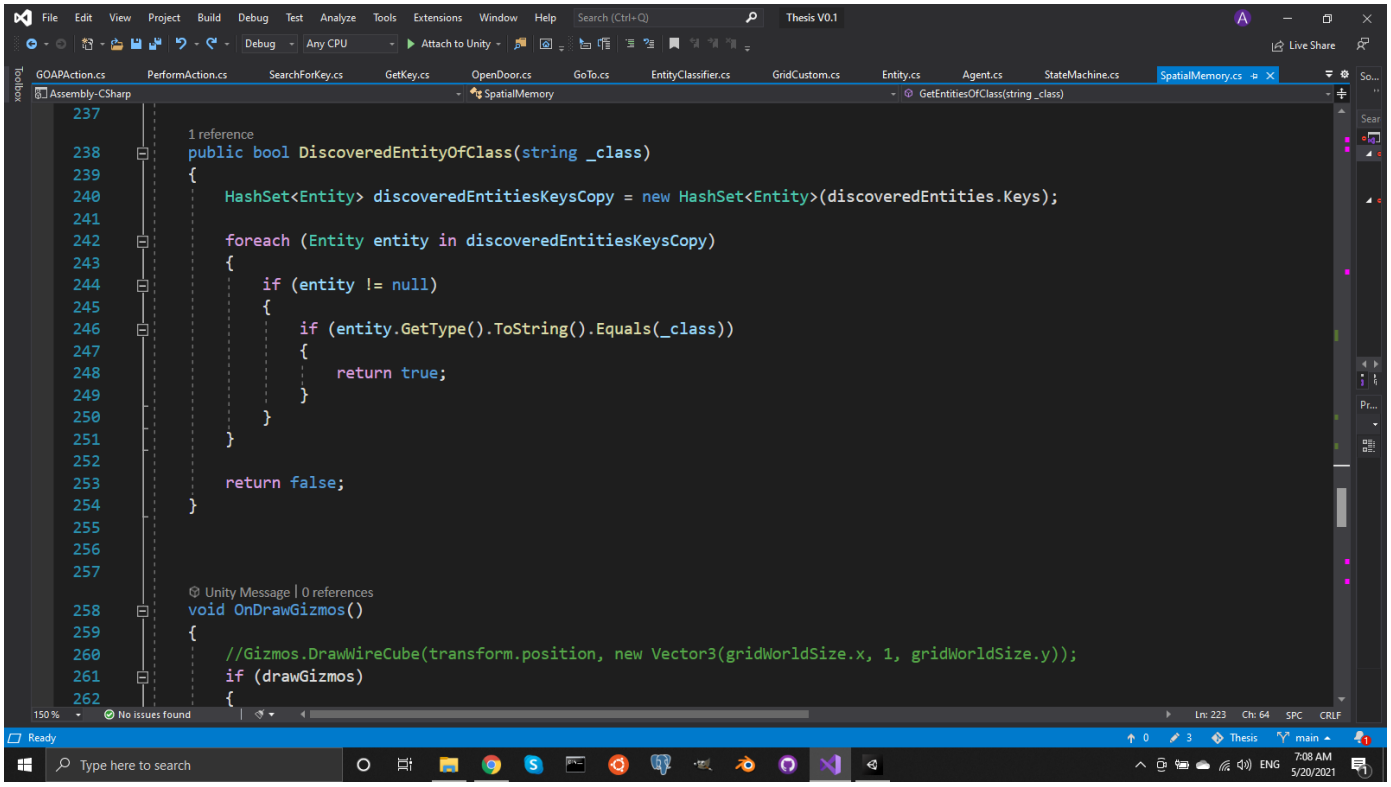
```
171     2 references
        void ForgetNode(int _index)
172     {
173         grid[_index].currentDuration = 0;
174         if (!recollection)
175             grid[_index].discoveryCount = 0;
176         grid[_index].discovered = false;
177     }
178
179     3 references
        public bool DiscoveredNode(int _gridX, int _gridY)
180     {
181         int index = _gridX + (gridSizeX * _gridY);
182         return grid[index].discovered;
183     }
184
185     0 references
        public bool DiscoveredPosition(Vector3 _position)
186     {
187         Node nodeInPosition = GridCustom.Instance.GetNodeFromWorldPosition(GridCustom.Instance.GetGrid(), _position);
188         return DiscoveredNode(nodeInPosition.gridX, nodeInPosition.gridY);
189     }
190
191     2 references
        public void DiscoverEntity(Entity _entity)
192     {
193         if (discoveredEntities.ContainsKey(_entity))
194         {
```



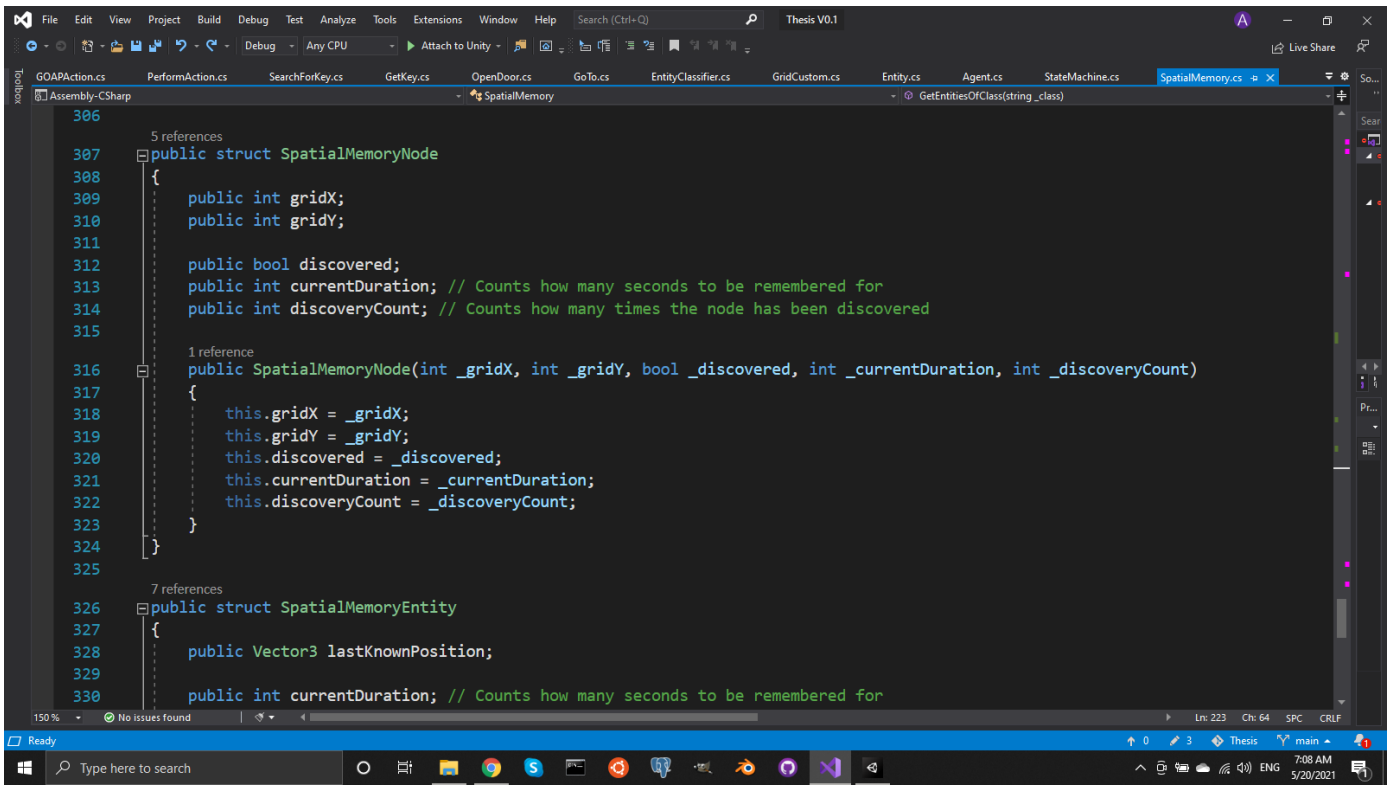
```
2 references
public void DiscoverEntity(Entity _entity)
{
    if (discoveredEntities.ContainsKey(_entity))
    {
        if (discoveredEntities[_entity].currentDuration <= 0)
        {
            int discoveryCount = Mathf.Clamp(discoveredEntities[_entity].discoveryCount + 1, 0, Mathf.CeilToInt(duration / e
            int currentDuration = Mathf.Clamp(entityMultiplier * discoveryCount, entityMultiplier, duration);

            discoveredEntities[_entity] = new SpatialMemoryEntity(_entity.transform.position, currentDuration, discoveryCour
            return;
        }
        else if ((discoveredEntities[_entity].lastKnownPosition).Round(1) != _entity.transform.position.Round(1)) // Check i
        {
            discoveredEntities[_entity] = new SpatialMemoryEntity(_entity.transform.position, discoveredEntities[_entity].cu
        }
    }
    else
    {
        discoveredEntities.Add(_entity, new SpatialMemoryEntity(_entity.transform.position, entityMultiplier, 1));
    }
}

5 references
public HashSet<Entity> GetEntitiesOfClass(string _class)
{
    HashSet<Entity> discoveredEntitiesKeysCopy = new HashSet<Entity>(discoveredEntities.Keys);
```



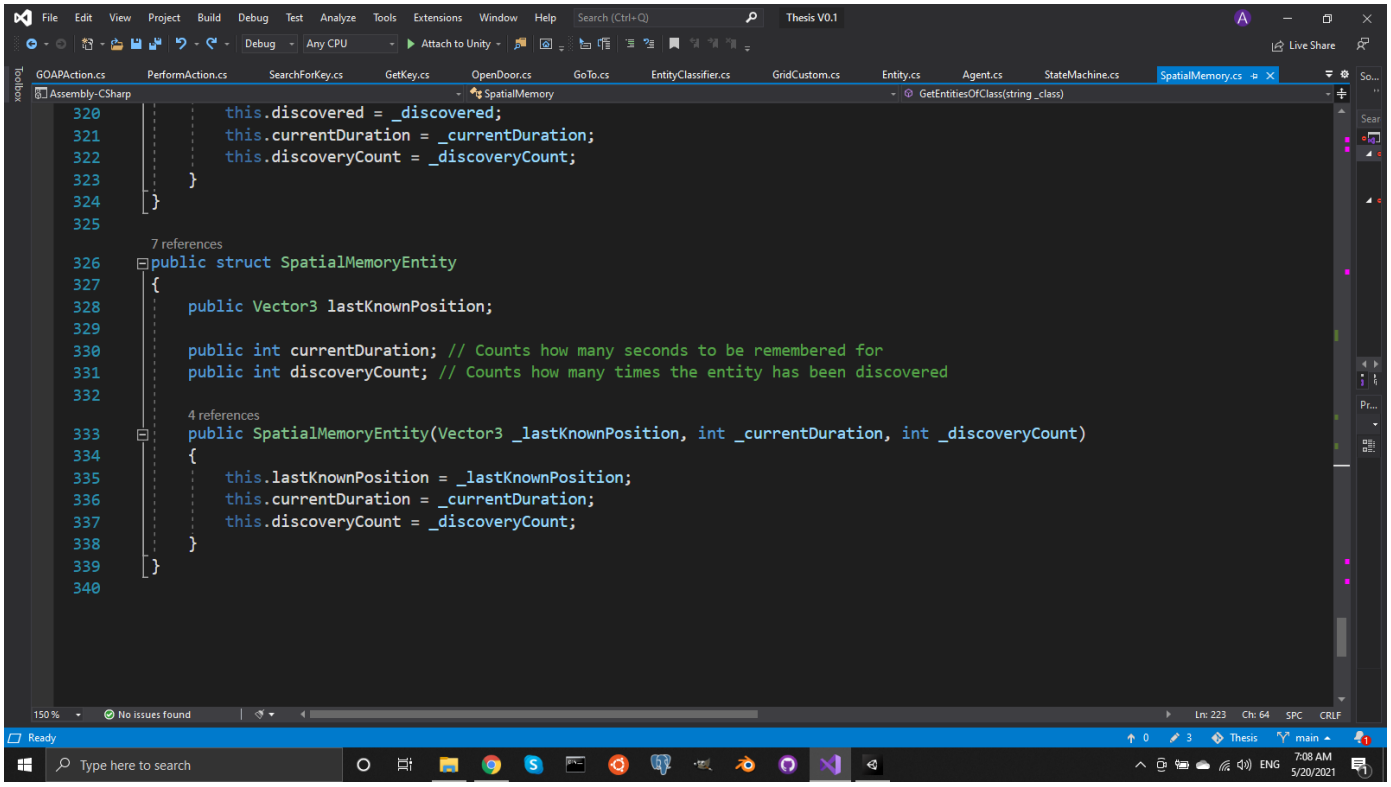
```
5 references
public HashSet<Entity> GetEntitiesOfClass(string _class)
{
    HashSet<Entity> discoveredEntitiesKeysCopy = new HashSet<Entity>(discoveredEntities.Keys);
    HashSet<Entity> entities = new HashSet<Entity>();

    foreach (Entity entity in discoveredEntitiesKeysCopy)
    {
        if (entity != null)
        {
            if (entity.GetType().ToString().Equals(_class)) ///(memory.Key.GetType() == Type.GetType(_requestedType))
            {
                entities.Add(entity);
            }
        }
    }

    return entities;
}

0 references
public bool DiscoveredEntity(Entity _entity)
{
    return discoveredEntities.ContainsKey(_entity);
}

1 reference
```



```
237
        1 reference
238     public bool DiscoveredEntityOfClass(string _class)
239     {
240         HashSet<Entity> discoveredEntitiesKeysCopy = new HashSet<Entity>(discoveredEntities.Keys);
241
242         foreach (Entity entity in discoveredEntitiesKeysCopy)
243         {
244             if (entity != null)
245             {
246                 if (entity.GetType().ToString().Equals(_class))
247                 {
248                     return true;
249                 }
250             }
251         }
252
253         return false;
254     }
255
256
257
        Unity Message | 0 references
258     void OnDrawGizmos()
259     {
260         //Gizmos.DrawWireCube(transform.position, new Vector3(gridWorldSize.x, 1, gridWorldSize.y));
261         if (drawGizmos)
262         {
```



```
306
        5 references
307     public struct SpatialMemoryNode
308     {
309         public int gridX;
310         public int gridY;
311
312         public bool discovered;
313         public int currentDuration; // Counts how many seconds to be remembered for
314         public int discoveryCount; // Counts how many times the node has been discovered
315
        1 reference
316         public SpatialMemoryNode(int _gridX, int _gridY, bool _discovered, int _currentDuration, int _discoveryCount)
317         {
318             this.gridX = _gridX;
319             this.gridY = _gridY;
320             this.discovered = _discovered;
321             this.currentDuration = _currentDuration;
322             this.discoveryCount = _discoveryCount;
323         }
324     }
325
        7 references
326     public struct SpatialMemoryEntity
327     {
328         public Vector3 lastKnownPosition;
329
330         public int currentDuration; // Counts how many seconds to be remembered for
```


File Edit View Project Build Debug Test Analyze Tools Extensions Window Help
Thesis V0.1
Debug
Any CPU
Attach to Unity
Live Share
GOAPAction.cs
PerformAction.cs
SearchForKey.cs
GetKey.cs
OpenDoor.cs
GoTo.cs
EntityClassifier.cs
GridCustom.cs
Entity.cs
Agent.cs
StateMachine.cs
SpatialMemory.cs
Assembly-CSharp
SpatialMemory
GetEntitiesOfClass(string _class)
this.discovered = _discovered;
this.currentDuration = _currentDuration;
this.discoveryCount = _discoveryCount;
}
}
7 references
public struct SpatialMemoryEntity
{
public Vector3 lastKnownPosition;
public int currentDuration; // Counts how many seconds to be remembered for
public int discoveryCount; // Counts how many times the entity has been discovered
4 references
public SpatialMemoryEntity(Vector3 _lastKnownPosition, int _currentDuration, int _discoveryCount)
{
this.lastKnownPosition = _lastKnownPosition;
this.currentDuration = _currentDuration;
this.discoveryCount = _discoveryCount;
}
}
150 %
No issues found
Ln: 223 Ch: 64 SPC CRLF
Ready
Thesis
main
Type here to search
ENG
7:08 AM
5/20/2021

ΣΥΜΠΕΡΑΣΜΑΤΑ

Με το **GOAP**, μπορούμε να δημιουργήσουμε μια μεγάλη σειρά δράσεων χωρίς τον πονοκέφαλο των διασυνδεδεμένων καταστάσεων που συχνά συνοδεύεται από μια μηχανή πεπερασμένων καταστάσεων. Οι δράσεις μπορούν να προστεθούν και να αφαιρεθούν από έναν πράκτορα για την παραγωγή δυναμικών αποτελεσμάτων, καθώς και να κάνουν τον κώδικα πολύ πιο διαχειρίσιμο. Συμπεριλαμβάνοντας και την χωρική μνήμη καταλήγουμε να έχουμε μία ευέλικτη, έξυπνη, δυναμική και πιο ρεαλιστική τεχνητή νοημοσύνη.

ΚΩΔΙΚΑΣ

- GOAPAction
- GOAPAgent
- GOAPPlanner
- IGOAP
- StateMachine
- Idle
- MoveTo
- PerformAction
- GoTo
- SearchForKey
- GetKey
- OpenDoor
- SpatialMemory
- AnimationManager
- Controller
- Door
- Inventory
- Key
- Target
- UIManager
- Agent
- ExtensionMethods
- GridCustom
- Heap
- Node
- Pathfinding
- PathRequestManager
- ShadowCastVisibility
- Entity
- EntityClassifier
- EntityClassifierEditor
- FieldOfView
- FieldOfViewEditor

ΠΗΓΕΣ

Artificial Intelligence and Games, Georgios N. Yannakakis and Julian Togelius
Unity
https://en.wikipedia.org/wiki/Unity_(game_engine)
https://docs.unity3d.com/Manual/index.html
Artificial Intelligence for video games
https://www.youtube.com/watch?v=gm7K68663rA&ab_channel=GDC
https://en.wikipedia.org/wiki/Behavior_tree_(artificial_intelligence,_robotics_and_control)
https://en.wikipedia.org/wiki/Utility_system
Goal Oriented Action Planning
https://www.youtube.com/watch?v=jUSrVF8mve4&ab_channel=Holistic3d
http://alumni.media.mit.edu/~jorkin/goap.html
https://www.youtube.com/watch?v=gm7K68663rA&ab_channel=GDC
https://gamedevelopment.tutsplus.com/tutorials/goal-oriented-action-planning-for-a-smarter-ai--cms-20793
Pathfinding
https://www.youtube.com/watch?v=-L-WgKMFuhE&list=PLFt_AvWsXl0cq5Umv3pMC9SPnKjfp9eGW&ab_channel=SebastianLague
AI State Machine
https://www.youtube.com/watch?v=PaLD1t-kIwM&ab_channel=TheNerdyCanuck
Vision Cone using Shadow Casting
http://www.adammil.net/blog/v125_Roguelike_Vision_Algorithms.html
Field of View:
https://www.youtube.com/watch?v=rQG9aUWarwE&ab_channel=SebastianLague
Best ways to find objects in Unity
https://www.youtube.com/watch?v=Ci1R5AOJ7Hw&ab_channel=JasonWeimann
Cinemachine
https://docs.unity3d.com/Packages/com.unity.cinemachine@2.2/manual/index.html
Mixamo
https://www.mixamo.com/#/