Πανεπιστήμιο Πειραιώς – Τμήμα Πληροφορικής
Πρόγραμμα Μεταπτυχιακών Σπουδών
«Προηγμένα Συστήματα Πληροφορικής»

Μεταπτυχιακή Διατριβή

Τίτλος Διατριβής	**Παραγωγή συμπεριφοράς πρακτόρων με τη χρήση του μηχανισμού επιλογής ενέργειας POSH**
Ονοματεπώνυμο Φοιτητή	**Αικατερίνα Τυροβολά**
Πατρώνυμο	**Βασίλειος**
Αριθμός Μητρώου	**ΜΠΣΠ/ 09010**
Επιβλέπων	**κ. Παναγιωτόπουλος Θεμιστοκλής, Καθηγητής**

Ημερομηνία Παράδοσης **Ιούλιος 2010**

Τριμελής Εξεταστική Επιτροπή

(υπογραφή)	(υπογραφή)	(υπογραφή)
Θεμιστοκλής Παναγιωτόπουλος Καθηγητής	Ευάγγελος Φούντας Καθηγητής	Τσιχριτζής Γεώργιος Καθηγητής

Περίληψη

Σκοπός αυτής της εργασίας είναι η παραγωγή συμπεριφοράς ενός ευφυούς εικονικού πράκτορα με τη χρήση του μηχανισμού επιλογής ενέργειας POSH στο περιβάλλον του Unreal Tournament 2004. Ένας ευφυής εικονικός πράκτορας είναι η αναπαράσταση μιας ενσώματης οντότητας με την ικανότητα αυτόματης συμπεριφοράς στα πλαίσια ενός εικονικού κόσμου. Τμήμα της συμπεριφοράς αποτελεί ο μηχανισμός επιλογής ενέργειας, δηλαδή η διαδικασία επιλογής της επόμενης ενέργειας που θα εκτελέσει ο πράκτορας, με δεδομένη την κατάσταση στην οποία βρίσκεται. Ένας τέτοιος μηχανισμός είναι το POSH (Parallel rooted, Ordered, Slip-stack Hierarchical). Το POSH χρησιμοποιεί προκατασκευασμένα πλάνα, που ουσιαστικά αποτελούν τη δομή της συμπεριφοράς του πράκτορα. Για την πρακτική εφαρμογή και την εξαγωγή συμπερασμάτων χρήσης του POSH ως εργαλείου παραγωγής συμπεριφοράς πρακτόρων δημιουργήθηκε ένα σενάριο κλέφτη και αστυνόμου. Στόχος του αστυνόμου είναι η προστασία του θησαυρού ενώ στόχος του κλέφτη είναι η εύρεση και απόκτηση του θησαυρού. Με βάση αυτό το γενικό σενάριο δημιουργήθηκαν τέσσερις διαφορετικές συμπεριφορές, δύο για τον κάθε ρόλο. Για το ρόλο του κλέφτη δημιουργήθηκε ο θαρραλέος κλέφτης και ο οπλισμένος κλέφτης, ενώ για το ρόλο του αστυνόμου, ο αστυνόμος κυνηγός και ο αστυνόμος φύλακας. Στο τέλος, έγινε μια σύγκριση των αποτελεσμάτων για το ποια συμπεριφορά ήταν η πιο ιδανική για τον κάθε ρόλο. Η υλοποίηση της εργασίας στηρίχτηκε στη χρήση αρκετών πακέτων λογισμικού. Για τη δημιουργία των συμπεριφορών των πρακτόρων χρησιμοποιήθηκε η πλατφόρμα Pogamut, η οποία αποτελεί ένα plugin στο Netbeans, σε συνδυασμό με το maven που η παρουσία του είναι απαραίτητη για την σωστή λειτουργία του Pogamut. Για τη σύνδεση του Pogamut με το περιβάλλον του Unreal Tournament 2004, που αποτελεί τον εικονικό κόσμο, απαραίτητη ήταν η ύπαρξη των Gamebots. Για την υλοποίηση του σεναρίου, αναγκαία ήταν η δημιουργία ενός χάρτη, καθώς και ενός νέου είδους παιχνιδιού, κάτι που απαιτούσε αρκετές τροποποιήσεις στο Unreal Tournament μέσω της Unreal Script. Το τμήμα αυτό της εργασίας, που απαιτούσε τη χρήση και τη συνένωση όλων αυτών των προγραμμάτων μεταξύ τους, ώστε να επικοινωνούν, αποτέλεσε ένα δύσκολο, αλλά συγχρόνως και πολύ ενδιαφέρον εγχείρημα.

Λέξεις κλειδιά: Unreal Tournament 2004 (UT2004), POSH, Pogamut, Unreal Script, Game-Bots, Μηχανισμός επιλογής ενέργειας, πλάνα, συμπεριφορά, ευφυής εικονικός πράκτορας, εικονικό περιβάλλον.

Abstract

The purpose of this project is the development of an intelligent Virtual agent's behavior using POSH action selection mechanism in the Unreal Tournament 2004. An Intelligent Virtual Agent is the representation of an embodied entity with the capacity of autonomous behavior in the context of a virtual world. Action selection is a basic part of an agent's behavior and is the means by which an autonomous agent solves the ongoing problem of choosing what to do next. One mode of action selection is POSH (Parallel rooted, Ordered, Slip-stack Hierarchical). POSH uses precast plans that constitute the structure of an agent's behavior. For the practical implementation and for drawing conclusions using POSH as action selection mechanism, a thief – police officer scenario was created. The thief's goal is to seek for and acquire the treasure while the police officer's goal is to protect it. According to this basic scenario four different behaviors were created, two for each role. For the role of thief two behaviors were implemented, a brave unarmed thief and an armed one, while for the police officer's role a hunter and a guard. Finally, the results of the combination of all four different behaviors were compared to determine which behavior was the most appropriate for each role. For the implementation of this project the use of several software packages was necessary. First of all, the platform of Pogamut (a plugin of Netbeans) was used for developing the behaviors. An important program for running

Pogamut is maven. In addition, Gamebots were used for connecting Pogamut with Unreal Tournament 2004, which was the virtual environment. For the scenario's implementation, the creation of a new map was necessary, along with a new game mode, which required a variety of changes in UT2004 using Unreal Script. The part of the project that required those programs to co-operate with each other, constituted a difficult challenge, but an interesting one to deal with nevertheless.

Keywords: Unreal Tournament 2004 (UT2004), POSH, Pogamut, Unreal Script, GameBots, Action Selection, plans, Behavior, Intelligent Virtual Agent, Virtual Environment.

Ευχαριστίες

Θα ήθελα να ευχαριστήσω τον επιβλέποντα της διπλωματικής μου εργασίας κ. Θεμιστοκλή Παναγιωτόπουλο, για την καθοδήγηση και την υπομονή του, όπως επίσης και τον Νίκο Αβραντινή για την πολύτιμη βοήθειά του και για το χρόνο που μου αφιέρωσε. Τέλος, θα ήθελα να ευχαριστήσω τους προαναφερθέντες για τη δυνατότητα που μου έδωσαν να παρουσιάσω την εργασία μου στα πλαίσια του προπτυχιακού μαθήματος «Κατανεμημένη Τεχνητή Νοημοσύνη».

Περιεχόμενα

2.Ευφυείς εικονικοί πράκτορες (Intelligent Virtual Agents- IVAs) και Ευφυή εικονικά περιβάλλοντα (Intelligent Virtual Enviroments-IVEs) 9

3.Παραγωγή συμπεριφοράς σε ευφυείς εικονικούς πράκτορες....10

3.1 Μηχανισμός Επιλογής Ενέργειας (Action Selection)....11

3.1.1.Environmental Determinism....12

3.1.2.Finite State Machine....13

3.1.3.Basic Reactive Plans....15

3.1.4. POSH Reactive Plans....18

3.2.Ένα απλό παράδειγμα των στοιχείων του POSH.....22

3.3.Ένα παράδειγμα μιας πλήρους ιεραρχίας POSH και επεξήγηση της ακολουθίας των βημάτων του πλάνου.....24

3.4.Behaviour Oriented Design (BOD)....26

3.4.1. Κύκλος ανάπτυξης του BOD (BOD Development Cycle)....27

4.Pogamut....28

4.1.Ιστορία ανάπτυξης του Pogamut....30

4.2.Γενικά πώς λειτουργεί το Pogamut3....30

5.Το Περιβάλλον του Unreal Tournament 2004....31

5.1.Gamebots....32

6.Επικοινωνία Pogamut-UT2004 μέσω GaviaLib-Gamebots2004....33

7.Παραγωγή συμπεριφοράς πρακτόρων σε ρόλους αστυνόμου και κλέφτη.....35

7.1.Υλοποίηση Κόσμου....36

7.2.Δημιουργία κανόνων παιχνιδιού....38

7.3.Ρόλος κλέφτη, 1η συμπεριφορά....38

7.4.Ρόλος κλέφτη, 2η συμπεριφορά....43

7.5.Ρόλος αστυνόμου, 1η συμπεριφορά....45

7.6.Ρόλος αστυνόμου, 2η συμπεριφορά.....49

8. Συμπεράσματα πειράματος....51

9.Ανοιχτά Ερευνητικά Πεδία 52

10. Βιβλιογραφία 54

11.Παράρτημα : Οδηγίες εγκατάστασης της εφαρμογής 56

1.Οδηγίες Εγκατάστασης Pogamut 56

2. Δημιουργία SPOSH Agent 58

3. Σύνδεση στο UT2004 67

12.Παράρτημα : Δημιουργία νέου τύπου παιχνιδιού στο UT2004 67

Πίνακας Εικόνων

Εικόνα 1: Environmental Determinism : Απαριθμεί τις πιθανές καταστάσεις του περιβάλλοντος και αναφέρει τι ενέργειες μπορούν να γίνουν στην κάθε μια κατάσταση. Το παράδειγμα του Conway : Game of Life (Gardner, 1970). 12

Εικόνα 2: Finite State Machines : Απαριθμεί κάθε πιθανή κατάσταση στην οποία μπορεί να βρεθεί ο πράκτορας , τις ενέργειες που μπορεί να κάνει και τα περιβαλλοντικά απρόοπτα που μπορούν να κάνουν τον πράκτορα να αλλάξει κατάσταση. Το παράδειγμα είναι το ίδιο με το (Game of Life) που αναφέρθηκε πιο πάνω. 13

Εικόνα 3: Παράδειγμα Finite State Machine χωρίς τις μεταβάσεις. Ο διπλός κύκλος σημαίνει το τερματικό στάδιο όπου και τερματίζεται η επιλογή δράσης. 14

Εικόνα 4: Παράδειγμα Finite State Machine χωρίς τις μεταβάσεις. 15

Εικόνα 5: Δομή πλάνου POSH και πορεία που ακολουθείται κατά την εκτέλεσή του. ...22

Εικόνα 6 : επικοινωνία μεταξύ του εικονικού κόσμου και της λογικής του πράκτορα μέσω GaviaLib.....30

Εικόνα 7: Η υψηλού επιπέδου αρχιτεκτονικής του Pogamut 3 που είναι ενσωματωμένο με τα GB2004 και το UT2004. Παράδειγμα επικοινωνίας.....33

Εικόνα 8 : Διαδικασία επικοινωνίας για τη δημιουργία του πράκτορα σε ένα παιχνίδι. Δεξιά με τα μπλε γράμματα φαίνονται οι συναρτήσεις σε Java που δημιουργούν τις αντίστοιχες εντολές που στέλνει το Pogamut (Java-Bot) στο GB2004.....35

Εικόνα 9: Εικόνα του χάρτη από ψηλά. 36

Εικόνα 10 : Εικόνα του χάρτη από την οπτική γωνία του παίχτη.....37

Εικόνα 11 : Εικόνα του χάρτη από την οπτική γωνία του παίχτη. Στο βάθος φαίνεται ο θησαυρός(medkit ή mini health pack) 37

Εικόνα 12 : Εικόνα εκκίνησης Unreal Tournament 2004 για την επιλογή του είδους του παιχνιδιού. 38

Εικόνα 13: Διαδικασία δημιουργίας πλάνου για την πρώτη συμπεριφορά του κλέφτη.Βήμα1. 39

Εικόνα 14: Διαδικασία δημιουργίας πλάνου για την πρώτη συμπεριφορά του κλέφτη.Βήμα2. 39

Εικόνα 15: Διαδικασία δημιουργίας πλάνου για την πρώτη συμπεριφορά του κλέφτη.Βήμα3. 40

Εικόνα 16 : Διαδικασία δημιουργίας πλάνου για την πρώτη συμπεριφορά του κλέφτη.Βήμα4. 40

Εικόνα 17: Παράδειγμα λάθους στο πλάνο. 41

Εικόνα 18 : Διαδικασία δημιουργίας πλάνου για την δεύτερη συμπεριφορά του κλέφτη.Βήμα1. 43

Εικόνα 19 : Διαδικασία δημιουργίας πλάνου για την δεύτερη συμπεριφορά του κλέφτη.Βήμα2. 44

Εικόνα 20 : Διαδικασία δημιουργίας πλάνου για την δεύτερη συμπεριφορά του κλέφτη.Βήμα3. 44

Εικόνα 21: Διαδικασία δημιουργίας πλάνου για την πρώτη συμπεριφορά του αστυνόμου.Βήμα1. 45

Εικόνα 22: Διαδικασία δημιουργίας πλάνου για την πρώτη συμπεριφορά του αστυνόμου.Βήμα2. 46

Εικόνα 23: Διαδικασία δημιουργίας πλάνου για την πρώτη συμπεριφορά του αστυνόμου.Βήμα3. 47

Εικόνα 24: Διαδικασία δημιουργίας πλάνου για την πρώτη συμπεριφορά του αστυνόμου.Βήμα4. 48

Εικόνα 25 : Διαδικασία δημιουργίας πλάνου για την πρώτη συμπεριφορά του αστυνόμου.Βήμα5. 49

Εικόνα 26: Πλάνο της δεύτερης συμπεριφοράς του αστυνόμου 50

1.Εισαγωγή

Η ιδέα της δημιουργίας ενός τεχνητού ανθρώπου πάντοτε ερέθιζε, κέντριζε και γοήτευε την ανθρώπινη φύση. Ακόμα και μέσα στα παραμύθια και τις ιστορίες των περασμένων αιώνων, ο άνθρωπος με οδηγό και δημιουργό την καλπάζουσα φαντασία του «κατασκεύασε» ανδρείκελα στα οποία προσέδιδε ανθρώπινη μορφή, χαρακτηριστικά και πάθη, όπως ο Πινόκιο (Κάρλο Κολόντι 1883[1]) ή υπερφυσικές ιδιότητες όπως ο Frankenstein (Mary Shelley 1818[2]).

Τον τελευταίο αιώνα, κάποιες από αυτές τις ιδέες άρχισαν να υλοποιούνται. Αναμφίβολα δύο από τα πιο αξιοσημείωτα μέσα που μπορούν να δώσουν σάρκα και οστά και να κάνουν πράξη αυτές τις φαντασιώσεις είναι τα προγράμματα λογισμικού που περιέχουν κάποιες γνωστικές ικανότητες ανθρώπινου επιπέδου, όπως τα προγράμματα που παίζουν σκάκι, και τα ρομπότ. Πρόσφατα ένα άλλο τέτοιο μέσο εμφανίστηκε στο χώρο: οι ευφυείς εικονικοί πράκτορες (intelligent virtual agents, IVAs) που «ζουν» μέσα στην εικονική πραγματικότητα, ή αλλιώς οι εικονικοί άνθρωποι (virtual humans).

Οι εικονικοί άνθρωποι είναι πράκτορες λογισμικού (Wooldridge, 2002[3]) οι οποίοι κατοικούν σε εικονικούς κόσμους. Εννοιολογικά θεωρείται ότι αποτελούνται από δύο μέρη που επηρεάζει το ένα το άλλο, το εικονικό σώμα (virtual body) και το εικονικό μυαλό (virtual mind). Το εικονικό σώμα υπόκειται στους νόμους του εικονικού κόσμου, όπως για παράδειγμα την βαρύτητα, και έχει προκαθορισμένες ικανότητες αίσθησης και κίνησης. Αυτές οι ικανότητες χρησιμοποιούνται από το εικονικό μυαλό για να ελέγχουν το εικονικό σώμα.

Εικονικός κόσμος είναι η αναπαράσταση που προκύπτει από το αποτέλεσμα της λειτουργίας του εικονικού περιβάλλοντος. Εικονικό περιβάλλον (virtual enviroment) είναι το υπολογιστικό σύστημα που έχει ως στόχο την παρουσίαση των αναπαριστώμενων κόσμων στο χρήστη και την αλληλεπίδραση του χρήστη με αυτούς. Γενικά δεν υπάρχει καθολικά αποδεκτή ορολογία σχετικά με τα εικονικά περιβάλλοντα και τους εικονικούς κόσμους γι' αυτό και σε πολλές περιπτώσεις χρησιμοποιούνται εναλλακτικά αυτοί οι δύο όροι.

Οι εικονικοί άνθρωποι, όπως και τα ρομπότ, χρησιμοποιούνται σε αρκετές βιομηχανικές εφαρμογές. Όμως μέχρι στιγμής ερευνητές από διάφορα πεδία, από την νευροβιολογία έως την τεχνητή νοημοσύνη, χρησιμοποιούν κυρίως τα ρομπότ για να διερευνήσουν τις ιδέες τους σχετικά με τις διάφορες γνωστικές και κινητικές ικανότητες. Οι εικονικοί άνθρωποι δεν χρησιμοποιούνται ακόμα ευρέως, εκτός από τον τομέα της τεχνητής νοημοσύνης των βιντεοπαιχνιδιών, παρόλα αυτά, αυτή η τεχνολογία είναι τώρα αρκετά ώριμη για να αξιοποιηθεί και σε άλλους τομείς.

Γεννάται το ερώτημα: Από πού μπορεί να ξεκινήσει ένας καινούριος επισκέπτης στον τομέα των εικονικών ανθρώπων και κυρίως των ευφυών εικονικών πρακτόρων; Πώς μπορεί να ερευνήσει αυτόν τον τομέα ένας δάσκαλος, ένας μαθητής ή ένας απλά ενθουσιασμένος από αυτόν; Το πρόβλημα είναι ότι η αρχική καμπύλη της μάθησης είναι αρκετά απότομη δηλαδή υπάρχει έλλειψη εκπαίδευσης των νεοεισερχόμενων στον τομέα των ευφυών εικονικών πρακτόρων (Brom et al., 2008). Βασικά, δύο είναι οι τομείς με τους οποίους μπορεί να ασχοληθεί κάποιος για τη δημιουργία ενός εικονικού ανθρώπου, το κομμάτι των γραφικών του πράκτορα (εικονικό σώμα) και το κομμάτι της συμπεριφοράς του (εικονικό μυαλό).

Η παρούσα εργασία πραγματεύεται τη διαδικασία παραγωγής της συμπεριφοράς ενός ευφυούς εικονικού πράκτορα (κεφάλαιο 2 και 3). Κομμάτι της συμπεριφοράς αποτελεί ο μηχανισμός επιλογής ενέργειας, δηλαδή η διαδικασία επιλογής της επόμενης ενέργειας που θα εκτελέσει ο πράκτορας, με δεδομένη την κατάσταση στην οποία βρίσκεται. Υπάρχουν διάφορα είδη μηχανισμού επιλογής ενέργειας, τα οποία αναλύονται στο κεφάλαιο 3. Ο μηχανισμός

[1] http://el.wikipedia.org/wiki/Πινόκιο

[2] http://en.wikipedia.org/wiki/Frankenstein

[3] Ο ορισμός του πράκτορα λογισμικού δίνεται από τον Wooldridge

επιλογής ενέργειας που επιλέχθηκε είναι το POSH (Parallel rooted, Ordered, Slip-stack Hierarchical). Το POSH χρησιμοποιεί δυναμικά πλάνα, που ουσιαστικά αποτελούν τη δομή της συμπεριφοράς του πράκτορα. Το POSH περιγράφεται αναλυτικότερα στα κεφάλαια 3.1.4, 3.2 και 3.3. Για την πρακτική εφαρμογή του POSH δημιουργήθηκε ένα σενάριο κλέφτη και αστυνόμου (κεφάλαιο 7) όπου εξάγονται συμπεράσματα για την χρήση του POSH ως μηχανισμού επιλογής ενέργειας. Στη συνέχεια δημιουργούνται τέσσερις διαφορετικές συμπεριφορές, δύο για κάθε ρόλο, και συγκρίνονται τα αποτελέσματα για το ποια συμπεριφορά είναι η πιο ιδανική για τον κάθε ρόλο.

Ενώ για τη δημιουργία του εικονικού σώματος υπάρχουν αρκετά διαθέσιμα λογισμικά, για την παραγωγή συμπεριφοράς πρακτόρων υπάρχει μια έλλειψη στην ύπαρξη ελεύθερου λογισμικού που να βοηθά στη δημιουργία συμπεριφοράς ευφυών εικονικών πρακτόρων (intelligent virtual agents). Ένα τέτοιο ελεύθερο λογισμικό είναι το Pogamut, που θα περιγραφεί αναλυτικότερα στο κεφάλαιο 4, και το οποίο χρησιμοποιήθηκε για την υλοποίηση της εργασίας. Το Pogamut είναι μια πλατφόρμα λογισμικού για ανάπτυξη συμπεριφοράς ευφυών εικονικών πρακτόρων οι οποίοι ενσωματώνονται σε ένα εικονικό κόσμο (δυσδιάστατο ή τρισδιάστατο). Αρχικά το Pogamut δημιουργήθηκε για να διευκολύνει τη χρήση των ευφυών εικονικών πρακτόρων (Intelligent Virtual Agents), τόσο στην έρευνα όσο και για εκπαιδευτικούς σκοπούς. Από τη μία πλευρά επιτρέπει την γρήγορη ανάπτυξη απλών reactive εικονικών πρακτόρων και από την άλλη διευκολύνει την ανάπτυξη πιο σύνθετων πρακτόρων χρησιμοποιώντας προηγμένες τεχνικές όπως τα νευρωνικά δίκτυα, την ασαφή λογική, τον εξελικτικό προγραμματισμό ή τα πλάνα (POSH).

Για τον εικονικό κόσμο χρησιμοποιήθηκε το περιβάλλον του Unreal Tournament 2004 το οποίο περιγράφεται στο κεφάλαιο 5. Για την σύνδεση και την επικοινωνία των προγραμμάτων μεταξύ τους (κεφάλαιο 6) απαραίτητη ήταν η ύπαρξη των Gamebots (κεφάλαιο 5.1), ενός προγράμματος που ενσωματώνεται στο UT2004 και του δίνει τη δυνατότητα επικοινωνίας με άλλα εξωτερικά προγράμματα (π.χ το Pogamut). Για την υλοποίηση του πειράματος, εκτός από τα παραπάνω προγράμματα, απαραίτητη ήταν και η δημιουργία ενός χάρτη μέσα στον οποίο θα λαμβάνει χώρα το πείραμα, καθώς και ενός νέου είδους παιχνιδιού με δικούς του κανόνες (μέσω Unreal Script). Η διαδικασία σύνδεσης των προγραμμάτων αλλά και η διαδικασία υλοποίησης του σεναρίου περιγράφονται αναλυτικά στα παραρτήματα (κεφάλαια 11 και 12).

2.Ευφυείς εικονικοί πράκτορες (Intelligent Virtual Agents- IVAs) και Ευφυή εικονικά περιβάλλοντα (Intelligent Virtual Enviroments-IVEs)

Τα εικονικά περιβάλλοντα δεν χρησιμοποιούνται μόνο για μια πλοήγηση του χρήστη σε ένα όμορφο τρισδιάστατο εικονικό κόσμο. Όσο ελκυστικός και αν είναι ένας συνθετικός κόσμος, εάν δεν υπάρχουν μεταβολές σε αυτόν χάνεται και το ενδιαφέρον του χρήστη. Για παράδειγμα, οι πόλεις χωρίς ανθρώπους δεν προκαλούν κανένα ενδιαφέρον στο χρήστη εφόσον δεν του δίνουν μια αληθινή αίσθηση πόλης. Βέβαια η απλή παρουσία των ανθρώπων με μια προκαθορισμένη κίνηση μπορεί να βοηθήσει λίγο στη δυναμικότητα του περιβάλλοντος, αλλά δεν προσδίδει καμία ελευθερία αλληλεπίδρασης που θα έπρεπε να έχει ένα εικονικό περιβάλλον για να είναι πιο ελκυστικό προς το χρήστη. Οι εικονικοί κόσμοι γίνονται πιο ελκυστικοί εφόσον υπάρχει το στοιχείο της αυτονομίας και λαμβάνουν χώρα σε αυτούς ενέργειες που δεν προκαλούνται από το χρήστη. Οι αυτόνομες οντότητες σε ένα εικονικό περιβάλλον ονομάζονται εικονικοί πράκτορες. Ένας εικονικός πράκτορας που δεν ακολουθεί προκαθορισμένα βήματα αλλά μπορεί να παίρνει αποφάσεις από μόνος του χρησιμοποιώντας τεχνικές από το χώρο της Τεχνητής Νοημοσύνης για να επιτυγχάνει τους στόχους του, ονομάζεται ευφυής εικονικός πράκτορας. Ευφυή εικονικά περιβάλλοντα ονομάζονται τα εικονικά περιβάλλοντα που περιέχουν ευφυείς εικονικούς πράκτορες.

Η έννοια του ευφυούς εικονικού πράκτορα ξεκίνησε να εμφανίζεται περίπου πριν δύο δεκαετίες. Σήμερα οι περισσότεροι (IVAs) είναι ικανοί να αντιλαμβάνονται σε πραγματικό χρόνο και να δρουν αυτόνομα μέσα σε ένα ευφυές εικονικό περιβάλλον. Αυτά τα περιβάλλοντα είναι κυρίως τρισδιάστατα αλλά και δυσδιάστατα. Οι ευφυείς εικονικοί πράκτορες IVAs μιμούνται μια σειρά ιδιοτήτων που μοιάζουν με αυτές των ανθρώπων, αλλά και των ζώων. Συχνά αντιδρούν συναισθηματικά και κατέχουν διάφορες γνωστικές ικανότητες υψηλού επιπέδου όπως η ικανότητα επικοινωνίας, η κοινωνική ευαισθητοποίηση και η επεισοδιακή μνήμη (Episodic Memory). Επίσης οι ευφυείς εικονικοί κόσμοι είναι δυναμικοί και απρόβλεπτοι, όπως και ο πραγματικός κόσμος και επιτρέπουν μέχρι ένα βαθμό την αλληλεπίδραση μεταξύ των πρακτόρων ή μεταξύ των πρακτόρων και των χρηστών.

Κατά τη διάρκεια των τελευταίων χρόνων, οι IVAs επεκτάθηκαν σε αρκετούς τομείς, όπως σε εμπορικά βιντεοπαιχνίδια (Aylett et al., 2005a), σε εικονικά συστήματα διδασκαλίας (Woolf, 2008), σε θεραπευτικές εφαρμογές (Hodges et al., 2001), σε εικονική αφήγηση (Cavazza et al., 2004), σε εφαρμογές πολιτιστικής κληρονομιάς (Magnenat-Thalmann & Papagiannakis, 2006), στη βιομηχανία του κινηματογράφου (SoftImage, 2009), στην έρευνα της γνωστικής επιστήμης (Burgess, 2002), στο στρατό (Johnson et al., 2004), στην ιατρική εκπαίδευση (Pulse, 2005-9), και σε βιομηχανικές εφαρμογές (Badler et al., 2002; Prendinger & Ishizuka, 2004; Badler et al., 2008; Magnenat-Thalmann & Thalmann, 2004).

Η δομή ενός ευφυούς εικονικού πράκτορα περιέχει πέντε επίπεδα από τα οποία είναι τα εξής:

- Επίπεδο Γραφικών (geometric)
- Επίπεδο Κινηματικής (kinematic)
- Επίπεδο φυσικής αναπαράστασης (physical)
- Επίπεδο συμπεριφοράς (behavioral)
- Γνωστικό επίπεδο (cognitive)

Παρόλο που για τα τρία πρώτα επίπεδα έχει γίνει αρκετή έρευνα και υπάρχει διαθέσιμο πολύ υλικό, τα δύο τελευταία είναι αυτά που έχουν τραβήξει το ενδιαφέρον των ερευνητών τα τελευταία χρόνια. Το επίπεδο συμπεριφοράς αφορά τις ικανότητες του πράκτορα ώστε να αντιδρά με τα ερεθίσματα που δέχεται από το περιβάλλον στο οποίο «ζει», ενώ το γνωστικό επίπεδο αφορά την ικανότητα του πράκτορα να αποκτά γνώση.

Όπως αντιλαμβάνεται κανείς η δημιουργία ενός ευφυούς εικονικού πράκτορα είναι μια αρκετά δύσκολη διαδικασία. Εκτός από τα θέματα των γραφικών, τα οποία αποτελούν ένα κόσμο δικός τους, υπάρχει και η γνωσιακή και η συμπεριφορική πλευρά του προβλήματος. Στην παρούσα εργασία θα ασχοληθούμε με τη δεύτερη πλευρά του προβλήματος, με την παραγωγή συμπεριφοράς εικονικών πρακτόρων και ιδιαίτερα με ένα κομμάτι αυτής, το μηχανισμό επιλογής ενεργειών (action selection).

3.Παραγωγή συμπεριφοράς σε ευφυείς εικονικούς πράκτορες

Η σχεδίαση της συμπεριφοράς ενός πράκτορα είναι μια βασική αλλά δύσκολη διαδικασία. Δεν υπάρχει κάποια σίγουρη και αποδεδειγμένη διαδικασία για το πώς μπορεί να σχεδιαστεί ακριβώς ένας πράκτορας. Πολλά ερωτήματα που έχουν έρθει στην επιφάνεια δεν έχουν ακόμη απαντηθεί. Η σχεδίαση των πρακτόρων επομένως είναι μια διαδικασία που δεν ακολουθεί συγκεκριμένους κανόνες, παρόλα αυτά υπάρχουν κάποια βήματα που μπορούν να βοηθήσουν.

Το πρώτο βήμα είναι ότι δοθέντος ενός σεναρίου πραγματικού κόσμου (π.χ. ένα πολυκατάστημα με πολύ κόσμο) και κάποιων ερωτήσεων πάνω σε αυτό για τις οποίες ψάχνουμε απαντήσεις (π.χ. πώς μπορούμε να μειώσουμε το μέσο χρόνο εκκένωσης σε περίπτωση φωτιάς;) πρέπει να σχεδιαστεί ένα μοντέλο που θα απαντά στο εξής:

Σενάριο + Ερωτήσεις → Σχεδίαση Μοντέλου

Αρκετές είναι οι σημαντικές αποφάσεις που πρέπει να παρθούν για έναν κατάλληλο σχεδιασμό. Μερικές από αυτές είναι: αν θα χρησιμοποιηθεί συγκεκριμένο ή γενικό μοντέλο, το επίπεδο αφαίρεσης του μοντέλου, το γνωστικό περιεχόμενο του πράκτορα και η επιλογή της αρχιτεκτονικής του πράκτορα.

Η παρούσα εργασία ασχολείται με την επιλογή της αρχιτεκτονικής του πράκτορα και συγκεκριμένα με ένα μέρος αυτής που αφορά τη διαδικασία επιλογής των ενεργειών του πράκτορα (action selection).

Όταν αναφερόμαστε σε αρχιτεκτονική πρακτόρων εννοούμε ουσιαστικά μεθοδολογίες σχεδιασμού πρακτόρων. Η ποικιλία των αρχιτεκτονικών που χρησιμοποιούνται αντικατοπτρίζει τη συλλογική γνώση για το ποιες μέθοδοι είναι χρήσιμες όταν προσπαθούμε να χτίσουμε μια νοημοσύνη. Είναι δηλαδή μια συλλογή γνώσης και μεθόδων. Διάφορες τέτοιες αρχιτεκτονικές είναι οι εξής:

- Behavior-Based Design
- Two-and Three layer architectures (Multi-layered)
- PRS (Procedural Reasoning Systems)
- ACT-R
- BOD (Behaviour oriented Design)
- Ymir

Γενικά κάθε αυτόνομη αρχιτεκτονική χρειάζεται να έχει τα εξής:

- Μια δομή που αποτελείται από υποομάδες και μια προσέγγιση της ανάπτυξης της βασικής δομής της συμπεριφοράς του πράκτορα, περιλαμβάνοντας αντίληψη, δράση και εκμάθηση.
- Ένα μέσο που να κατασκευάζει εύκολα ατομικές ικανότητες για πολύπλοκα καθήκοντα. Αυτό προφανώς απαιτεί ένα μηχανισμό που να κανονίζει την επιλογή ενέργειας του πράκτορα.
- Ένα μηχανισμό που να αντιδρά γρήγορα στις αλλαγές του περιβάλλοντος. Αυτό γενικά παίρνει τη μορφή ενός συστήματος που λειτουργεί παράλληλα με την επιλογή ενέργειας, το οποίο παρακολουθεί το περιβάλλον για αξιοπρόσεκτα χαρακτηριστικά ή γεγονότα.

Η αρχιτεκτονική που ακολουθήθηκε στη συγκεκριμένη εργασία είναι η BOD (Behaviour Oriented Design) η οποία περιγράφεται αναλυτικά παρακάτω.

3.1 Μηχανισμός Επιλογής Ενέργειας (Action Selection)

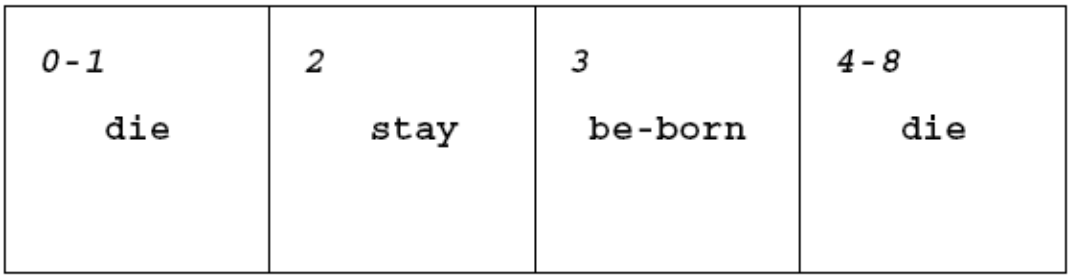
Όπως αναφέρθηκε και προηγουμένως βασικό στοιχείο της αρχιτεκτονικής του πράκτορα που καθορίζει τη συμπεριφορά του, είναι η επιλογή της ενέργειάς του (action selection). Η επιλογή της ενέργειας είναι ο τρόπος με τον οποίο ο πράκτορας λύνει το τρέχον πρόβλημά του διαλέγοντας ποια κίνηση θα κάνει μετά. Η επιλογή ενέργειας είναι το εκτελεστικό μέρος της νοημοσύνης του πράκτορα. Ένα πρόβλημα για την κατανόηση της επιλογής ενέργειας είναι ο καθορισμός του επιπέδου αφαίρεσης που χρησιμοποιείται για τον προσδιορισμό της «ενέργειας». Στο πιο βασικό επίπεδο αφαίρεσης μιας ατομικής ενέργειας θα μπορούσε να είναι οτιδήποτε, από το να συστέλλεται ένα μυϊκό κύτταρο μέχρι του να προκαλείται ένας πόλεμος. Τυπικά για κάθε μηχανισμό επιλογής ενέργειας το σύνολο των δυνατών ενεργειών είναι προκαθορισμένο και σταθερό. Τα είδη που διακρίνονται είναι τα εξής:

3.1.1.Environmental Determinism

Δεν υπάρχει κανένα καθιερωμένο όνομα για το ποιος πραγματικά είναι ο απλούστερος τρόπος για να γίνει η επιλογή ενέργειας γι' αυτό και ονομάζεται «Περιβαλλοντικός Ντετερμινισμός» . Αυτό το μοντέλο υποθέτει ότι:

- Υπάρχει μόνο ένας περιορισμένος αριθμός καταστάσεων όπου μπορεί να βρεθεί ο πράκτορας
- Αυτές οι καταστάσεις αλληλοαναιρούνται και
- Οι δράσεις μπορούν εύκολα να χαρτογραφηθούν σε καταστάσεις.

Παρόλο που αυτές οι υποθέσεις μπορεί να φαίνονται εξωπραγματικές, έχουν χρησιμοποιηθεί σε αρκετά αφηρημένα μοντέλα. Σε αυτό το μηχανισμό πρέπει να απαριθμηθούν οι καταστάσεις στις οποίες μπορεί να βρεθεί ο πράκτορας αλλά και το τι θα κάνει σε κάθε μια από αυτές. Ένα παράδειγμα χρήσης αυτού του μηχανισμού είναι στο Game of Life του Conway (Gardner, 1970), το οποίο είναι ένα ALife σύστημα που λαμβάνει χώρα σε ένα δυσδιάστατο πλέγμα. Κάθε κύτταρο-κελί του πλέγματος είναι ένας πράκτορας. Ο Conway υποστηρίζει ότι υπάρχουν εννέα πιθανές περιβαλλοντικές καταστάσεις και το μόνο πράγμα που καθορίζει την ενέργεια στο σύστημά του είναι το πόσους γείτονες έχει ο κάθε πράκτορας. Μπορεί να έχει από 0 έως 8 γείτονες στο δυσδιάστατο πλέγμα. Η εικόνα παρακάτω συνοψίζει τα παραπάνω σε τέσσερες πιθανές καταστάσεις. Αν το κύτταρο έχει δύο γείτονες τότε διατηρεί την τωρινή κατάσταση δηλαδή ζωντανό ή νεκρό. Αν έχει ακριβώς τρεις γείτονες το κύτταρο είναι ζωντανό ανεξάρτητα από την προηγούμενη κατάστασή του αλλά αν έχει τέσσερις ή παραπάνω γείτονες τότε είναι νεκρό.



Εικόνα 1: Environmental Determinism : Απαριθμεί τις πιθανές καταστάσεις του περιβάλλοντος και αναφέρει τι ενέργειες μπορούν να γίνουν στην κάθε μια κατάσταση. Το παράδειγμα του Conway : Game of Life (Gardner, 1970).

Υποστηρίζεται ότι αυτό το σύστημα είναι ένα αρκετά ρεαλιστικό μοντέλο της βακτηριακής ζωής σε ένα δίσκο καλλιέργειας μικροβίων.

Ο προγραμματισμός του Enviromental Determinism απαιτεί ένα σύνολο από If-then εντολών. Για παράδειγμα:

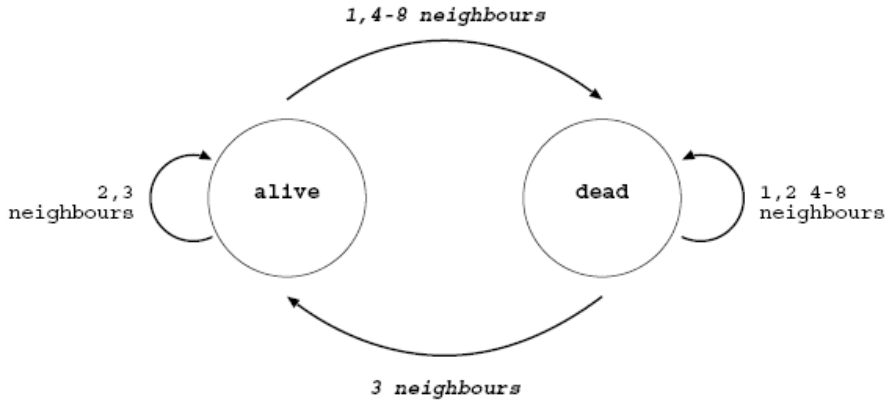
```
if (cell is dead) AND (number-of-neighbors is 3)
then {set cell alive}; /* new cell is born */

if (cell is alive) AND (number-of-neighbors is 2 OR 3)
then {no action}; /* Leave the cell alive */

if (cell is alive AND (number-of-neighbors is NOT 2 OR 3)
then {set cell dead}; /*lonely or over-crowded cells die*/

if (cell is dead) AND (number-of-neighbors is NOT 3)
then {no action}; /* leave cell dead */
```

3.1.2.Finite State Machine

Γενικότερα οι προγραμματιστές προτιμούν να σκέφτονται για τους πράκτορες παρά για το περιβάλλον. Έτσι οι προγραμματιστές βρίσκουν πιο απλό να αναπτύσσουν τον κώδικα γύρω από τις δράσεις και όχι γύρω από τα γεγονότα. Ο πιο συνήθης τρόπος για τον έλεγχο πολλών μηχανημάτων, κυρίως ευφυών εικονικών πρακτόρων για βιντεοπαιχνίδια, είναι ο Finite State Machine (FSM). Αυτός ο μηχανισμός προϋποθέτει τα εξής:

- Απαρίθμηση των καταστάσεων που μπορεί να βρεθεί ένας πράκτορας
- Περιγραφή των αιτιών που θα αναγκάσουν τον πράκτορα να αλλάξει θέση.

Για το μηχανισμό επιλογής ενέργειας στο σύστημα ALife, υποθέτουμε ότι κάθε κατάσταση είναι αυτή για την οποία κάθε πράκτορας μπορεί να εκτελέσει μια συγκεκριμένη ενέργεια. Για το παράδειγμα a Game of Life πρέπει να σκεφτούμε το κύτταρο-πράκτορα να εκφράζει δύο συμπεριφορές, το να είναι ζωντανό ή το να είναι νεκρό. Η τεκμηρίωση του FSM πρέπει να περιλαμβάνει ένα διάγραμμα (Εικόνα 2).



Εικόνα 2: Finite State Machines : Απαριθμεί κάθε πιθανή κατάσταση στην οποία μπορεί να βρεθεί ο πράκτορας , τις ενέργειες που μπορεί να κάνει και τα περιβαλλοντικά απρόοπτα που μπορούν να κάνουν τον πράκτορα να αλλάξει κατάσταση. Το παράδειγμα είναι το ίδιο με το (Game of Life) που αναφέρθηκε πιο πάνω.

Ο κώδικας πρέπει να περιλαμβάνει τις πιθανές μεταβάσεις, ομαδοποιημένες από κάθε κατάσταση:

```
/* Μεταβάσεις από την κατάσταση  DEAD */
 if (self dead) AND (number-of-neighbors is 3)
then (self be-born);

/* Μεταβάσεις από την κατάσταση  ALIVE */
if (self alive) AND ((number-of-neighbors NOT in {2,3})
then (self die);
```

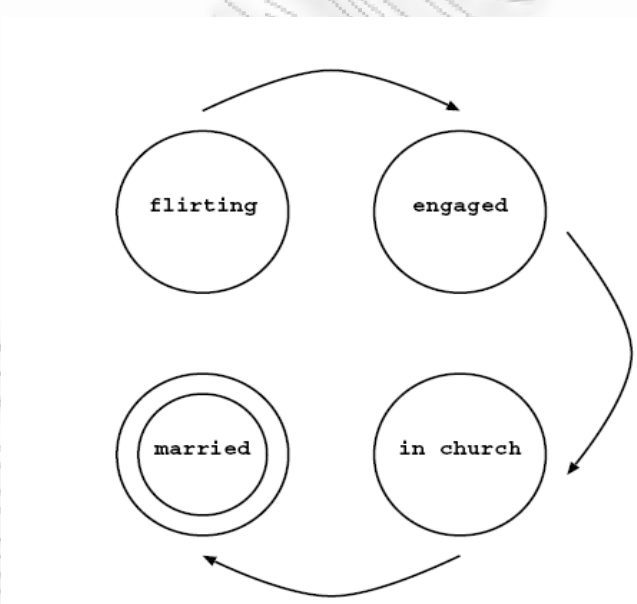
Για να είμαστε πιο τυπικοί, ένα Finite State Machine θα πρέπει επίσης να απαριθμεί όλα τα πιθανά περιβαλλοντικά γεγονότα, και να προσδιορίζει τις περιπτώσεις στις οποίες δεν πραγματοποιείται καμία αλλαγή.

Πρέπει να γίνει αντιληπτό ότι και στον περιβαλλοντικό ντετερμινισμό και στα Finite State Machines ο προγραμματιστής χρειάζεται να δημιουργήσει διακριτές κατηγορίες για τα περιβαλλοντικά γεγονότα και τις ενέργειες της συμπεριφοράς του πράκτορα. Οι μοναδικές διαφορές είναι ότι στα FSM ο προγραμματιστής χρειάζεται να απαριθμήσει τις καταστάσεις που μπορεί να βρεθεί ένας πράκτορας και ότι οι ενέργειες που μπορεί να κάνει ο πράκτορας είναι συνδεδεμένες με αυτές τις κατάστάσεις, και όχι με τις περιβαλλοντικές καταστάσεις.

Οι περισσότεροι πράκτορες, φυσικά, έχουν περισσότερες από δύο δυνατές ενέργειες. Εάν υπάρχει συγκεκριμένος τρόπος της μετάβασης από κάθε ενέργεια στην επόμενη, τότε τα FSM είναι ένας καλός τρόπος περιγραφής της συμπεριφοράς. Ωστόσο εάν η συμπεριφορά επηρεάζεται σε μεγάλο βαθμό από το περιβάλλον και το περιβάλλον είναι δυναμικό και απρόβλεπτο, τότε μπορεί να υπάρχουν μεταβάσεις από κάθε κατάσταση σε οποιαδήποτε άλλη κατάσταση. Έτσι σε κάθε καινούρια ικανότητα που θα προστίθεται στον πράκτορα θα πρέπει να προστίθενται και οι αντίστοιχες μεταβάσεις για κάθε κατάσταση και ενέργεια. Αυτό είναι αρκετά πολύπλοκο και χρονοβόρο αν λάβουμε υπόψη μας ότι ο αριθμός των μεταβάσεων (και επομένως το μέγεθος του κώδικα) αυξάνεται εκθετικά, αφού για Ν κόμβους πρέπει να υπάρχουν Ν-1 μεταβάσεις σε αυτούς. Θα ήταν καλύτερο να υπάρχει ένας τρόπος να προγραμματίζουμε το μηχανισμό επιλογής ενέργειας έτσι ώστε ο κώδικάς του να μην μεγαλώνει πιο γρήγορα από το αριθμό των πιθανών ενεργειών.

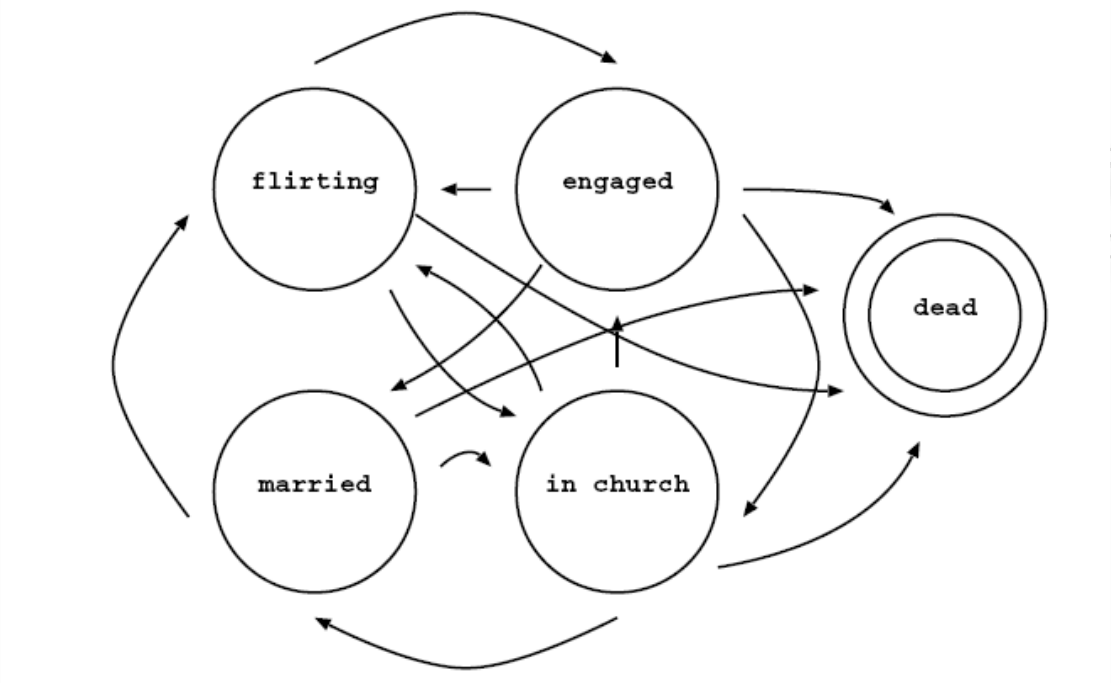
Παρακάτω ακολουθεί ένα παράδειγμα όπου οι πράκτορες έχουν ένα πιο ανθρωποειδές είδος νοημοσύνης και περνούν τα εξής στάδια: φλερτ, αρραβώνας, στην εκκλησία και γάμος.

Ένα FSM για τον πράκτορα είναι το ακόλουθο χωρίς τις μεταβάσεις μεταξύ των καταστάσεων:



Εικόνα 3: Παράδειγμα Finite State Machine χωρίς τις μεταβάσεις. Ο διπλός κύκλος σημαίνει το τερματικό στάδιο όπου και τερματίζεται η επιλογή δράσης.

Το πρόβλημα ανακύπτει, όταν προσπαθούμε να επισημάνουμε τις μεταβάσεις. Για παράδειγμα κάποιος θα μπορεί να πάει στην εκκλησία σε οποιοδήποτε στάδιο της ζωής του και όχι μόνο για να παντρευτεί. Επίσης ο γάμος μπορεί να μην είναι το τερματικό στάδιο για κάποιον. Μερικοί μπορεί να συνεχίζουν να φλερτάρουν και μπορούν να αρραβωνιαστούν ξανά. Στην πραγματικότητα το τερματικό στάδιο είναι ο θάνατος που μπορεί να έρθει σε οποιοδήποτε στάδιο (εικόνα 4).



Εικόνα 4: Παράδειγμα Finite State Machine χωρίς τις μεταβάσεις.

Το πρόβλημα με τα FSM είναι ότι πρέπει να αναπαραστήσουν όλες τις πιθανές μεταβάσεις για τον πράκτορα. Αλλά για τη δημιουργία ενός πράκτορα που να λειτουργεί σωστά πρέπει να καθοριστούν μόνο οι μεταβάσεις τις οποίες κάνει μόνος του ο πράκτορας. Στην πραγματικότητα στην Τεχνητή Νοημοσύνη για αφηρημένες προσομοιώσεις, γενικά το μόνο που χρειαζόμαστε είναι να αναπαράγουμε τις μεταβάσεις τις οποίες ο πράκτορας λογικά θα διαλέξει για την επιδίωξη των στόχων του.

3.1.3.Basic Reactive Plans

Τα Basic Reactive Plans (BRP) είναι ένα είδος αντιδραστικού σχεδιασμού (reactive planning). Το reactive planning ή αλλιώς Dynamic planning είναι ένας μηχανισμός επιλογής ενέργειας ενός πράκτορα που βρίσκεται σε μια κατάσταση σύμφωνα με τον οποίον ένα σύστημα διαλέγει τι ενέργεια θα κάνει βασισμένο σε μια υπάρχουσα δομή δεδομένων. Αυτό που κάνει το σύστημα δυναμικό είναι ότι η απόφαση αυτή βασίζεται στην τωρινή κατάσταση του περιβάλλοντος, την οποία αντιλαμβάνεται ο πράκτορας μέσα από τους αισθητήρες του. Όμως αν ο πράκτορας αντιδρά μόνο στις αλλαγές του περιβάλλοντος μπορεί να υπάρξει αναποφασιστικότητα μεταξύ των στόχων. Έτσι ένα δυναμικό πλάνο χρησιμοποιεί ένα είδος εσωτερικής κατάστασης που παρακολουθεί τους τωρινούς στόχους του πράκτορα.

Έτσι ένα BRP απαιτεί μερικές παραδοχές. Για παράδειγμα απαιτεί ότι ο πράκτορας να έχει κάποιο στόχο και κάποιο σύνολο ενεργειών που μπορεί να εκτελέσει, ώστε να τον οδηγήσουν να πετύχει το στόχο αυτό. Το BRP είναι μια λίστα προτεραιότητας των πράξεων-ενεργειών αυτών. Η πιο σημαντική ενέργεια είναι αυτή με την οποία πετυχαίνει το στόχο, η δεύτερη πιο σημαντική είναι αυτή που ενεργοποιεί την πιο σημαντική πράξη, και ούτω καθεξής.

Εάν ο πράκτορας μπορεί να εκτελέσει την πιο σημαντική πράξη, τότε δεν χρειάζεται να εκτελέσει τις υπόλοιπες. Ένας αντιδραστικός σχεδιασμός μπορεί να αναγνωρίσει αυτή την κατάσταση, κάνοντας τον πράκτορα να συμπεριφέρεται ανάλογα με τις δεδομένες συνθήκες επιλέγοντας από το σύνολο των δράσεων που έχει στη διάθεσή του μόνο αυτές που είναι χρήσιμες στις δεδομένες καταστάσεις. Αυτό σημαίνει ότι κάθε ενέργεια συνδέεται όχι μόνο με

την προτεραιότητα αλλά και με μια αντιληπτική συνθήκη η οποία της επιτρέπει να γνωρίζει το πότε μπορεί να εκτελεστεί η ενέργεια αυτή. Αυτό είναι σημαντικό και για τον καιροσκοπισμό (παρακάμπτοντας τα περιττά βήματα σε ένα συμβατικό σχέδιο) αλλά και για την ευρωστία (πιθανώς επαναλαμβάνοντας τα βήματα εάν αποτύχουν ή αν πρέπει να επαναληφτούν όπως το σκάψιμο ενός πηγαδιού μέχρι να βρεθεί νερό).

Αντί να χρειάζεται να υπάρχει ακριβής κώδικας για το πώς να αναγνωρίζεται κάθε πιθανή μετάβαση μεταξύ ενεργειών, ένας BRP προγραμματιστής χρειάζεται μόνο τον κώδικα για το πώς θα αναγνωρίζεται η κατάσταση στην οποία κάθε ενέργεια μπορεί να πυροδοτηθεί, μια δουλειά που γίνεται απλούστερη όταν καμία καλύτερη (υψηλότερης προτεραιότητας) ενέργεια δε μπορεί να πυροδοτηθεί. Αυτό σημαίνει ότι για κάθε ενέργεια, ο προγραμματιστής πρέπει να προγραμματίσει μόνο μια κατάσταση: τις ελάχιστες απαιτήσεις όπου η ενέργεια μπορεί να φανεί χρήσιμη. Δεν υπάρχει λόγος να περιγραφεί το πότε μια ενέργεια θα πρέπει να παραλειφθεί, διότι αυτό είναι κωδικοποιημένο από την προτεραιότητα μέσα στο BRP.

Για να γίνουν αυτά πιο κατανοητά έστω ότι στο προηγούμενο παράδειγμα υποθέτουμε ότι ο πράκτορας είναι ¨αγνός¨ και χωρίς πονηρές προθέσεις. Στόχος του είναι να παντρευτεί. Για να παντρευτεί ο πράκτορας πρέπει να είναι στην εκκλησία, αλλά δεν υπάρχει λόγος να πάει στην εκκλησία χωρίς να έχει αρραβωνιαστικιά και δεν μπορεί να έχει αρραβωνιαστικιά χωρίς να έχει αρραβωνιαστεί. Εάν δεν έχει αρραβωνιαστεί πρέπει πρώτα να φλερτάρει. Το φλερτ έχει και αυτό προαπαιτούμενα, αλλά για λόγους απλότητας σταματάμε εδώ.

Παρακάτω φαίνεται το BRP του παραδείγματος :

1*:*	(fiancé present AND in church)	→	marry
2*:*	(fiancé present)	→	go to church
3*:*	(engaged)	→	go to fiancé
4:	(receiving attention)	→	become engaged
5:	()	→	flirt

Οι αριθμοί στα αριστερά δείχνουν τη σειρά προτεραιότητας, με το 1 να αντιστοιχεί στην υψηλότερη. Έτσι αν βρίσκεται στην εκκλησία και δεν υπάρχει παρούσα η αρραβωνιαστικιά δεν μπορεί να παντρευτεί. Η αρραβωνιαστικιά μπορεί να υπάρχει αλλά να μη βρίσκεται εκείνη τη στιγμή μαζί του. Αν δεν υπάρχει καθόλου αρραβωνιαστικιά εννοείται ότι δεν μπορεί να παντρευτεί αλλά μπορεί να φλερτάρει. Εκτός φυσικά αν ήδη λαμβάνει προσοχή (receiving attention) (άρα παραλείπεται το στάδιο του φλερτ), οπότε τότε μπορεί να αρραβωνιαστεί και στην επόμενη επανάληψη να παντρευτεί υποθέτοντας βέβαια ότι τίποτα δεν έχει γίνει που να έχει απομακρύνει τη νέα αρραβωνιαστικιά από την εκκλησία μεταξύ των προγραμματιστικών κύκλων. Από την άλλη πλευρά ένας τον οποίο δεν προσέχουν ποτέ, μπορεί να φλερτάρει συνεχώς ή, αν ξαφνικά αντιληφθεί ότι είναι αρραβωνιασμένος engaged (ίσως από προξενιό), να παραλείψει το στάδιο go to fiancé. Αν βρίσκεται στην εκκλησία και η αρραβωνιαστικιά είναι παρούσα τότε παντρεύεται (ικανοποιεί το στόχο του).

Προφανώς προγραμματίζοντας αυτό το BRP σε FSM θα χρειαζόταν τουλάχιστον 25 γραμμές, υποθέτοντας ότι ο στόχος χρειάζεται μια γραμμή για να περιγραφτεί και κάθε πιθανή μετάβαση μεταξύ καταστάσεων μπορεί να περιγραφτεί με μια γραμμή. Ομοίως, ένας μεγάλος αριθμός πιθανών περιβαλλοντικών καταστάσεων έχουν αγνοηθεί σαν μη σχετικές με την επιδίωξη του στόχου του πράκτορα.

Για όλη αυτή την «κομψότητά του», το BRP είναι απλό να προγραμματιστεί σε αρκετές γλώσσες προγραμματισμού. Ουσιαστικά το BRP προέρχεται από ένα συμβατικό, διαδοχικό σχέδιο, για παράδειγμα :

flirt → become engaged → go to church → get married

Το BRP απλά αντιστρέφει τη σειρά και μετά ορίζει ένα μηχανισμό που αναγνωρίζει το πότε μπορεί να ενεργοποιηθεί κάθε στοιχείο.

Πιο αναλυτικά, στο BRP κάθε βήμα (ουσιαστικά κάθε στόχος) είναι μια πλειάδα (π, ρ, α) όπου π είναι η προτεραιότητα, ρ είναι ο ελευθερωτής (releaser) και α είναι η ενέργεια. Ένα BRP είναι ένα σύνολο τέτοιων βημάτων συναφών με μια συνθήκη στόχου. Το ρ είναι αυτό που καθορίζει το πότε μπορεί να εκτελεστεί το βήμα. Το α μπορεί να είναι μια πρωταρχική ενέργεια, ένα άλλο BRP ή μια απλή ακολουθία (που περιγράφεται αναλυτικά παρακάτω).

Η σειρά που θα εκτελεστούν τα βήματα εξαρτάται από το π και από το ρ. Το ρ σημαίνει ότι μπορούν να εκτελεστούν τα βήματα και το π ποιο μπορεί να εκτελεστεί πρώτο. Αν δύο βήματα, που μπορούν να εκτελεστούν, έχουν το ίδιο π, τότε επιλέγεται ένα από αυτά τυχαία. Αν κανένα βήμα δεν μπορεί να εκτελεστεί τότε το BRP τερματίζεται. Συνήθως το πρώτο βήμα στο BRP είναι η συνθήκη στόχου. Το ρ1 σε αυτή την περίπτωση αναγνωρίζει ότι το BRP είναι επιτυχές και το α1 τερματίζει το BRP. Ένα άλλο παράδειγμα που θα μας βοηθήσει στην κατανόηση του BRP είναι το παρακάτω.

Έστω ότι έχουμε έναν κόσμο που αποτελείται από σωρούς χρωματιστών τούβλων και ο πράκτορας θέλουμε να κρατήσει το μπλε τούβλο. Τότε έχουμε το εξής :

Priority	Releaser ⇒ Action
4	(holding block) (block blue) ⇒ *goal*
3	(holding block) ⇒ drop-held, lose-fixation
2	(fixated-on blue) ⇒ grasp-top-of-stack
1	(blue-in-scene) ⇒ fixate-blue

Σε αυτήν την περίπτωση η προτεραιότητα που φαίνεται στα αριστερά είναι αυστηρώς αριθμημένη για κάθε θέση, με υψηλότερη αυτή στην κορυφή.

Υποθέτουμε την περίπτωση όπου ο κόσμος αποτελείτο από ένα σωρό από ένα κόκκινο τούβλο πάνω σε ένα μπλε. Εάν ο πράκτορας δεν έχει βρει ακόμα το μπλε τούβλο πριν ενεργοποιηθεί το πλάνο αυτό (και δεν κρατά τίποτα) , τότε η πρώτη κίνηση θα είναι αυτή με τον αριθμό 1 γιατί είναι η μοναδική της οποίας ο απελευθερωτής της ικανοποιείται. Εάν σαν μέρος του προηγούμενου πλάνου, ο πράκτορας έχει εστιάσει στο μπλε τούβλο, η κίνηση 1 θα αποφευχθεί γιατί στην κίνηση 2, που έχει υψηλότερη προτεραιότητα, ικανοποιείται ο απελευθερωτής της. Μόλις γίνει η εστίαση (fixation), το στοιχείο 2 θα πυροδοτηθεί. Εάν το πιάσιμο του τούβλου είναι επιτυχής τότε θα ακολουθήσει το στοιχείο 3 αλλιώς θα επαναληφθεί το 2. Αν υποθέσουμε ότι το κόκκινο τούβλο το έχει πιάσει και το έχει απορρίψει, η επόμενη επιτυχής κίνηση του στοιχείου 2 θα έχει ως αποτέλεσμα να κρατάει ο πράκτορας το μπλε μπλοκ, που σ' αυτό το σημείο το στοιχείο 4 θα καταλάβει ότι ο στόχος έχει εκπληρωθεί και θα τερματίσει το πλάνο. Αυτό το απλό αντιδραστικό πλάνο μπορεί να παράγει ένα μεγάλο αριθμό από ακολουθίες. Στο πλαίσιο του κόκκινου τούβλου πάνω στο μπλε, μπορούμε να περιμένουμε να εκτελεστεί το σχέδιο 1–2–3–1–2–4. Αλλά αν ο πράκτορας έχει ήδη εστιάσει στο μπλε τούβλο και αποτύχει να πιάσει το κόκκινο με την πρώτη προσπάθεια, τότε το πλάνο θα είναι ως εξής: 2–1–2–3–1–2–4. Το αντιδραστικό πλάνο είναι το ίδιο εύρωστο και καιροσκοπικό σε αλλαγές που μπορούν να προκληθούν και από έναν άλλο πράκτορα.

Το βασικό στοιχείο του BRP είναι ότι είναι εύκολο να σχεδιαστεί. Για τη δημιουργία του, ο σχεδιαστής φαντάζεται το χειρότερο σενάριο για την επίτευξη του στόχου αγνοώντας τα περιττά βήματα. Οι προτεραιότητες για κάθε βήμα καθορίζονται με την αντίστροφη σειρά από αυτή που εκτελέστηκαν τα βήματα. Στη συνέχεια τοποθετούνται τα προαπαιτούμενα, ξεκινώντας από το υψηλότερης προτεραιότητας βήμα, τα οποία καθορίζουν πότε μπορεί να πυροδοτηθεί. Στο παραπάνω παράδειγμα τα προαπαιτούμενα είναι το blue-in-scene, fixated-on-blue κλπ..

Εάν μια ενέργεια συνέχεια αποτυγχάνει (για παράδειγμα η grasp-top-of-stack) τότε το BRP θα οδηγηθεί σε ατέρμονο βρόγχο. Τη λύση εδώ παρέχουν τα competences που

χρησιμοποιεί το POSH, όπως θα δούμε παρακάτω. Επίσης τι γίνεται όταν θέλουμε ο στόχος του πράκτορα να μην επιτυγχάνεται ποτέ; Για παράδειγμα ο στόχος ενός πράκτορας να είναι η επιβίωσή του στον κόσμο. Τη λύση εδώ δίνει πάλι το Posh με τη χρήση των Drive Collection που θα δούμε επίσης στο επόμενο κεφάλαιο.

3.1.4. POSH Reactive Plans

Μερικοί πράκτορες έχουν παραπάνω από δύο στόχους. Περαιτέρω, πολλές «ενέργειες» μπορεί να απαιτούν κάποιες πολλαπλές υπό-ενέργειες για να πραγματοποιηθούν, και αυτές με τη σειρά τους μπορεί να απαιτούν ένα BRP, για να τις οργανώσει.

Σύμφωνα με την Bryson (Bryson 2000a) υπάρχουν τρία είδη προβλημάτων που πρέπει να αντιμετωπιστούν :

- Μερικά πράγματα πρέπει να ελέγχονται διαρκώς. Για παράδειγμα: ένας δυνατός θόρυβος θα κάνει κάποιον να σταματήσει οτιδήποτε και αν κάνει και να γυρίσει το βλέμμα του προς την κατεύθυνση που άκουσε το θόρυβο, χωρίς οποιαδήποτε συνειδητή επεξεργασία της αιτίας που προκάλεσε την αλλαγή της προσοχής του.
- Μερικά πράγματα χρειάζεται να ελέγχονται πολύ σπάνια. Για παράδειγμα: όταν κάποιος περπατάει, έχει ένα αξιόπιστο πλάνο για τον έλεγχο των ποδιών του το οποίο είναι με τη σειρά του χαρακτηριστικό του επιμέρους βηματισμού του. Θα ήταν πολύ ασυνήθιστο να προσέχει ή να προσπαθεί να ελέγξει τους μύες που εμπλέκονται.
- Μερικά πράγματα πρέπει να ελέγχονται μόνο σε συγκεκριμένες περιπτώσεις, αλλά σε απρόβλεπτη σειρά. Για παράδειγμα, εάν κάποιος λύνει ένα πάζλ, μπορεί να έχει ένα σύνολο από κανόνες σχετικά με το πού βρίσκονται τα ακρινά κομμάτια ή κομμάτια με μια συγκεκριμένη απόχρωση του μπλε πάνω τους. Αντίθετα με την κατάσταση περπατήματος δεν μπορεί κανείς να προβλέψει πως θα είναι το επόμενο κομμάτι που θα τραβήξει την προσοχή του και έτσι δεν μπορεί να σχεδιάσει μια σειρά εκ των προτέρων. Αυτή είναι η κατάσταση που χρειάζεται ένα BRP.

Μπορεί να φαίνεται απλό τα πάντα να αναπαρίστανται σαν ένα ζεύγος αίσθησης/δράσης, δηλαδή ένα τεράστιο BRP. Όμως δεν υφίσταται (είναι υπολογιστικά δύσχρηστο) να έχεις κάθε πιθανή ικανότητα που να απαιτεί αυτό το επίπεδο προσοχής και να είναι και συνεχώς προσπελάσιμη.

Η Bryson ανάπτυξε ένα μηχανισμό επιλογής δράσης που υποστηρίζει και τις τρεις παραπάνω καταστάσεις με τρεις τύπους αναπαράστασης και μια μεθοδολογία ανάπτυξης που καθορίζει πότε να εφαρμοστεί το καθένα (Bryson and Stein, 2001b; Bryson, 2001). Η μεθοδολογία ανάπτυξης ονομάζεται BOD (Behaviour Oriented Design) και είναι μια επαναληπτική διαδικασία που καθορίζει ποιος μηχανισμός επιλογής δράσης θα χρησιμοποιηθεί αλλά και την διακριτότητα των αρχικών ενεργειών. Οι αρχικές ενέργειες κωδικοποιούνται σαν ενότητες συμπεριφοράς που αντιστοιχούν σε αντικείμενα. Περισσότερες πληροφορίες για το BOD υπάρχουν παρακάτω στο κεφάλαιο 3.4.

Ο αντιπροσωπευτικός σκελετός για την επιλογή δράσης του BOD ονομάζεται Parallel rooted, Ordered, Slip-stack Hierarchical (POSH) reactive plans. Το Posh αναπτύχθηκε από την Joanna Bryson στα πλαίσια της διδακτορικής της διατριβής το 2001, η οποία αφορούσε τη διαμόρφωση και το συντονισμό πολύπλοκων και προσαρμόσιμων ευφυών πρακτόρων.

Τα POSH πλάνα περιέχουν πέντε είδη στοιχείων. Πρώτα είναι οι **πρωταρχικές ενέργειες (actions) και αισθήσεις (senses).**

Οι πρωταρχικές ενέργειες είναι οι βασικές ενέργειες που μπορεί να κάνει ο πράκτορας στο περιβάλλον για παράδειγμα το να περπατά, να τρέχει, να πηδάει, να τρώει κλπ. Αυτές οι ενέργειες είναι ήδη καθορισμένες από την πλατφόρμα λογισμικού που θα χρησιμοποιήσουμε και δεν θα χρειαστεί, τουλάχιστον για τη συγκεκριμένη εργασία να τις πειράξουμε. Το ίδιο αφορά και τις αισθήσεις. Αισθήσεις είναι το τι βλέπει ο πράκτορας γύρω του, ποια είναι η εσωτερική του

κατάσταση δηλαδή αν πεινάει, αν διψάει, αν είναι κουρασμένος, κλπ. Οι διαφορές μεταξύ των πρωταρχικών ενεργειών και των αισθήσεων είναι οι εξής:

- Επιστροφή τιμών: οι πρωταρχικές ενέργειες δεν επιστρέφουν καμία ουσιαστική τιμή, εκτός της περίπτωσης ακραίας αποτυχίας, ενώ οι αισθήσεις (senses) επιστρέφουν ουσιαστικές τιμές που μπορούν να χρησιμοποιηθούν ως προϋποθέσεις για την εκτέλεση ενεργειών
- Αλλαγή στο περιβάλλον: Οι αισθήσεις δεν μπορούν να φέρουν καμία αλλαγή στον κόσμο ενώ οι πρωταρχικές ενέργειες μπορούν να επιφέρουν αλλαγές αλλάζοντας πράγματα στον κόσμο. Για παράδειγμα: Η ενέργεια «τρώω ένα μήλο» επιφέρει αλλαγή στον κόσμο, εφόσον το μήλο πρέπει να εξαφανιστεί από αυτόν εφόσον, φαγωθεί. Η αίσθηση όμως «βλέπω ένα μήλο» δεν φέρει καμία αλλαγή.

Τα άλλα τρία είδη στοιχείων στα σχέδια POSH είναι τα *action patterns τα competences* και τα *drive collections.*

Action Patterns

Το Posh χρησιμοποιεί αυτό το στοιχείο για να περιγράφει απλές ακολουθίες κινήσεων. Τα action patterns είναι απλές ακολουθίες (simple sequences) ενεργειών και χρησιμοποιούνται από την BOD για να καλύψουν την $2^{η}$ κατάσταση που αναφέρθηκε προηγουμένως «Μερικά πράγματα χρειάζεται να ελέγχονται πολύ σπάνια».

Μια δομή θεμελιώδους σημασίας είναι οι ακολουθίες από πρωταρχικές ενέργειες (i_1, i_2, …, i_n). Αυτό το είδος στοιχείου είναι σημαντικό για δύο πράγματα:

- Επιτρέπει στο σχεδιαστή του πράκτορα να διατηρεί το σύστημα απλό και
- Επιτρέπει γρήγορή βελτιστοποίηση των στοιχείων που τρέχουν σε σειρά.

Εκτέλεση της ακολουθίας σημαίνει: Έναυσμα της ακολουθίας μετά από την απελευθέρωση της πρώτης ενέργειας i_1 για εκτέλεση. Η ολοκλήρωση μιας ενέργειας (i_i) απελευθερώνει την επόμενη (i_{i+1}) μέχρι να μην υπάρχουν άλλα ενεργά στοιχεία- ενέργειες.

Η ακολουθία είναι ένα μέρος του ελέγχου, τα στοιχεία μπορεί να εκτελεστούν και με άλλες σειρές σε άλλες ακολουθίες.

Το γεγονός ότι τα στοιχεία της ακολουθίας απελευθερώνονται, όταν τελειώσει το προηγούμενό τους στοιχείο είναι χρήσιμο σε περιβάλλοντα πραγματικού χρόνου. Και το γεγονός ότι καταστέλλονται ενεργά από την ύπαρξη του προηγούμενού τους στοιχείου αυξάνει την ευρωστία του σχεδίου-πλάνου.

Γενικότερα οι ακολουθίες βοηθούν στο να υπάρχει γρήγορος και απλός έλεγχος των καταστάσεων, όπου οι κινήσεις ακολουθούν η μία την άλλη. Όμως ο έλεγχος αυτός συχνά είναι σύνθετος αφού εμπλέκονται οι ικανότητες του περιβάλλοντος αλλά και του πράκτορα. Αρκετά είδη γεγονότων μπορεί να εισβάλλουν κατά την τελειοποίηση μιας μελλοντικής ακολουθίας κινήσεων. Τα είδη αυτά είναι δύο κατηγοριών :

- Μερικές δυσκολίες ή ευκαιρίες μπορεί να αναγκάσουν την τωρινή σειρά των κινήσεων να αλλάξει (reordering).
- Μερικά στοιχεία (ενέργειες) μπορεί να χρειαστούν να επαναληφθούν και άλλα να απορριφθούν.
- Μερικά γεγονότα, όπως ένας κίνδυνος ή απλά ένα αίτημα, μπορεί να απαιτήσουν τη χρησιμοποίηση μιας τελείως διαφορετικής ακολουθίας με άλλες ενέργειες από το να ολοκληρωθεί η τωρινή ακολουθία.

Γι’ αυτό το Posh χρησιμοποιεί τα άλλα δύο είδη στοιχείων , τα Competences και τα Drive collections.

Competences

Τα Competences είναι μια μορφή BRP (το οποίο αναφέρθηκε αναλυτικά στο υποκεφάλαιο 3.1.3) και χρησιμοποιούνται στο BOD για να καλύψουν την 3η κατάσταση που αναφέρθηκε προηγουμένως «Μερικά πράγματα πρέπει να ελέγχονται μόνο σε συγκεκριμένες περιπτώσεις». Όπως και το action pattern, έτσι και τα competences εστιάζουν την προσοχή τους σε ένα συγκεκριμένο σύνολο από στοιχεία κατάλληλα για να εκτελέσουν μια συγκεκριμένη ενέργεια. Ένα Competence είναι χρήσιμο όταν τα στοιχεία δεν μπορούν να μπουν σε μια σειρά εκ των προτέρων.

Η διαφορά του με ένα κλασικό BRP είναι ότι το competence επιτρέπει την προδιαγραφή ενός ορίου για τον αριθμό των επαναλήψεων κάθε βήματος. Δηλαδή το όριο αυτό τοποθετείται σε κάθε βήμα. Έτσι το βήμα σε ένα competence είναι μια τετράδα (π, ρ, α, η) όπου η είναι ο αριθμός των επαναλήψεων. Το αρνητικό η δηλώνει απεριόριστες επαναλήψεις.

Επίσης, τα competences επιστρέφουν τιμή:

Τ Εάν τερματίσουν λόγω της ενεργοποίησης του στόχου και

⊥ Εάν τερματίσουν γιατί κανένα βήμα δε μπορεί να ενεργοποιηθεί.

Drive Collections

Τα drive collections είναι και αυτά ένα είδος BRP και χρησιμοποιούνται στο BOD για να καλύψουν την 1η κατάσταση που αναφέρθηκε προηγουμένως «Μερικά πράγματα πρέπει να ελέγχονται διαρκώς». Είναι ένα από τα πιο χρήσιμα και βασικά στοιχεία του Posh και αποτελεί τη ρίζα του πλάνου. Ένα drive collection αποτελείται από drive elements ή αλλιώς βήματα που αντιστοιχούν σε κάθε βασικό στόχο του πράκτορα. Κάθε βήμα αποτελείται από πέντε στοιχεία (π, ρ, α , Α, ν). Το π (προτεραιότητα) και το ρ (ελευθερωτής) είναι τα ίδια με πριν. Το Α είναι η ρίζα της ιεραρχίας του βήματος που είναι και αυτό ένα BRP. Το α είναι το τωρινό ενεργό στοιχείο του βήματος. Όταν ένα βήμα ενεργοποιείται, το α πυροδοτείται, όπως σε ένα BRP. Όμως αν το α είναι ένα competence και ενεργοποιήσει ένα παιδί, το β, που είναι επίσης ένα POSH στοιχείο (competence ή action pattern),τότε το α γι' αυτό το drive collection, λαμβάνει την τιμή του β. Από την άλλη πλευρά αν το α είναι ένα competence ή ένα action pattern και τερματίσει, ή αν αυτή είναι η πρώτη φορά που το βήμα πυροδοτείται τότε το α αντικαθιστάται με Α, τη ρίζα της ιεραρχίας. Παρατηρούμε δηλαδή ότι κάθε drive element μπορεί να είναι ένα competence ή ένα action pattern ή ακόμα και μια πρωταρχική ενέργεια.

Η πολιτική που ακολουθείται είναι να υπάρχει ένα μόνο ενεργό στοιχείο του POSH σε κάθε βήμα του Drive collection και αποτελεί ένα από τα χαρακτηριστικά της σχεδίασης του POSH – **the slip-stack hierarchy** . Για κάθε κύκλο του action selection μόνο το drive collection και το πολύ άλλη μια ένωση Posh στοιχείων θα έχουν ελεγμένες τις προτεραιότητές τους. Η ύπαρξη προτεραιοτήτων στους στόχους (drives), αλλά και στα παιδιά τους όταν αυτά είναι competences δικαιολογούν το χαρακτηριστικό **Ordered** του POSH.

Το πέμπτο μέρος ενός βήματος, ν, είναι μια προαιρετική μέγιστη τιμή συχνότητας επίσκεψης αυτού του βήματος. Αυτή είναι μια ευκολία που παρέχει το drive collection, όπως το όριο επανάληψης (η) στα competences. Η συχνότητα σε ένα σύστημα πραγματικού χρόνου δηλώνει ένα χρονικό όριο για το πόσο συχνά εκτελείται ένα βήμα (drive element). Για παράδειγμα: Ένα κινούμενο ρομπότ μπορεί να έχει σαν βήμα υψηλότερης προτεραιότητας να ελέγχει το επίπεδο της μπαταρίας του και να εκτελείται αυτό κάθε δύο λεπτά. Μια άλλη χαμηλότερη προτεραιότητα είναι να περιηγείται στο χώρο. Έτσι καθώς το ρομπότ θα περιηγείται στο χώρο κάθε δύο λεπτά, η προτεραιότητά του θα αλλάζει στο να ελέγχει τις μπαταρίες του.

Ένα άλλο χαρακτηριστικό που διακρίνει τα drive collections από τα competences και το BRP είναι το εξής:

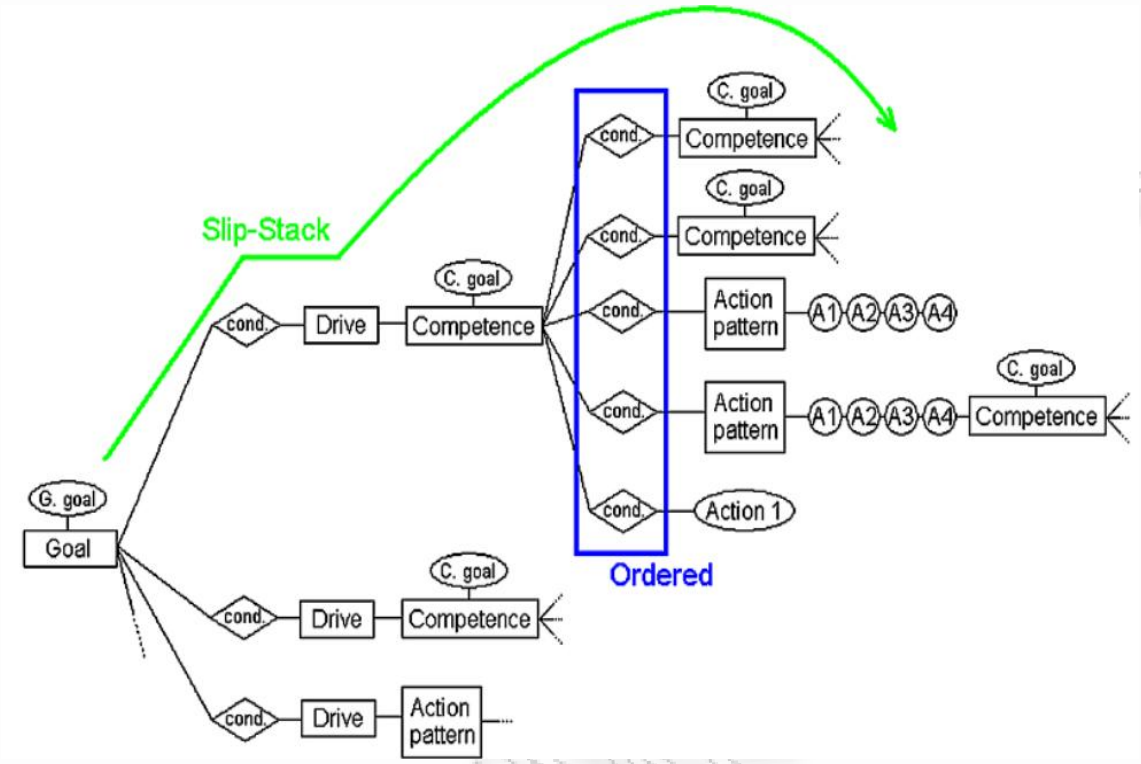
Στα competences μόνο ένα στοιχείο μπορεί να εκτελείται κάθε φορά, αλλά σε ένα drive collection πολλαπλά drives μπορούν να δρουν αποτελεσματικά ταυτόχρονα. Αυτό σημαίνει ότι εάν ένα υψηλής προτεραιότητας drive τραβήξει την προσοχή του μηχανισμού επιλογής ενέργειας (action selection), η κατάσταση κάθε ενεργού drive χαμηλότερης προτεραιότητας διατηρείται. Για παράδειγμα: Στην περίπτωση του ρομπότ εάν το drive της περιήγησης βρίσκεται στη διαδικασία επιλογής κατεύθυνσης, αλλά εκείνη τη στιγμή πρέπει να ελέγξει τη μπαταρία του, τότε ελέγχει τη μπαταρία του και μόλις αυτό ολοκληρωθεί επιτυχώς, η διαδικασία επιστρέφει στην επιλογή κατεύθυνσης από εκεί που είχε σταματήσει. Στα δύο αυτά χαρακτηριστικά του drive collection οφείλεται και ο ψευδοπαραλληλισμός που χαρακτηρίζει το POSH (**Parallelrooted**, Ordered, Slip-stack Hierarchical).

Για όλα τα παραπάνω χρησιμοποιούμε ένα drive collection σαν τη βάση του πλάνου. Η λογική είναι ότι κάθε drive element αποτελεί ένα βασικό στόχο του πράκτορα που για να ικανοποιηθεί μπορεί να χρησιμοποιηθεί είτε ένα competence ή ένα action pattern ή ακόμα και μια πρωταρχική ενέργεια. Δηλαδή υπάρχει ένα βασικό πλάνο, το drive collection το οποίο περιέχει στοιχεία που είναι από μόνα τους πλάνα δηλαδή τα competences ή τα action patterns. Αυτό το πλάνο ονομάζεται ιεραρχικό. Η ερώτηση που δημιουργείται είναι πότε ξεκινά αυτό το πλάνο και πότε τελειώνει; Γενικά όταν το πλάνο τελειώσει, τότε τελειώνει και η νοημοσύνη του πράκτορα. Αυτό είναι χρήσιμο για κάποια συγκεκριμένα είδη πρακτόρων που καλούνται μόνο για μια συγκεκριμένη δουλειά. Όμως τι συμβαίνει όταν ο πράκτορας θέλουμε να διαρκεί για κάποιο χρονικό διάστημα και να αποφασίζει το τι θα κάνει με βάση ένα σύνολο από κίνητρα και αρχές; Ονομάζουμε αυτό το είδος του πράκτορα ολοκληρωμένο πράκτορα. Τη λύση αυτή μας δίνει το Drive Collection και αυτός είναι και ένας βασικός λόγος που χρησιμοποιείται σαν τη βάση της ιεραρχίας του πλάνου. Το Drive Collection είναι σχεδιασμένο να μην τερματίζει ποτέ. Αυτός ο ειδικός σχεδιασμός απλά μπλοκάρει τους δύο λόγους που τερματίζουν ένα BRP. Πρώτον, το Drive Collection δεν έχει στόχο γι' αυτό ποτέ δεν επιτυγχάνει ή δεν ολοκληρώνεται. Δεύτερον, έχει ένα στοιχείο που πάντα μπορεί να τρέχει και ποτέ δεν αποτυγχάνει.

Γενικότερα τα drive collections χρησιμεύουν ως βάση της ιεραρχίας του POSH πλάνου. Τα χαρακτηριστικά του είναι τα εξής:

- Υπάρχει μόνο ένα Drive collection και ελέγχεται σε κάθε επανάληψη του μηχανισμού action selection.
- Κάθε βήμα του drive collection, δηλαδή κάθε drive, αναπαριστά έναν ξεχωριστό στόχο του πράκτορα. Αυτοί οι στόχοι μπορούν να αντιμετωπιστούν παράλληλα (ψευδοπαράλληλα καλύτερα), έτσι κάθε βήμα του drive collection παρακολουθεί όχι μόνο το πιο είναι το παιδί του αλλά και το τι έκανε πρόσφατα – το τωρινό πλαίσιο του action selection.
- Για τη διευκόλυνση του ψευδοπαραλληλισμού τα drive elements έχουν τις συχνότητες. Έτσι κάποιες ενέργειες (για παράδειγμα η αναπνοή, το κοιτάζω γύρω μου) μπορεί να έχουν υψηλή προτεραιότητα κάθε λίγα δευτερόλεπτα, αλλά μετά την αρχικοποίηση της ενέργειας ο μηχανισμός επιλογής δράσης (action selection) είναι ελεύθερος στο να σκεφτεί άλλους στόχους χαμηλότερης προτεραιότητας μέσα στο επόμενο διάστημα χρόνου.

Τελικά ένα ολοκληρωμένο POSH πλάνο έχει την εξής δομή :



Εικόνα 5: Δομή πλάνου POSH και πορεία που ακολουθείται κατά την εκτέλεσή του.

3.2.Ένα απλό παράδειγμα των στοιχείων του POSH.

Παρακάτω ακολουθεί ένα παράδειγμα για τη διαδικασία δημιουργίας ενός πλάνου που αφορά τη συμπεριφορά μιας μαϊμούς, με τη χρήση των στοιχείων του POSH.

Έστω ότι έχουμε μια μαϊμού πράκτορα που θέλει να φάει μια μπανάνα. Τότε το πιο απλό πλάνο είναι το εξής :

$$\langle \text{get a banana} \rightarrow \text{peel a banana} \rightarrow \text{eat a banana} \rangle$$ *(πλάνο 1)*

Αυτό είναι ένα μια απλή ακολουθία ή ένα action pattern. Τα get a banana, peel a banana και eat a banana είναι οι βασικές ή αλλιώς, όπως έχουμε αναφέρει, πρωταρχικές ενέργειες που διαθέτει η μαϊμού. Οι ενέργειες αυτές θα πρέπει να πραγματοποιηθούν με τη συγκεκριμένη σειρά.

Φυσικά το να σχεδιάσουμε ολόκληρη τη συμπεριφορά της μαϊμούς καθ' όλη τη διάρκεια της ζωής της με μια ακολουθία είναι βαρετό και βέβαια αδύνατο. Χρήσιμο θα ήταν να γνωρίζει η μαϊμού και το πότε μπορεί να κάνει μια ενέργεια. Ένας τρόπος για να καθορίσουμε το **πότε** (θα κάνει κάτι) είναι να το συνδυάσουμε με ένα συγκεκριμένο πλαίσιο, όπου ο πράκτορας θα μπορεί να αντιληφθεί το **τί** θα κάνει. Αυτός ο συνδυασμός είναι ένας κανόνας παραγωγής (*production rule*). Το πλαίσιο ονομάζεται προϋπόθεση (*precondition*) και το **τι** ονομάζεται πράξη. Έτσι το προηγούμενο πλάνο μετατρέπεται σε :

$$\begin{aligned} (\text{have hunger}) &\Rightarrow \text{get a banana} \\ (\text{have a banana}) &\Rightarrow \text{peel a banana} \\ (\text{have a peeled banana}) &\Rightarrow \text{eat a banana} \end{aligned}$$ *(πλάνο 2)*

Μέσα στην παρένθεση είναι οι προϋποθέσεις. Ας υποθέσουμε ότι είναι ερωτήσεις. Αν η απάντηση που δίνουν είναι θετική τότε εκτελείται η πράξη. Στις ερωτήσεις αυτές δίνουν απαντήσεις για το αν ικανοποιούνται ή όχι οι αισθήσεις του πράκτορα. Αυτό το πλάνο είναι καλύτερο από το προηγούμενο, αφού δείχνει και το πότε μπορεί να εκτελεστεί η ενέργεια. Αν όμως θέλαμε η μαϊμού μας να μπορεί να κάνει και κάτι άλλο, όπως το να δώσει την μπανάνα της σε κάποιον άλλο δηλαδή :

$$\langle \text{get a banana from left} \rightarrow \text{pass a banana to right} \rangle$$

(πλάνο 3)

Που με τις προϋποθέσεις γίνεται :

$$\begin{aligned} (\text{left neighbor offers banana}) &\Rightarrow \text{get a banana from left} \\ (\text{have a banana}) &\Rightarrow \text{pass a banana to right} \end{aligned}$$

(πλάνο 4)

Τώρα έχουμε δύο κανόνες με το ίδιο πλαίσιο «have a banana». Η μαϊμού τι θα κάνει σε αυτή την περίπτωση; Θα την ξεφλουδίσει (peel a banana) ή θα τη δώσει (pass a banana to right);

Θα μπορούσαμε να βοηθήσουμε τη μαϊμού, βάζοντας και άλλες προϋποθέσεις. Για παράδειγμα: Όλοι οι κανόνες στο πλάνο 2 να έχουν την προϋπόθεση «have hunger» και όλοι οι κανόνες του πλάνου 4 να έχουν την προϋπόθεση «at a party». Όμως τι θα γινόταν αν η μαϊμού ήταν σε πάρτι και πεινούσε; Γι' αυτό το λόγο βάζουμε τις προτεραιότητες. Εδώ θα επιλέξουμε αν θέλουμε η μαϊμού μας να είναι ευγενική ή όχι.

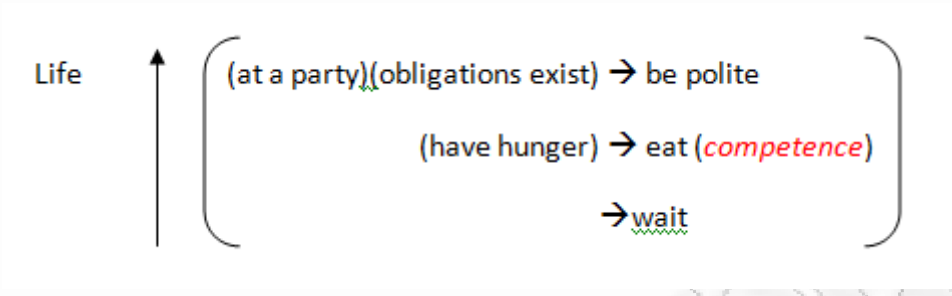
Γι' αυτό το λόγο χρησιμοποιούμε το competence. Έτσι το πλάνο 1 και 2 θα είναι ως εξής :

$$(\text{have hunger}) \Rightarrow \uparrow \left\langle \begin{aligned} (\text{full}) &\Rightarrow \textit{goal} \\ (\text{have a peeled banana}) &\Rightarrow \text{eat a banana} \\ (\text{have a banana}) &\Rightarrow \text{peel a banana} \\ &\Rightarrow \text{get a banana} \end{aligned} \right\rangle$$

(πλάνο 5)

Εδώ παρατηρούμε ότι έχουν ομαδοποιηθεί οι κανόνες που σχετίζονται με μια προϋπόθεση δηλαδή τη have hunger. Επίσης το βέλος δείχνει την προτεραιότητα του κάθε κανόνα με το πρώτο να έχει την υψηλότερη. Έτσι αν η μαϊμού έχει μια ξεφλουδισμένη μπανάνα θα την φάει. Αν έχει μια ολόκληρη θα την ξεφλουδίσει, αλλιώς θα προσπαθήσει να πάρει μια μπανάνα. Επίσης παρατηρούμε ότι υπάρχει και ένας κανόνας με την υψηλότερη προτεραιότητα που αντιστοιχεί στο στόχο ο οποίος είναι να χορτάσει η μαϊμού. Έτσι αν η μαϊμού φάει μια μπανάνα και δεν χορτάσει μπορεί να φάει και άλλη μια! Ο στόχος ανιχνεύει πότε τελείωσε το έργο του competence. Το competence τερματίζει, όταν έχει επιτύχει το στόχο του ή όταν κανένας από τους κανόνες του δεν μπορεί να πραγματοποιηθεί.

Όπως αναφέραμε πιο πάνω, η μαϊμού εκτός από το στόχο της να ικανοποιήσει την πείνα της έχει και σαν άλλο στόχο να είναι ευγενική με το να δώσει τη μπανάνα της σε κάποιον άλλο. Άρα εδώ το πλάνο μας γίνεται πιο σύνθετο έχοντας δύο βασικούς στόχους. Επίσης θέλουμε η μαϊμού να έχει και διάρκεια ζωής. Σε αυτά τη λύση μας δίνει το Drive Collection. Έτσι έχουμε το εξής Drive Collection:



(πλάνο 6)

Παρατηρούμε ότι προστέθηκε και μια αίσθηση για αντίληψη των υποχρεώσεων. Έτσι η μαϊμού μπορεί να τρώει ακόμα και όταν είναι σε πάρτι, όσο δεν την ενδιαφέρουν οι κοινωνικές υποχρεώσεις. Εδώ δεν έχει προσδιοριστεί στόχος και υπάρχει μια χαμηλής προτεραιότητας συμπεριφορά που μπορεί πάντα να εκτελεστεί (wait) έτσι ώστε η ζωή (life) να μην τερματίζεται ποτέ. Παρατηρούμε επίσης ότι το 2ο drive element ενεργοποιεί το competence eat που είναι αυτό που περιγράψαμε πιο πάνω.

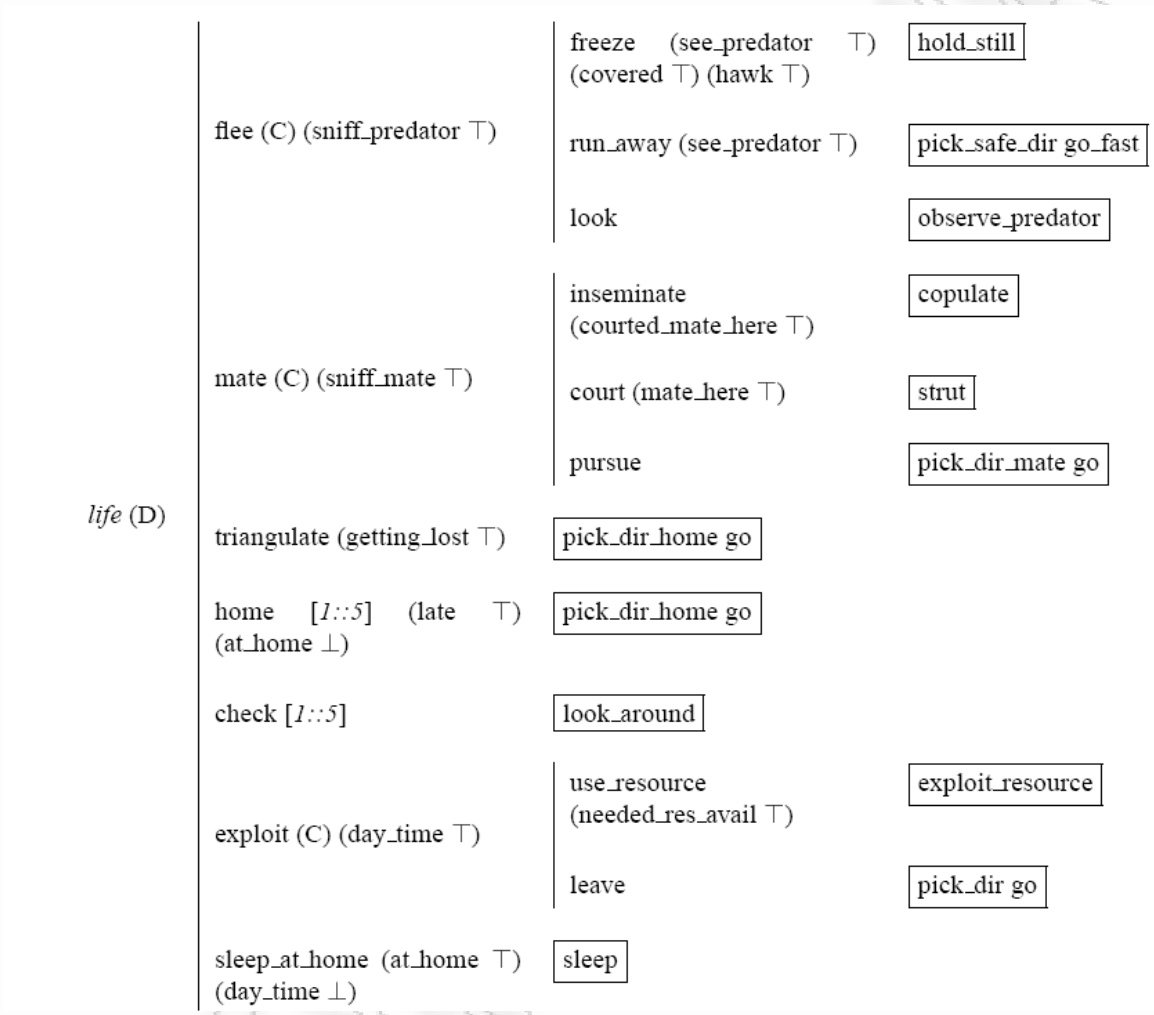
3.3.Ένα παράδειγμα μιας πλήρους ιεραρχίας POSH και επεξήγηση της ακολουθίας των βημάτων του πλάνου.

Παρακάτω ακολουθεί ένα παράδειγμα που αναφέρεται στη διαχείριση πολλών αντικρουόμενων στόχων. Ο Tyrrell [1993] δημιούργησε ένα περιβάλλον προσομοίωσης. Μέσα σε αυτό ένα τρωκτικό προσπαθεί να επιβιώσει σε μια σαβάνα, γεμάτη από κινδύνους, παθητικούς και ενεργητικούς, ψάχνοντας να βρει τροφή, καταφύγιο και ευκαιρίες αναπαραγωγής. Το τρωκτικό έχει περιορισμένες δυνατότητες αίσθησης και δεν είναι ποτέ σίγουρο για οτιδήποτε παρά μόνο για το πολύ κοντινό του περιβάλλον. Μπορεί να βλέπει καλύτερα κατά τη διάρκεια της μέρας παίρνοντας την όρθια θέση, στηριζόμενο μόνο στα πίσω του πόδια, κάτι που το κάνει πιο ορατό στους εχθρούς του. Οι στόχοι του πράκτορα σε αυτό το περιβάλλον προσομοίωσης σύμφωνα με τον Tyrell είναι οι εξής:

- Εύρεση τροφής. Μαζί με το νερό υπάρχουν και άλλα τρία είδη τροφής.
- Αποφυγή των αρπακτικών. Υπάρχουν αιλουροειδή και πτηνά αρπακτικά με διαφορετική αντίληψη και ικανότητες.
- Αποφυγή κινδύνων. Οι παθητικοί κίνδυνοι στο περιβάλλον περιλαμβάνουν τη δηλητηριώδη τροφή ,τους βράχους, την υψηλή θερμοκρασία και το σκοτάδι. Όμως το περιβάλλον παρέχει διάφορα είδη καταφυγίου, όπως τα δέντρα, το γρασίδι και τις φωλιές .
- Φροντίδα. Η φροντίδα είναι απαραίτητη για τον έλεγχο της θερμοκρασίας του τρωκτικού και γενικότερα για την υγεία του.
- Ύπνος στο σπίτι. Το τρωκτικό είναι τυφλό τη νύχτα, η φωλιά του παρέχει καταφύγιο από τα αρπακτικά και το βοηθά να διατηρεί τη θερμοκρασία του καθώς διατηρεί και την ενέργειά του.
- Αναπαραγωγή. Το τρωκτικό είναι γένους αρσενικού, έτσι ο στόχος του είναι να βρει και να αναπαραχθεί με το ταίρι του.

Αυτά τα προβλήματα διαφέρουν από πολλές πλευρές: εξάρτηση από τα εσωτερικά ενάντια στα εξωτερικά ερεθίσματα, περιοδική, συνεχόμενη ή περιστασιακή έκφραση , βαθμός του πόσο επείγον είναι κάτι κ.α. Εκτός από αυτά τα προβλήματα, το περιβάλλον είναι δυναμικό. Οι ποσότητες του νερού και του φαγητού καθώς και η θερμοκρασία και το φως ποικίλουν και τα ζώα μετακινούνται. Οι αισθήσεις και οι πράξεις είναι αβέβαιες. Η αντίληψη είναι αρκετά περιορισμένη και εμποδίζεται λόγω του θορύβου, έτσι το ζώο δεν αντιλαμβάνεται σωστά οτιδήποτε που δεν είναι δίπλα του, εκτός αν επιλέξει να εκτεθεί κοιτώντας γύρω του σε ανοιχτό

χώρο. Η επιτυχία του τρωκτικού είναι ο αριθμός των φορών που έχει βρει ταίρι κατά τη διάρκεια της ζωής του. Αυτό συσχετίζεται κατά κάποιο τρόπο με τη διάρκεια της ζωής του, αλλά μια μακροζωία δεν εγγυάται και πολλές ευκαιρίες αναπαραγωγής. Η επιλογή δράσης (action selection) του τρωκτικού φαίνεται παρακάτω:



Οι κάθετες γραμμές είναι BRPs με μεγαλύτερη προτεραιότητα του στοιχείου κορυφής, όπως συνήθως. Το D δηλώνει το drive collection και το C το competence. Δύο από τα στοιχεία του drive collection έχουν ένα παράγοντα προγραμματισμού (schedule), επειδή αυτό είναι ένα σύστημα διακριτού χρόνου και όχι ένα σύστημα πραγματικού χρόνου, ο προγραμματισμός είναι για Ν κύκλους, έτσι το [1 :: 5] σημαίνει ότι αν αυτό το στοιχείο έχει την υψηλότερη προτεραιότητα και δεν έχει ενεργοποιηθεί σε πέντε κύκλους, θα ενεργοποιηθεί. Τα κουτιά δηλώνουν τα action patterns που στη συγκεκριμένη υλοποίηση είναι η μόνη δομή που έχει πρωταρχικές ενέργειες και αποτελούνται από μόνο ένα στοιχείο.

Η σειρά των βημάτων που ακολουθείται στο συγκεκριμένο πλάνο (η λογική είναι ίδια για όλα τα POSH πλάνα) είναι η εξής:

Αρχικά το πλάνο ελέγχει το πρώτο drive element (λόγω προτεραιότητας) που στην περίπτωσή μας είναι το flee. Αν αυτό ικανοποιείται δηλαδή αν το sniff_predator (αίσθηση) επιστρέφει true, τότε ενεργοποιείται το competence που ακολουθεί και ελέγχεται αν ικανοποιείται το πρώτο σε προτεραιότητα στοιχείο του competence που είναι το freeze (για να

ικανοποιείται πρέπει να επιστρέφουν και οι τρεις αισθήσεις του see_predator, covered, hawk την τιμή true). Αν συμβεί αυτό, τότε εκτελείται το action pattern που αποτελείται από μια μόνο πρωταρχική ενέργεια την hold_still. Έπειτα ο έλεγχος επιστρέφει στο Drive Collection, όπου και ελέγχεται πάλι ποιο Drive element μπορεί να ικανοποιηθεί τώρα, με τη λογική ότι μπορεί να έχουν αλλάξει κάποιες συνθήκες όσον αφορά την κατάσταση του πράκτορα και του περιβάλλοντος. Αν το πρώτο Drive element δεν ικανοποιείται πια, ο έλεγχος γίνεται στο δεύτερο. Αν αυτό ικανοποιείται δηλαδή sniff_mate=true, τότε ενεργοποιείται το αντίστοιχο competence και ξεκινάει ο έλεγχος για το αν ικανοποιείται το πρώτο στοιχείο του competence, το inseminate, και συνεχίζει τον έλεγχο κατά τον ίδιο τρόπο. Η πορεία που ακολουθείται κατά την εκτέλεση βασίζεται στην εικόνα 5 της ενότητας 3.1.4.

Η εμφάνιση του πλάνου του παραδείγματος διαφέρει από το προηγούμενο παράδειγμα που συναντήσαμε στο προηγούμενο κεφάλαιο. Η δομή όμως είναι η ίδια. Για παράδειγμα το Competence που ενεργοποιείται από το Drive element exploit γράφεται αλλιώς ως εξής:

Exploit	(needed_res_avail T) → use_resource → leave	*(το T σημαίνει true , και το ⊥ false)*

Και το use_resource είναι ένα action pattern με ένα element το exploit_resource το οποίο είναι πρωταρχική ενέργεια.

3.4.Behaviour Oriented Design (BOD)

Όπως αναφέρθηκε και πιο πάνω, τα Posh reactive plans αποτελούν βασικό κομμάτι της μεθοδολογίας BOD, γι αυτό και θα αναφερθούν κάποια βασικά στοιχεία της.

Το BOD είναι μια μεθοδολογία κατασκευής σύνθετων πρακτόρων. Δημιουργήθηκε από την Joanna Bryson στα πλαίσια της διδακτορικής της διατριβής το 2002. Το BOD είναι σχεδιασμένο με τέτοιο τρόπο, ώστε να μπορεί να εφαρμοστεί σε πολλές γλώσσες προγραμματισμού. Η ονομασία BOD προέρχεται από την Behavior-Based Artificial Intelligence(BBAI) [Brooks, 1991a, Maes, 1991a, Matari´c, 1997], και από το Object-Oriented Design (OOD) [e.g. Coad et al., 1997]. Η Behavior-Based Τεχνητή Νοημοσύνη είναι ένας τρόπος σχεδιασμού, ο οποίος χωρίζει τη νοημοσύνη σε συμπεριφορές, όπως το περπάτημα ή το να τρώει κάποιος, παρά σε γενικές διαδικασίες όπως το να αισθάνεται κάποιος ή να σχεδιάζει. Οι συμπεριφορές είναι υπομονάδες (modules) που περιγράφονται σαν σύνολα από πράξεις-ενέργειες που ενημερώνονται από τη διαδικασία της αίσθησης. Αυτή η διαδικασία της αίσθησης πρέπει να ενημερώνει το πότε οι ενέργειες πρέπει να πραγματοποιηθούν και το πώς.

Σχετικά με το BOD πρέπει να σημειωθεί ότι μια απλή διαδικασία αίσθησης είναι σπάνια επαρκής για την ανίχνευση του περιβάλλοντος ή για τον έλεγχο των πράξεων. Τα δύο παραπάνω χαρακτηριστικά του BOD απαιτούν την ύπαρξη πλήρης αντίληψης που με τη σειρά της απαιτεί την ύπαρξη μνήμης. Η ύπαρξη μνήμης βοηθά στο να αποθηκεύει τις εμπειρίες και να δημιουργεί προσδοκίες. Αυτή η παρατήρηση έχει δύο συνέπειες πάνω στη BOD μεθοδολογία. Η πρώτη είναι ότι η μνήμη γίνεται ένα κομμάτι της συμπεριφοράς .Στην πραγματικότητα, οι απαιτήσεις της μνήμης παίζουν σημαντικό ρόλο στην διαδικασία αποσύνθεσης της συμπεριφοράς (η διαδικασία καθορισμού τού πώς θα χωριστεί η νοημοσύνη σε σύνολο υπομονάδων). Αυτή η στρατηγική είναι ανάλογη με το κεντρικό δόγμα ενός object oriented σχεδιασμού (ο χωρισμός του συστήματος σε υποομάδες καθορίζεται καλύτερα από τις απαιτήσεις της κάθε κατάστασης). Η δεύτερη συνέπεια είναι ότι, επειδή ο καθορισμός του περιβάλλοντος είναι αρκετά σημαντικός αλλά και αρκετά δύσκολος, χρειάζεται τη δική του αναπαράσταση. Οι αποφάσεις για τον έλεγχο του περιβάλλοντος συχνά πρέπει να διατηρούνται, όταν το αρχικό έναυσμα δεν είναι πλέον αισθητό. Το BOD χρησιμοποιεί ιεραρχικές δομές αντιδραστικού σχεδιασμού, για να εξασφαλίσει την παρακολούθηση του περιβάλλοντος και των αποφάσεων για τον έλεγχο του πλαισίου-περιβάλλοντος.

Η επιρροή του OOD στο BOD ξεκινά με την αναλογία μεταξύ της συμπεριφοράς και του αντικειμένου. Οι συμπεριφορές κωδικοποιούνται σαν αντικείμενα, και τα αρχικά στοιχεία των αντιδραστικών πλάνων του BOD κωδικοποιούνται σαν μέθοδοι των αντικειμένων συμπεριφοράς. Εξίσου σημαντική είναι και η διαδικασία σχεδιασμού. Όπως και στην OOD, έτσι και στην BOD τονίζεται ο κυκλικός σχεδιασμός. Η διαδικασία ανάπτυξης ενός πράκτορα εναλλάσσεται μεταξύ της ανάπτυξης βιβλιοθηκών για τις συμπεριφορές και της ανάπτυξης αντιδραστικών πλάνων (reactive plans) που ελέγχουν τις εκφράσεις των συμπεριφορών αυτών. Το BOD παρέχει κατευθυντήριες γραμμές όχι μόνο για την ανάπτυξη της αρχικής αποσύνθεσης της συμπεριφοράς αλλά και για την αναγνώριση τού πότε μια αποσύνθεση είναι ανεπαρκής. Αυτή η επαναληπτική διαδικασία ανάπτυξης έχει ως αποτέλεσμα τη συνεχή βελτιστοποίηση ενός BOD πράκτορα όσον αφορά την απλότητα, τη σαφήνεια, την επεκτασιμότητα και την ορθότητα.

Συνοπτικά, το BOD αποτελεί έναν τρόπο για την ανάπτυξη συστημάτων Τεχνητής Νοημοσύνης και απαιτεί τα εξής:

- Μια βιβλιοθήκη για τη συμπεριφορά του πράκτορα προγραμματισμένη σε οποιαδήποτε αντικειμενοστρεφή γλώσσα προγραμματισμού και
- Μια έκδοση ενός POSH μηχανισμού επιλογής ενέργειας για αυτή τη γλώσσα προγραμματισμού.

3.4.1. Κύκλος ανάπτυξης του BOD (BOD Development Cycle)

Ο κύκλος ανάπτυξης του BOD αποτελείται από τρία βασικά επίπεδα τα οποία είναι τα εξής:

1.Η αρχική αποσύνθεση (The Initial Decomposition)

Η αρχική αποσύνθεση αποτελείται από ένα σύνολο βημάτων τα οποία είναι τα εξής:

1. Προσδιορισμός του τι πρόκειται να κάνει ο πράκτορας σε υψηλό επίπεδο
2. Περιγραφή πιθανών δραστηριοτήτων σε μορφή ακολουθιών. Αυτές οι ακολουθίες των πράξεων είναι η βάση των αρχικών αντιδραστικών πλάνων (reactive plans).
3. Προσδιορισμός μιας αρχικής λίστας από αρχικές ενέργειες και αισθήσεις (primitives,act-sense) που προέρχεται από το βήμα 2.
4. Προσδιορισμός της αναγκαίας κατάστασης που επιτρέπει την περιγραφή των primitives και των drives. Ομαδοποίηση των συναφών στοιχείων κατάστασης σε συμπεριφορές. Αυτή είναι η βάση της δημιουργίας βιβλιοθήκης συμπεριφοράς.
5. Προσδιορισμός και ιεράρχηση των στόχων ή των κινήσεων (drives) που ο πράκτορας σκοπεύει να κάνει. Αυτό περιγράφει τις αρχικές ρίζες για την ιεραρχία του POSH action selection.
6. Επιλογή της πρώτης συμπεριφοράς για εφαρμογή.

Οι λίστες που μεταγλωττίζονται (compile) κατά τη διάρκεια αυτής της διαδικασίας θα πρέπει να διατηρούνται, δεδομένου ότι είναι ένα σημαντικό μέρος της τεκμηρίωσης του πράκτορα. Κατά τη διάρκεια επιλογής της πρώτης συμπεριφοράς καλό είναι να επιλέγεται μια χαμηλού επιπέδου προτεραιότητα που μπορεί να είναι συνεχώς ενεργή, ώστε ο πράκτορας να μην «πεθαίνει» αμέσως.

2. Επαναληπτική Ανάπτυξη (Iterative Development)

Η καρδιά της μεθοδολογίας BOD είναι η επαναληπτική διαδικασία ανάπτυξης:

1. Επιλογή ενός μέρους της προδιαγραφής για να εφαρμοστεί έπειτα.
2. Επέκταση του πράκτορα με αυτή την υλοποίηση:
    - Κωδικοποίηση των συμπεριφορών και των αντιδραστικών πλάνων

- Δοκιμή και αποφυγή λαθών (debug) του κώδικα.

3. Επανεξέταση της τρέχουσας προδιαγραφής.

Το πρώτο βήμα είναι η επεξεργασία του τρέχοντος μοντέλου και το επόμενο είναι η επανεξέταση του μοντέλου για την εύρεση της βέλτιστης αναπαράστασης. Βέβαια, ανεξάρτητα από τη διαδικασία της βελτιστοποίησης, ο πράκτορας θα συνεχίσει να αναπτύσσεται με πολυπλοκότητα. Αλλά εάν αυτή η ανάπτυξη παρακολουθείται προσεκτικά, κατευθύνεται και περιορίζεται όπου χρειάζεται, τότε το αποτέλεσμα θα είναι ένας πράκτορας πιο «κομψός» που μπορεί εύκολα να διατηρηθεί και να δεχτεί περεταίρω προσαρμογή. Αντίθετα με τις συμπεριφορές, που εύκολα κωδικοποιούνται απευθείας σε μια αντικειμενοστραφής γλώσσα προγραμματισμού, τα αντιδραστικά πλάνα (reactive plans) αποθηκεύονται σε script files. Το πλάνο κανονικά διαβάζεται όταν αρχικοποιείται ο πράκτορας, ή αλλιώς «γεννιέται», αν και θεωρητικά, νέα πλάνα θα μπορούσαν να προστεθούν κατά τη διάρκεια της εκτέλεσης. Τα αντιδραστικά πλάνα ενός πράκτορα αυξάνουν την πολυπλοκότητά τους κατά τη διάρκεια της ανάπτυξης. Επίσης, συχνά πολλαπλά αντιδραστικά πλάνα αναπτύσσονται για μια ενιαία πλατφόρμα Τεχνητής Νοημοσύνης (και ένα σύνολο ενοτήτων συμπεριφοράς), καθένα από τα οποία δημιουργεί πράκτορες με διαφορετικά χαρακτηριστικά όπως οι στόχοι ή η προσωπικότητα. Ακόμα και αν υπάρχουν διαφορετικά plan scripts για την ίδια πλατφόρμα, γενικά θα υπάρχει μια βιβλιοθήκη συμπεριφοράς, ένα σύνολο κώδικα. Φυσικά, κάθε πράκτορας θα έχει τη δική του διαδικασία ή συμπεριφορά, όταν εκτελείται και θα μπορεί να αποθηκεύει την κατάσταση του χρόνου εκτέλεσης στο δικό του αποθηκευτικό χώρο. Αλλά αξίζει να γίνει προσπάθεια ώστε να υποστηρίζονται όλα τα script σε μια ενιαία βιβλιοθήκη συμπεριφοράς. Η δοκιμή θα πρέπει να γίνεται όσο το δυνατόν συχνότερα. Χρησιμοποιώντας γλώσσες που δεν χρειάζονται μεταγλώττιση, όπως η lisp ή η perl, αυξάνεται η ταχύτητα της διαδικασίας ανάπτυξης αν και μπορεί να επιβραδύνουν το χρόνο εκτέλεσης του προγράμματος.

3. Αναθεώρηση των προδιαγραφών (Revising the Specifications)

Το πιο ενδιαφέρον μέρος της μεθοδολογίας BOD είναι το σύνολο των κανόνων για την αναθεώρηση των προδιαγραφών. Γενικά, η βασική σχεδιαστική αρχή του BOD είναι «σε περίπτωση αμφιβολίας, ευνοείται η απλότητα» . Ένα primitive προτιμάται από ένα sequence και ένα sequence από ένα competence. Παρομοίως η κατάσταση ελέγχου προτιμάται από την κατάσταση εκμάθησης και η ειδική εκμάθηση από την γενικού σκοπού εκμάθηση ή σχεδιασμό.

Μια κατευθυντήρια αρχή σε όλη την τεχνολογία λογισμικού είναι η μείωση των πλεονασμών. Εάν ένα πλάνο ή συμπεριφορά είναι δυνατό να ξαναχρησιμοποιηθεί, τότε πρέπει να ξαναχρησιμοποιηθεί. Εάν ένα μέρος του πλάνου ή της αρχικής δράσης (primitive) μπορεί να χρησιμοποιηθεί, τότε πρέπει να γίνει αλλαγή στην αρχική αποσύνθεση. Στην περίπτωση της αρχικής δράσης, αυτή πρέπει να χωριστεί σε δύο ή περισσότερες αρχικές δράσεις και η πρώτη δράση να αντικατασταθεί με στοιχείο πλάνου. Το καινούριο αυτό στοιχείο πλάνου πρέπει να έχει το ίδιο όνομα και την ίδια λειτουργία με την αρχική δράση.

Εάν μια ακολουθία (action pattern) χρειάζεται να περιέχει κύκλο, ή δε χρειάζεται μερικά από τα στοιχεία της, τότε είναι ένα competence και όχι ένα action pattern. Εάν ένα competence εκτελείται σχεδόν πάντα μέσα από μια συγκεκριμένη σειρά των στοιχείων της, τότε θα πρέπει να απλοποιηθεί σε action pattern.

Στην παραπάνω ενότητα περιγράφτηκε η λογική και η μεθοδολογία που ακολουθήθηκε στη συγκεκριμένη εργασία για τη διαδικασία παραγωγής της συμπεριφοράς ενός πράκτορα. Παρακάτω ακολουθούν τα κεφάλαια που περιγράφουν τα προγράμματα με τα οποία έγινε αυτή η υλοποίηση και στη συνέχεια ακολουθεί το πείραμα που περιγράφει αναλυτικά τη διαδικασία παραγωγής συμπεριφοράς πρακτόρων.

4.Pogamut

Όπως προαναφέρθηκε στο κεφάλαιο 1, το Pogamut είναι μια πλατφόρμα λογισμικού για την ανάπτυξη συμπεριφοράς των ευφυών εικονικών πρακτόρων που «ζουν» σε εικονικά

περιβάλλοντα. Το Pogamut είναι κατά κύριο λόγο προσαρμοσμένο στον τρισδιάστατο εικονικό κόσμο του βιντεοπαιχνιδιού Unreal Tournament 2004, αλλά παράλληλα μπορεί να συνδεθεί και με άλλους εικονικούς κόσμους. Η αρχιτεκτονική του είναι αρκετά ευέλικτη και ενσωματώνει τα εξής:

- Ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) με δυνατότητα εκσφαλμάτωσης (debugging).
- Μια βιβλιοθήκη με πρωταρχικές κινήσεις και αισθήσεις, με αλγόριθμους εύρεσης μονοπατιού (path-finding algorithms) και με τεχνικές δυναμικών πλάνων για τον έλεγχο των εικονικών ανθρώπων.
- Εργαλεία για τον καθορισμό και την εκτέλεση πειραμάτων.

Επί του παρόντος, η πλατφόρμα είναι καταλληλότερη για τη διερεύνηση των μηχανισμών ελέγχου των εικονικών ανθρώπων σε βιντεοπαιχνίδια, ωστόσο μπορεί να χρησιμοποιηθεί από οποιονδήποτε ενδιαφέρεται για τη δημιουργία πειραμάτων που χρησιμοποιούν μέχρι και 10 εικονικούς ανθρώπους που κατοικούν σε εικονικό κόσμο μεσαίου μεγέθους (πχ. 200 m^2). Επίσης μπορεί να χρησιμοποιηθεί σαν εργαλείο σε πανεπιστήμια όπου οι φοιτητές μπορούν να εξασκούνται και να πειραματίζονται στην ανάπτυξη συμπεριφορών σε ευφυείς εικονικούς πράκτορες. Προς το παρόν η πλατφόρμα δεν είναι κατάλληλη για προσομοιώσεις μεγάλων πληθών (e.g. Badler et al., 2008).

Η πλατφόρμα έχει εμπλουτιστεί με διάφορα χαρακτηριστικά που διευκολύνουν την έρευνα στους τομείς της εικονικής αφήγησης (virtual storytelling) και της υπολογιστικής γνωστικής ψυχολογίας (computational cognitive psychology). Αυτά τα χαρακτηριστικά είναι τα εξής :

- Ενσωμάτωση της γενικής γνωστικής αρχιτεκτονικής ACT-R (Anderson, 2007),
- Ένα εργαλείο που επιτρέπει τον έλεγχο των κινήσεων
- Ένα γενικό διαχειριστή ιστοριών (generic story manager)
- Ένα γενικό module που προσθέτει αισθήματα στους πράκτορες
- Ένα module επεισοδιακής μνήμης
- Και σύνδεση του Pogamut με άλλα τρισδιάστατα εικονικά περιβάλλοντα

Η αρχιτεκτονική του Pogamut διαθέτει τέσσερα συστατικά στοιχεία τα οποία είναι :

1. έναν προσομοιωτή του εικονικού κόσμου
2. μια διεπαφή γραφικών για το χρήστη (GUI)
3. μια βιβλιοθήκη με βασικές κλάσεις για τον προγραμματισμό της συμπεριφοράς των ευφυών εικονικών πρακτόρων (IVA), που συμπεριλαμβάνουν τις πρωταρχικές κινήσεις και αισθήσεις τους
4. ένα ενδιάμεσο λογισμικό χειρισμού της επικοινωνίας μεταξύ (1) και (3).

Η διαδικασία σχεδιασμού των ευφυών εικονικών πρακτόρων μπορεί να περιγραφεί πολύ πρόχειρα ως εξής: Αρχικά ο προγραμματιστής δημιουργεί ένα ευφυή εικονικό πράκτορα είτε χρησιμοποιώντας έναν εξωτερικό πρόγραμμα επεξεργασίας (editor) είτε το GUI άμεσα. Και στις δύο περιπτώσεις θα χρησιμοποιηθεί ο κώδικας της βιβλιοθήκης του Pogamut. Έπειτα ο προγραμματιστής θα χρησιμοποιήσει το GUI για την εκτέλεση του ευφυούς πράκτορα στον εικονικό κόσμο, αλλά και για την εκσφαλμάτωση του κώδικά του. Γενικότερα το Pogamut επιτρέπει τη δημιουργία ενός πράκτορα σε Java (και όχι μόνο) ο οποίος θα μπορεί να :

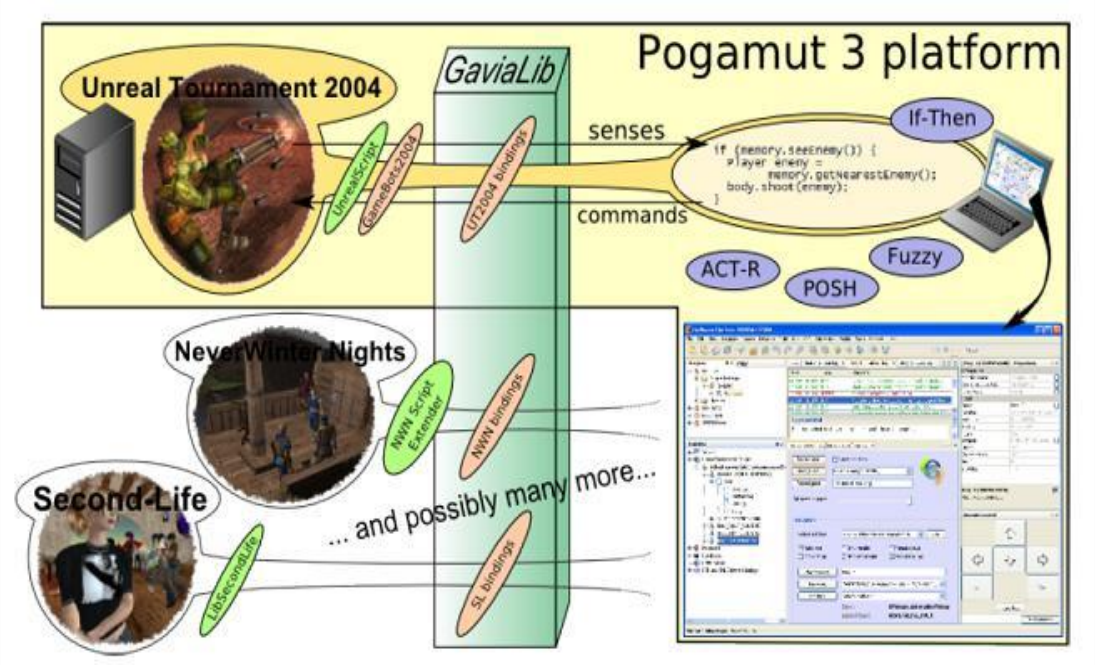
- δέχεται μηνύματα (αισθήσεις) από το περιβάλλον εκτέλεσης (εικονικό περιβάλλον)
- αποφασίζει την επόμενη (-ες) ενέργειά του βάσει των πληροφοριών αίσθησης από τον εικονικό κόσμο και της εσωτερικής του κατάστασης
- και να παράγει εντολές προς το περιβάλλον για την εκτέλεση της ενέργειας.

4.1.Ιστορία ανάπτυξης του Pogamut

Η πρώτη έκδοση της πλατφόρμας Pogamut θεωρείται το Pogamut 1. Αυτό ήταν προσαρμοσμένο με το βιντεοπαιχνίδι Unreal tournament 1999 (Epic, 1999) και ήταν σχεδιασμένο σαν μια βιβλιοθήκη γραμμένη σε Python με ένα απλό γραφικό περιβάλλον για την εκτέλεση του πράκτορα και είχε περιορισμένες δυνατότητες εκσφαλμάτωσης (Bvda et al., 2006). Ο διάδοχός του είναι το Pogamut 2 (Kadlec et al., 2007) και είναι βασισμένο σε Java γλώσσα και το γραφικό του περιβάλλον είναι ένα ολοκληρωμένο περιβάλλον ανάπτυξης (IDE) που έχει αναπτυχθεί σαν plugin στο Netbeans. Επίσης το Pogamut 2 προσαρμόστηκε στην επόμενη και πιο εξελιγμένη έκδοση του Unreal, το Unreal Tournament 2004 (Epic, 2004) για το οποίο θα μιλήσουμε και παρακάτω. Επιπλέον χρησιμοποιήθηκε σαν βάση στο διαγωνισμό 2K Bot Prize που πραγματοποιήθηκε στην Αυστραλία στον οποίο οι διαγωνιζόμενοι κλήθηκαν να αναπτύξουν έναν πράκτορα που μπορεί να ξεγελάσει τους κριτές ότι πρόκειται για ένα άνθρωπο. Τέλος δημιουργήθηκε το Pogamut 3 () το οποίο είναι παρόμοιο με το 2 αλλά με κάποια πρόσθετα χαρακτηριστικά όπως τη δυνατότητα σύνδεσής του με άλλα εικονικά περιβάλλοντα (εκτός του Unreal Tournament).

4.2.Γενικά πώς λειτουργεί το Pogamut3

Η καρδιά του Pogamut3 είναι η βιβλιοθήκη GaviaLib (General Autonomous Virtual Intelligent Agents Library). Η GaviaLib σχεδιάστηκε σαν ένα ενδιάμεσο λογισμικό που μεσολαβεί στην επικοινωνία μεταξύ του εικονικού κόσμου (αριστερά στο σχήμα παρακάτω) και της λογικής του πράκτορα (δεξιά στο σχήμα παρακάτω). Το Pogamut 3 χρησιμοποιεί το UT2004 γι' αυτό και η Gavialib έχει υλοποιηθεί με αυτό τον τρόπο, ώστε να δημιουργεί το «δέσιμο» με το συγκεκριμένο εικονικό κόσμο (εικόνα παρακάτω – UT2004 bindings). Παρόλα αυτά το Pogamut3 είναι τώρα δυνατό να προσαρμοστεί και με άλλα περιβάλλοντα όπως πχ. το περιβάλλον του Quake III (Id software, 1999). Γενικά η Gavialib είναι ένας μεταφραστής που μεταφράζει το java κώδικα του πράκτορα από το Pogamut στη γλώσσα του εικονικού περιβάλλοντος (πχ για το Unreal Tournament είναι η Unreal Script η οποία αναφέρεται και στο επόμενο κεφάλαιο) και το αντίστροφο. Ουσιαστικά στο Pogamut σχεδιάζει το μυαλό του πράκτορα, ενώ το εικονικό του σώμα βρίσκεται στο UT2004.



Εικόνα 6 : επικοινωνία μεταξύ του εικονικού κόσμου και της λογικής του πράκτορα μέσω GaviaLib.

Ο αγωγός επικοινωνίας (pipeline) μεταξύ του εικονικού κόσμου και της λογικής του πράκτορα (δηλαδή η μεταφορά των γεγονότων που συμβαίνουν στον εικονικό κόσμο) και αντίστροφα (δηλαδή η μεταφορά εντολών από τον πράκτορα) μπορεί εύκολα να τροποποιηθεί από έναν προγραμματιστή που θέλει να συνδέσει τη GaviaLib με ένα άλλο εικονικό κόσμο ή να επεκτείνει το πρωτόκολλο επικοινωνίας με το UT2004 (δηλαδή τα Gamebots που αναφέρουμε παρακάτω). Επίσης ο αγωγός αυτός μοιάζει σαν ένα υψηλού επιπέδου API (application programming interface) στα μάτια ενός προγραμματιστή για ευφυείς εικονικούς πράκτορες. Από τη μια πλευρά, αυτό περιορίζει τους προγραμματιστές να χρησιμοποιούν μόνο τα δοθέντα εικονικά περιβάλλοντα, από την άλλη πλευρά το API τους επιτρέπει να έχουν εύκολη πρόσβαση στις πρωταρχικές κινήσεις και αισθήσεις του πράκτορα, έτσι το μόνο που χρειάζονται να προγραμματίσουν είναι το τι θα κάνει ο πράκτορας και πότε θα το κάνει χωρίς να χρειάζεται να γνωρίζουν το πώς λειτουργεί η επικοινωνία χαμηλού επιπέδου. Επίσης η διεπαφή της GaviaLib επιτρέπει και τον έλεγχο του σώματος του πράκτορα μέσω δικτύου. Τέλος η GaviaLib μπορεί να χρησιμοποιηθεί και από άλλες βοηθητικές μονάδες, όπως το POSH (Bryson, 2001a) στην περίπτωση που θέλουμε να προσθέσουμε δυναμικά πλάνα για επιλογή ενέργειας.

Ο τρόπος με τον οποίο επικοινωνεί το Pogamut (μέσω της GaviaLib) με το εικονικό περιβάλλον (UT2004) περιγράφεται αναλυτικά στο επόμενο κεφάλαιο.

5.Το Περιβάλλον του Unreal Tournament 2004

Όπως αναφέραμε και προηγουμένως η πλατφόρμα Pogamut χρησιμοποιεί το περιβάλλον του Unreal Tournament (συγκεκριμένα το Pogamut3 χρησιμοποιεί το Unreal Tournament 2004).

Το Unreal Tournament 2004 (UT2004) είναι ένα παιχνίδι πρώτου προσώπου των εταιριών Epic Games και Digital Extremes, το οποίο σχεδιάστηκε κυρίως για multiplayer παιχνίδια παρόλο που περιέχει και υποστήριξη για παιχνίδι single-player, όπου το ρόλο των συμπαιχτών ή αντιπάλων αναλαμβάνουν τα AI-bots. Κυκλοφόρησε το 2004 σαν συνέχεια στη σειρά Unreal Tournament (το 1999) και Unreal Tournament 2003 (το 2003). Η μηχανή γραφικών που χρησιμοποιεί είναι η Unreal Engine 2.

Οι τύποι των παιχνιδιών (gametypes)που υποστηρίζει το UT2004 είναι οι εξής:

- Deathmatch:
  Ίσως ο πιο γνωστός τύπος παιχνιδιού στο οποίο ο κάθε παίχτης προσπαθεί να σκοτώσει όσο πιο πολλές φορές μπορεί τον αντίπαλό του.
- Team Deathmatch:
  Αυτό είναι παρόμοιο με το Deathmatch με τη διαφορά ότι οι παίχτες εδώ μπορούν να σχηματίσουν ομάδες. Επιτρέπει το σχεδιασμό μέχρι και τεσσάρων ομάδων: τους κόκκινους, τους μπλε, τους χρυσούς και τους πράσινους. Το ίδιο ισχύει και για τους τύπους (Capture the Flag και Domination).
- Capture The Flag
  Σε αυτό τον τύπο δύο ομάδες ανταγωνίζονται μεταξύ τους . Κάθε ομάδα έχει τη βάση της στην οποία βρίσκεται η σημαία της. Σκοπός της κάθε ομάδας είναι να κλέψει τη σημαία της αντίπαλης ομάδας και να την επιστρέψει στη βάση της, ενώ παράλληλα να προστατέψει και τη δική της σημαία.
- Domination
  Σε αυτό τον τύπο δύο ομάδες ανταγωνίζονται μεταξύ τους προσπαθώντας να λάβουν υπό την κατοχή τους διάφορα σημεία στο χάρτη ώστε να κερδίσουν βαθμούς και στη συνέχεια το παιχνίδι.
- Assault
  Σε αυτό τον τύπο παιχνιδιού υπάρχουν δύο αντίπαλες ομάδες, όπου η μία προσπαθεί να καταλάβει μια βάση και η άλλη προσπαθεί να αμυνθεί.
- Last Man Standing

Αυτός ο τύπος παιχνιδιού έχει αρκετές ομοιότητες με το Deathmatch. Όπως αναφέρει και το όνομά του νικητής είναι αυτός που θα καταφέρει να επιβιώσει από τη μάχη, δηλαδή θα παραμείνει στο παιχνίδι περισσότερο χρόνο από τους αντιπάλους του.

- Onslaught
  Στον τύπο αυτό λαμβάνουν χώρα δύο αντίπαλες ομάδες οι οποίες ξεκινούν από αντιδιαμετρικά σημεία στο χάρτη και προσπαθούν να πάρουν υπό την κατοχή τους όσα περισσότερα σημεία του χάρτη μπορούν, ώστε να φτάσουν και να χτυπήσουν τον πυρήνα της βάσης της αντίπαλης ομάδας.

Γενικότερα το UT2004 είναι ένα από τα λίγα παιχνίδια που έχουν ένα μέρος του κώδικά τους ανοιχτό για τροποποιήσεις. Ο πυρήνας του παιχνιδιού αποτελείται από την Unreal Engine 2 η οποία παρέχει τις βασικές λειτουργίες για την εκτέλεση του εικονικού κόσμου, όπως την απόδοση των 3D μοντέλων. Για την εκτέλεση βασικών λειτουργιών του πυρήνα οι δημιουργοί του παιχνιδιού έφτιαξαν την Unreal Script, μια γλώσσα προγραμματισμού ειδικά για το παιχνίδι. Τα game mechanics, όπως η λογική των bot, είναι προγραμματισμένα σε Unreal Script. Ενώ ο κώδικας της Unreal Engine δεν είναι ανοιχτός- προσπελάσιμος, ο κώδικας που είναι γραμμένος σε Unreal Script είναι ανοιχτός για τροποποιήσεις. Το γεγονός αυτό αλλά και με τη βοήθεια κάποιων εργαλείων (το UnrealEd που είναι ο virtual world editor του UT2004) επιτρέπει στους χρήστες τη δημιουργία νέων χαρτών και μοντέλων καθώς και νέων επεκτάσεων για το παιχνίδι (τα λεγόμενα mods). Αυτό καθιστά το UT2004 πολύ “ελκυστικό” για το χρήστη, εφόσον είναι σχεδιασμένο με τέτοιο τρόπο, ώστε με τη χρήση διάφορων εργαλείων να επιτρέπει στους προγραμματιστές να παρέμβουν στη δομή του παιχνιδιού και στους ερευνητές να πειραματιστούν σε θέματα τεχνητής νοημοσύνης πάνω σε αυτό.

5.1.Gamebots

Ένα περιβάλλον που δημιουργήθηκε πάνω στο UT2004 γι’ αυτό το σκοπό ,δηλαδή την παροχή πρόσβασης στους προγραμματιστές για να παρέμβουν στη δομή του παιχνιδιού, είναι τα Gamebots2004 (GB2004). Τα Gamebots είναι ένα mod για το UT2004. Ο αρχικός του σκοπός ήταν να κάνει διαθέσιμο τον εικονικό κόσμο του UT2004 σε εικονικούς πράκτορες που έχουν αναπτυχθεί χωρίς Unreal Script. Αυτό το επιτυγχάνουν παρέχοντας ένα δικτυακό TCP/IP text protocol για να ελέγχει τα avatars (εικονικά σώματα πρακτόρων) που βρίσκονται στο παιχνίδι μέσω εξωτερικών προγραμμάτων, όπως το Pogamut.

Το αρχικό σχέδιο των Gamebots (Adobbati et al., 2001) ξεκίνησε από τους Andrew N. Marshal και Gal Kaminka στο Πανεπιστήμιο της Νότιας Καλιφόρνια Information Sciences Institute. Τα Gamebots χρησιμοποιούσαν την παλαιότερη έκδοση του παιχνιδιού (Unreal Tournament 1999). Ο στόχος του σχεδίου ήταν να κάνει το περιβάλλον του Unreal Tournament αξιοποιήσιμο για την έρευνα στην τεχνητή νοημοσύνη. Το σχέδιο των Gamebots συνέχισαν οι Joe Manojlovich, Tim Garwood και Jessica Bayliss από το RIT πανεπιστήμιο (RIT,2005). Εν τω μεταξύ, το πρώτο σκέλος των Pogamut GameBots προέκυψε μέσα από την αποσφαλμάτωση της παλιάς έκδοσης των Andrew N. Marshal και Gal Kaminka και χρησιμοποιήθηκε από το Pogamut 1. Τελικά δημιουργήθηκε το GameBots 2004 από τους Jakub Gemrot, Cyril Brom, Rudolf Kadlec, Michal Bvda, Ondřej Burkert, Michal Zemčak, Radek Pvbil, Tomaš Plch στο πανεπιστήμιο της Πράγας , το οποίο συνδέεται στο UT2004 και χρησιμοποιείται από το Pgamut 2 και 3.

Υπάρχουν τρία είδη συνδέσεων στα GameBots τα οποία είναι τα εξής:

- **Σύνδεση Bot** (Bot Connection) - χρησιμοποιείται για τη δημιουργία και τον έλεγχο των bots στο περιβάλλον και ταυτόχρονα για την λήψη πληροφοριών για την κατάσταση των bots (δηλαδή πληροφορίες για την εσωτερική κατάστασή τους, όπως την ομάδα τους, την ενέργειά τους και την ασπίδα τους, αλλά και πληροφορίες για το τι βλέπουν, όπως αντικείμενα ή άλλους παίχτες).

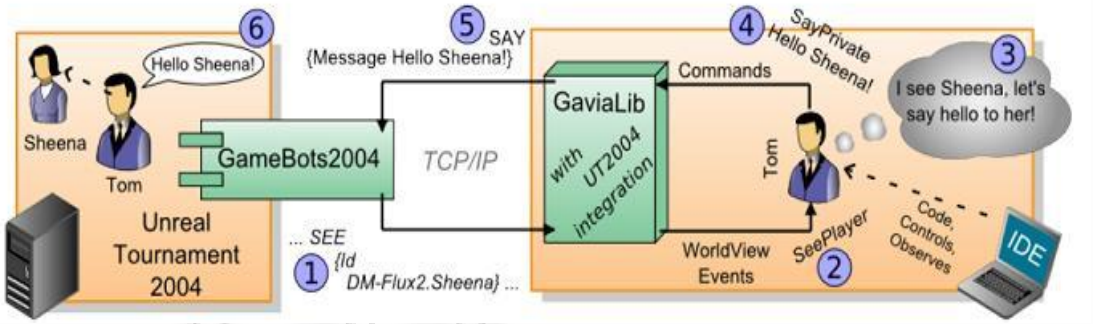
- **Σύνδεση ελέγχου** (Control Connection). - χρησιμοποιείται για τον έλεγχο του παιχνιδιού, δηλαδή για την παύση του παιχνιδιού, την αλλαγή του χάρτη κ.α.
- **Σύνδεση παρατηρητή** (Observer Connection). – χρησιμοποιείται για την παρατήρηση των bots ή των παιχτών στο παιχνίδι. Πληροφορίες για το τι κάνουν τα Bots ή οι παίχτες εξάγονται μέσω συγκεκριμένων μηνυμάτων από τα GameBots

Οι ανάγκες της εργασίας ικανοποιούνται με τη χρήση του πρώτου είδους σύνδεσης, δηλαδή της σύνδεσης Bot, αφού αυτό που μας ενδιαφέρει περισσότερο είναι ο έλεγχος των bot στο περιβάλλον και οι πληροφορίες για την εσωτερική τους κατάσταση αλλά και για το τι μπορούν να δουν γύρω τους στο περιβάλλον.

Κάθε πρόγραμμα για να συνδεθεί με τα GameBots χρησιμοποιεί το πρωτόκολλο επικοινωνίας TCP/IP. Το port που χρησιμοποιείται είναι το 3000 για τη σύνδεση bot, το 3001 για τη σύνδεση ελέγχου και το 3002 για τη σύνδεση παρατηρητή. Εφόσον η σύνδεση που χρησιμοποιήσαμε για την εργασία είναι η σύνδεση Bot, το port που χρησιμοποιήθηκε είναι το 3002.

6.Επικοινωνία Pogamut-UT2004 μέσω GaviaLib-Gamebots2004

Το GB2004 ξεκινά ένα διακομιστή (server) που παρέχει ένα μέσο ελέγχου των σωμάτων των πρακτόρων (avatars) μέσα στο UT2004. Τυπικά κάθε πράκτορας διαθέτει ένα instantiation της βιβλιοθήκης GaviaLib και συνδέεται στο διακομιστή αυτό σαν πελάτης (client). Όταν γίνει η σύνδεση στον διακομιστή του GB2004 , το GB2004 δημιουργεί το εικονικό σώμα του πράκτορα μέσα στο UT2004 και ο διακομιστής γίνεται τα μάτια και τα αυτιά του πράκτορα, στέλνοντας περιοδικά πληροφορίες αίσθησης και λαμβάνοντας τις εντολές από το μυαλό του (παρακάτω εικόνα).



Εικόνα 7: Η υψηλού επιπέδου αρχιτεκτονικής του Pogamut 3 που είναι ενσωματωμένο με τα GB2004 και το UT2004. Παράδειγμα επικοινωνίας.

Στην παραπάνω εικόνα απεικονίζεται ένα παράδειγμα μίας επανάληψης του κύκλου sense-rreason-act. Στην αριστερή πλευρά βρίσκεται ο διακομιστής του UT2004 με το Gamebots και δύο πράκτορες που βρίσκονται στον εικονικό κόσμο, τον Tom και τη Sheena. Το GB2004 αντιλαμβάνεται ότι ο Tom βλέπει τη Sheena και στέλνει την πληροφορία αυτή (μέσω TCP/IP πρωτοκόλλου) μέσω του μηνύματος SEE (1). Το SEE μεταφράζεται σε αντικείμενο της Java μέσα στην GaviaLib και παρουσιάζεται στο μηχανισμό επιλογής ενέργειας του Tom, μέσω του υψηλού επιπέδου API, με τη μορφή γεγονότος WorldView το SEEPlayer (2). Έπειτα ο μηχανισμός επιλογής ενέργειας αποφασίζει ότι πρέπει να χαιρετίσει τη Sheena (3) και δίνει την εντολή SayPrivate (4). Η εντολή αυτή μεταφράζεται σε μήνυμα χαμηλού επιπέδου SAY (5), όπου γίνεται αντιληπτό από το GB2004 το οποίο με τη σειρά του δίνει την εντολή στο εικονικό σώμα του Tom να χαιρετήσει τη Sheena.

Γενικά τα GB2004 λαμβάνουν εντολές από την GaviaLib-Pogamut και στέλνουν μηνύματα προς αυτή. Όλες οι εντολές και τα μηνύματα έχουν την ίδια μορφή κειμένου. Ένα παράδειγμα ενός μηνύματος των GameBots είναι το εξής:

PLAYER {Name GoodBot} {Location 1400,500,0} {Rotation 32000,0,0} {Health 100} {Height 170.59} {IsHuman False}

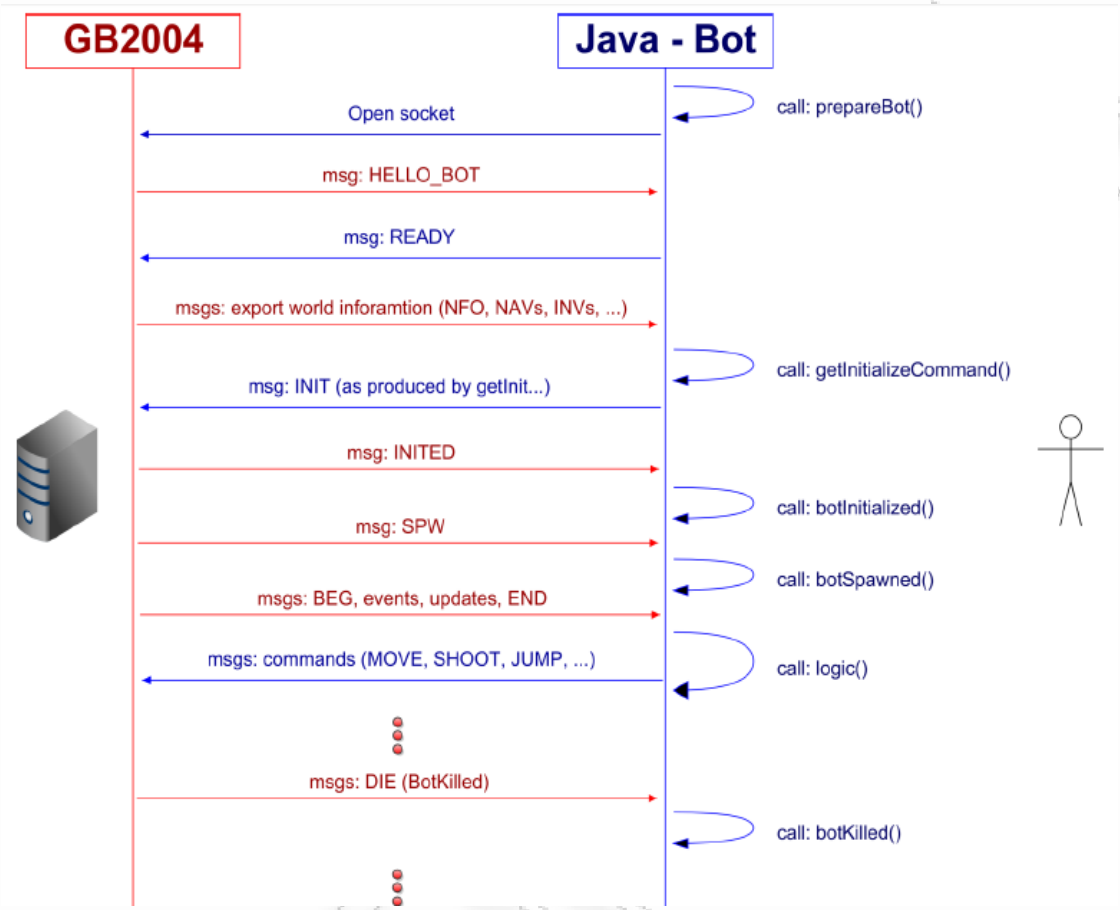
Οι πρώτοι χαρακτήρες μέχρι το πρώτο κενό είναι ο τύπος του μηνύματος. Έπειτα ακολουθούν σε ζεύγη τα χαρακτηριστικά που βρίσκονται μέσα σε αγκύλες ({}). Το ζεύγος κάθε χαρακτηριστικού αποτελείται από το όνομα του χαρακτηριστικού και την τιμή του. Έτσι στο παράδειγμά μας έχουμε τα εξής:

- Τύπος Μηνύματος – PLAYER
- 1^{ο} χαρακτηριστικό – Το όνομα του χαρακτηριστικού είναι “Name”. Η τιμή του είναι “GoodBot”.
- 2^{ο} χαρακτηριστικό – Το όνομα του χαρακτηριστικού είναι “Location”. Η τιμή του είναι “1400,500,0”.
- 3^{ο} χαρακτηριστικό – Το όνομα του χαρακτηριστικού είναι “Rotation”. Η τιμή του είναι “32000,0,0”.
- 4^{ο} χαρακτηριστικό – Το όνομα του χαρακτηριστικού είναι “Health”. Η τιμή του είναι “100”.
- 5^{ο} χαρακτηριστικό – Το όνομα του χαρακτηριστικού είναι “Height”. Η τιμή του είναι “170.69”.
- 6^{ο} χαρακτηριστικό – Το όνομα του χαρακτηριστικού είναι “IsHuman”. Η τιμή του είναι “False”.

Δεν υπάρχει περιορισμός στο πλήθος των χαρακτηριστικών όσο αφορά τα μηνύματα. Οι εντολές όμως έχουν περιορισμό, για παράδειγμα στα Gamebots του Pogamut είναι 32.

Τα μηνύματα και οι εντολές που ανταλλάσσονται κατά τη διάρκεια της δημιουργίας μιας νέας σύνδεσης ενός πράκτορα με τον εικονικό κόσμο αλλά και η σειρά με την οποία λαμβάνουν χώρα είναι τα εξής (εικόνα 8) :

- **HELLO_BOT** (HelloBotHandshake): Είναι το πρώτο μήνυμα που στέλνει το UT για να ξεκινήσει η σύνδεση με τον πράκτορα.
- **READY:** Εντολή που στέλνει το Pogamut σαν απάντηση στο HELLO_BOT.
- **NFO** (GameInfo): Μήνυμα που στέλνει το GB2004, αφού έχει λάβει την εντολή READY και δίνει πληροφορίες για το παιχνίδι, ποιος είναι ο τύπος του παιχνιδιού, αριθμός ομάδων κλπ.
- **NAV** (NavPoint): Μήνυμα που στέλνει το GB2004 το οποίο περιέχει πληροφορίες για τα navigation points του χάρτη (Πχ Τοποθεσία, όνομα κλπ.
- **INV** (Item): Μήνυμα που στέλνει το GB2004 το οποίο περιέχει πληροφορίες για τα αντικείμενα που βρίσκονται στο χάρτη (π.χ τοποθεσία, όνομα, αν είναι ορατό, ποσότητα κλπ)
- **INIT** (Initialize) : Εντολή από το Pogamut για τη δημιουργία του πράκτορα στο παιχνίδι. Περιέχει πληροφορίες, όπως το όνομα του πράκτορα, η μορφή του (skin) κλπ.)
- **INITED** (InitedMessage): Μήνυμα από τα GB2004 σαν απάντηση προς την εντολή INIT που περιέχει πληροφορίες για τα χαρακτηριστικά του πράκτορα, όπως την ταχύτητά του, τη μέγιστη ενέργειά του κ.α.
- **SPW** (Spawn) : Μήνυμα που στέλνουν τα GB2004 για τη δημιουργία του πράκτορα στον εικονικό κόσμο.
- **BEG** (BeginMessage), **events**, **updates**, **END** (EndMessage) : Σειρά από batch μηνύματα που στέλνουν τα GB2004 και περιέχουν πληροφορίες για τα γεγονότα που συμβαίνουν στο εικονικό περιβάλλον ή για τις αλλαγές . Αυτά ξεκινούν πάντα με το μήνυμα BEG και τελειώνουν με το END.
- **MOVE, SHOOT, JUMP** : Εντολές που στέλνει το Pogamut για να πραγματοποιηθούν από ο εικονικό σώμα του πράκτορα. Εδώ υπάρχουν διάφορες εντολές που μπορούν να σταλούν.
- **DIE** : Μήνυμα που στέλνεται από το GB2004, όταν ο πράκτορας πεθάνει.



Εικόνα 8 : Διαδικασία επικοινωνίας για τη δημιουργία του πράκτορα σε ένα παιχνίδι. Δεξιά με τα μπλε γράμματα φαίνονται οι συναρτήσεις σε Java που δημιουργούν τις αντίστοιχες εντολές που στέλνει το Pogamut (Java-Bot) στο GB2004.

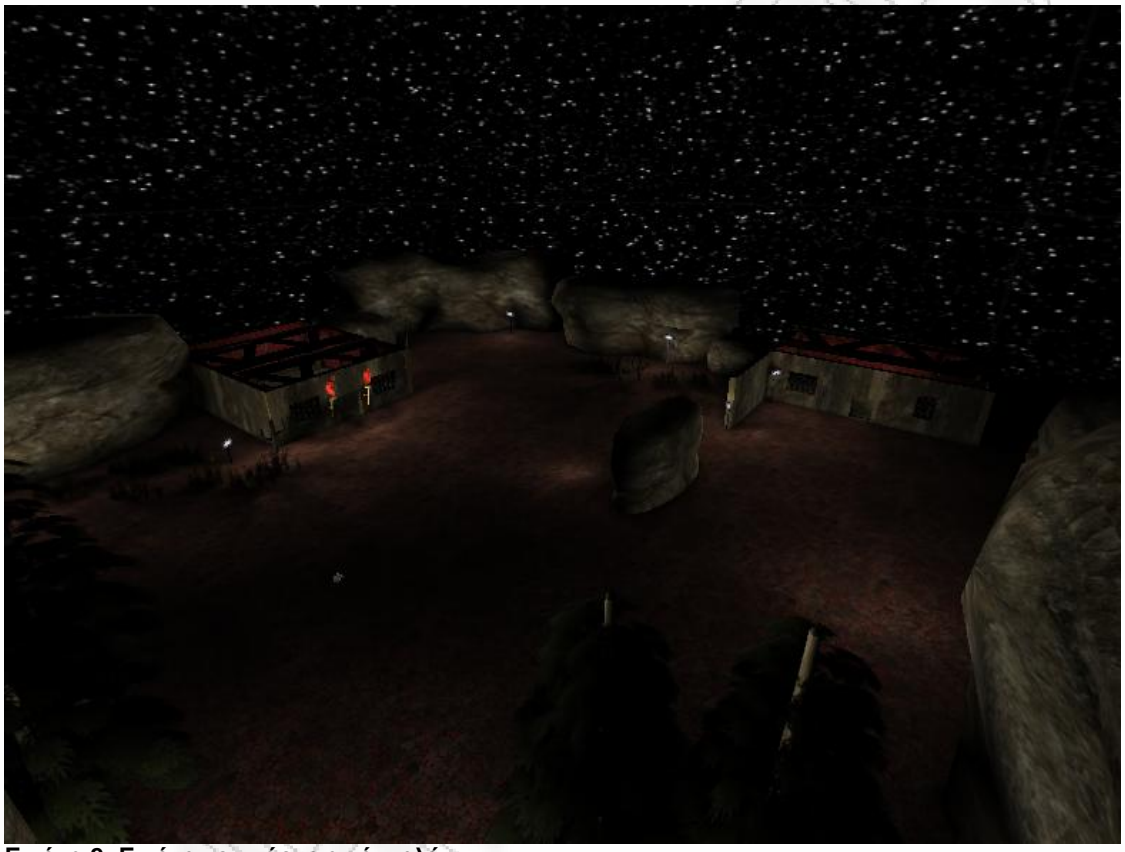
7.Παραγωγή συμπεριφοράς πρακτόρων σε ρόλους αστυνόμου και κλέφτη.

Παρακάτω ακολουθεί ένα πείραμα παραγωγής συμπεριφοράς πρακτόρων με τη χρήση του POSH ως μηχανισμό επιλογής ενέργειας. Οι πράκτορες χωρίζονται σε δύο κατηγορίες: τον αστυνόμο και τον κλέφτη, δηλαδή σε δύο διαφορετικούς ρόλους. Ο ρόλος του αστυνομικού είναι να προστατεύσει το θησαυρό πυροβολώντας και σκοτώνοντας όποιον πάει να τον κλέψει. Από την άλλη πλευρά, ο κλέφτης θέλει να αποκτήσει τον θησαυρό προσπαθώντας να μη σκοτωθεί. Το πείραμα βασίζεται σε αυτούς τους δύο ρόλους και παρουσιάζει τα αποτελέσματα του συνδυασμού διαφορετικών συμπεριφορών για καθένα από τους ρόλους αυτούς. Για παράδειγμα: Παρουσιάζονται τα αποτελέσματα μιας συμπεριφοράς ενός «θαρραλέου» κλέφτη που δεν δίνει μεγάλη σημασία στη σωματική του ακεραιότητα, αλλά στην ταχύτητά του για να πιάσει το θησαυρό, με αντίπαλο ένα αστυνομικό ο οποίος δίνει μεγαλύτερη σημασία στην προστασία του θησαυρού παρά στο να κυνηγήσει τον κλέφτη. Όμως τι θα συνέβαινε αν αυτός ο θαρραλέος κλέφτης έχει να αντιμετωπίσει έναν πιο «τολμηρό» αστυνομικό που κυνηγάει τον κλέφτη αφήνοντας όμως απροστάτευτο το θησαυρό ;

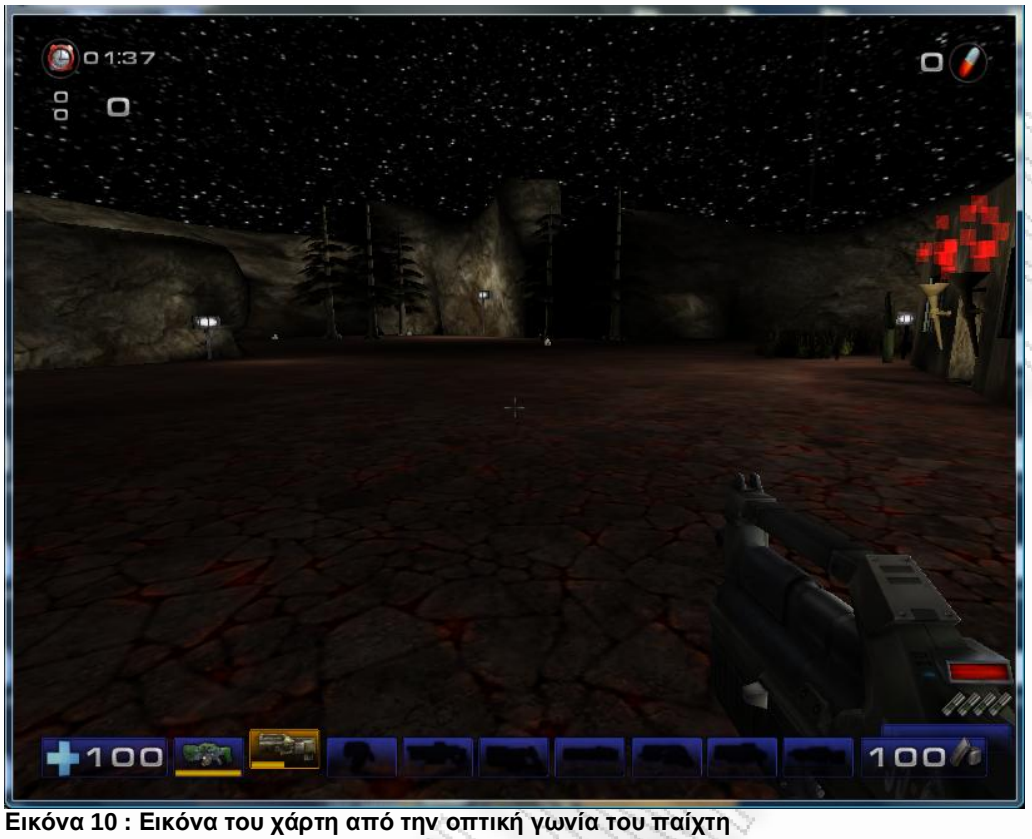
Έτσι λοιπόν για το πείραμα αυτό δημιουργήθηκαν δύο διαφορετικοί τύποι συμπεριφοράς για κάθε έναν από τους δύο ρόλους. Παρακάτω θα αναλυθεί η διαδικασία που ακολουθήθηκε για τη δημιουργία της κάθε συμπεριφοράς.

7.1.Υλοποίηση Κόσμου

Αρχικά για την υλοποίηση του πειράματος απαραίτητη είναι πρώτα η δημιουργία ενός κόσμου όπου το πείραμα θα λαμβάνει χώρα. Έτσι με τη χρήση του Unreal Editor δημιουργήθηκε μια απλή πίστα (χάρτης) με δύο σημεία εκκίνησης, ένα για τον κλέφτη και ένα για τον αστυνομικό. Στην πίστα υπάρχουν διάσπαρτα σφαίρες με τις οποίες μπορεί να εφοδιαστεί ο αστυνομικός και ένας θησαυρός σε μορφή mini health pack που αποτελεί και το στόχο του κλέφτη. Ο κόσμος του πειράματος είναι μια ερημική περιοχή μέσα σε βράχια. Ο θησαυρός που έχει τη μορφή ενός medkit (mini health pack) βρίσκεται σε ένα από τα δύο σπίτια που υπάρχουν στο χάρτη. Η ώρα που το πείραμα λαμβάνει χώρα είναι βράδυ και αυτό φαίνεται από τον έναστρο και ξάστερο ουρανό. Στις εικόνες παρακάτω φαίνεται η μορφή αυτού του χάρτη:



Εικόνα 9: Εικόνα του χάρτη από ψηλά.



Εικόνα 10 : Εικόνα του χάρτη από την οπτική γωνία του παίχτη

Εικόνα 11 : Εικόνα του χάρτη από την οπτική γωνία του παίχτη. Στο βάθος φαίνεται ο θησαυρός(medkit ή mini health pack)

7.2.Δημιουργία κανόνων παιχνιδιού

Στη συνέχεια δημιουργήθηκε ένα νέο είδος παιχνιδιού που κληρονομεί τους κανόνες του DeathMatch και προσθέτει άλλον έναν, την απόκτηση του θησαυρού. Το είδος του παιχνιδιού ονομάζεται Katerina's Pogamut και χρησιμοποιεί τα Gamebots για την επικοινωνία των πρακτόρων, με τον εικονικό κόσμο. Η δημιουργία του νέου αυτού mod (Katerina's Pogamut) έγινε με τη χρήση της Unreal Srcipt. Έτσι κατά την εκκίνηση του UT2004, εκτός από τα υπόλοιπα είδη παιχνιδιών, εμφανίζεται και το εξής:

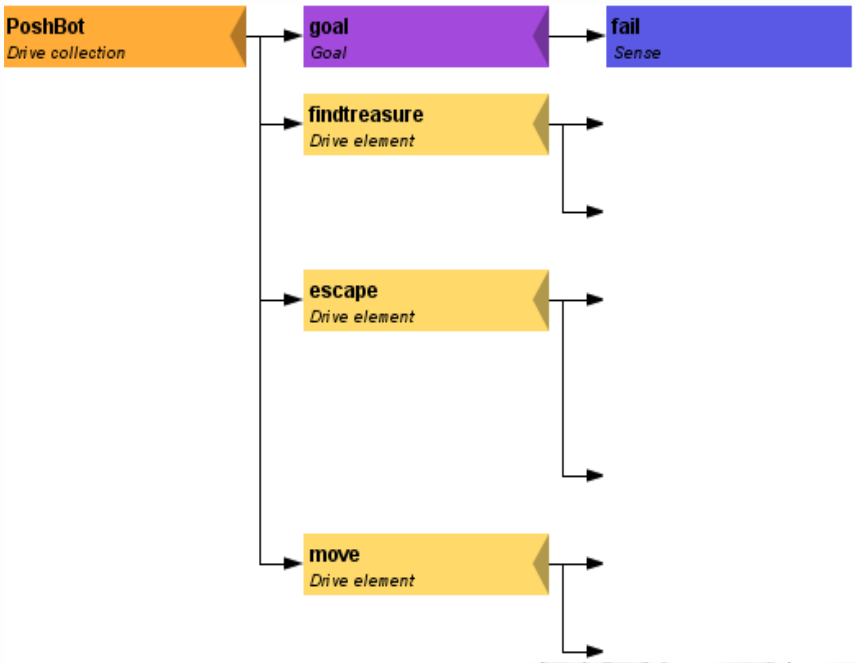
Εικόνα 12 : Εικόνα εκκίνησης Unreal Tournament 2004 για την επιλογή του είδους του παιχνιδιού.

Αναλυτικά οι οδηγίες που ακολουθήθηκαν για τη δημιουργία του νέου είδους παιχνιδιού ακολουθούν στο παράτημα ενότητα 12.

7.3.Ρόλος κλέφτη, 1η συμπεριφορά

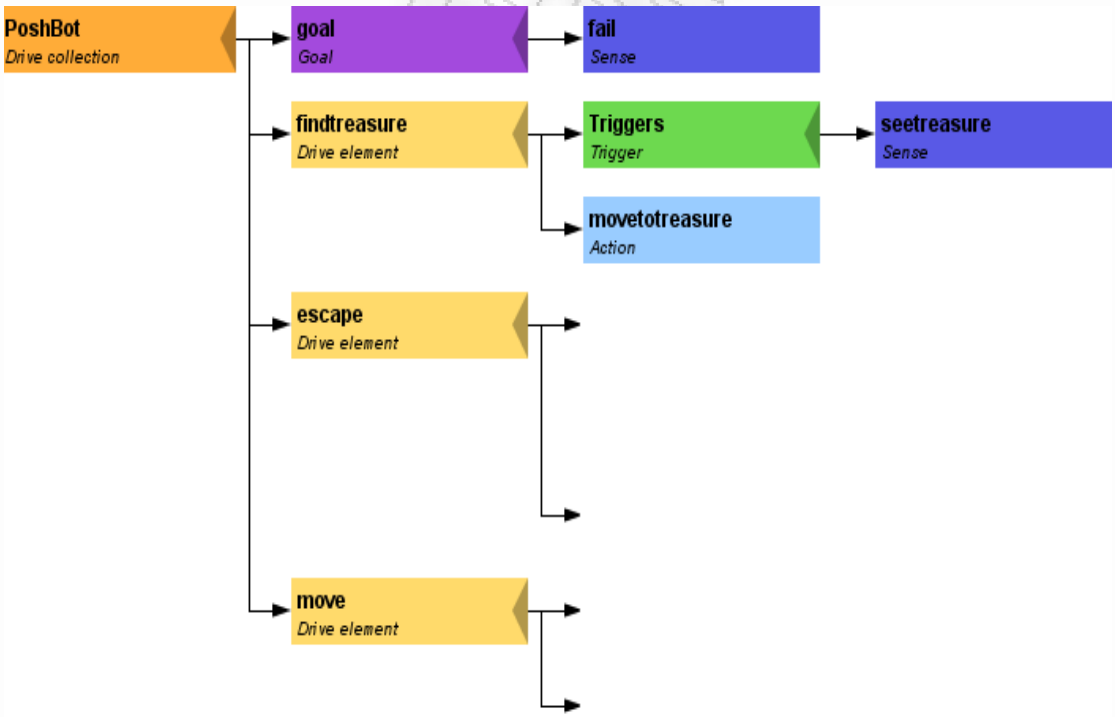
Όσον αφορά το ρόλο του κλέφτη, όπως αναφέραμε και παραπάνω, στόχος του είναι να βρει και να πάρει το θησαυρό, ο οποίος βρίσκεται κάπου στον εικονικό κόσμο, χωρίς βέβαια να σκοτωθεί. Η πρώτη συμπεριφορά όσον αφορά τον κλέφτη είναι η εξής:

Βασικός του στόχος είναι η κατάκτηση του θησαυρού. Δευτερεύων στόχος είναι η σωματική του ακεραιότητα και τελευταίος είναι η πλοήγησή του στο χώρο, ώστε να βρει το θησαυρό. Άρα ξεκινάμε με ένα drive collection, αφού ο πράκτοράς μας έχει διάρκεια ζωής. Έτσι ο στόχος του drive collection δεν ικανοποιείται ποτέ γι' αυτό και η αίσθηση που θα τον ικανοποιήσει έχει την τιμή fail (δηλαδή επιστρέφει πάντα την τιμή false), όπως φαίνεται και παρακάτω. Τα Drive elements είναι οι τρεις στόχοι που περιγράψαμε πριν, με πρώτο σε προτεραιότητα το βασικό του στόχο. Άρα έχουμε ένα σχήμα σαν και το παρακάτω (εικόνα 13):



Εικόνα 13: Διαδικασία δημιουργίας πλάνου για την πρώτη συμπεριφορά του κλέφτη.Βήμα1.

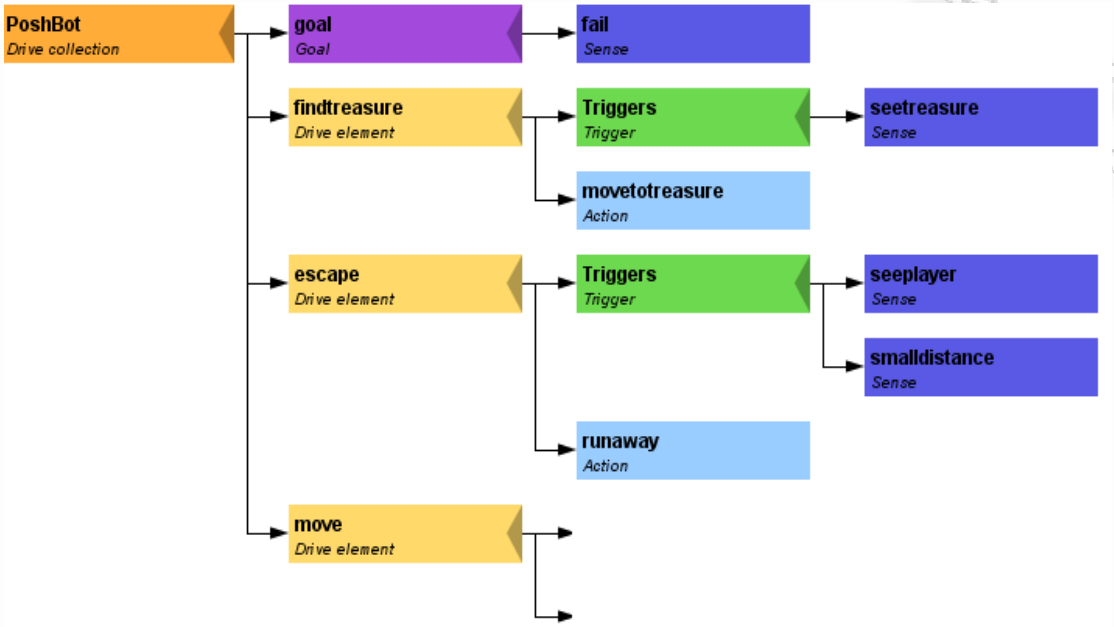
Για το πρώτο Drive element (findtreasure) ορίζουμε το πότε θα εκτελεστεί (δηλαδή ένα sense) και το τι θα κάνει (εδώ μπορεί να είναι είτε ένα απλό action, είτε ένα action pattern είτε ένα competence). Στην περίπτωσή μας έχουμε ορίσει ένα απλό action το movetomedkit και ένα sense το seemedkit. Έτσι το πλάνο γίνεται ως εξής:



Εικόνα 14: Διαδικασία δημιουργίας πλάνου για την πρώτη συμπεριφορά του κλέφτη.Βήμα2.

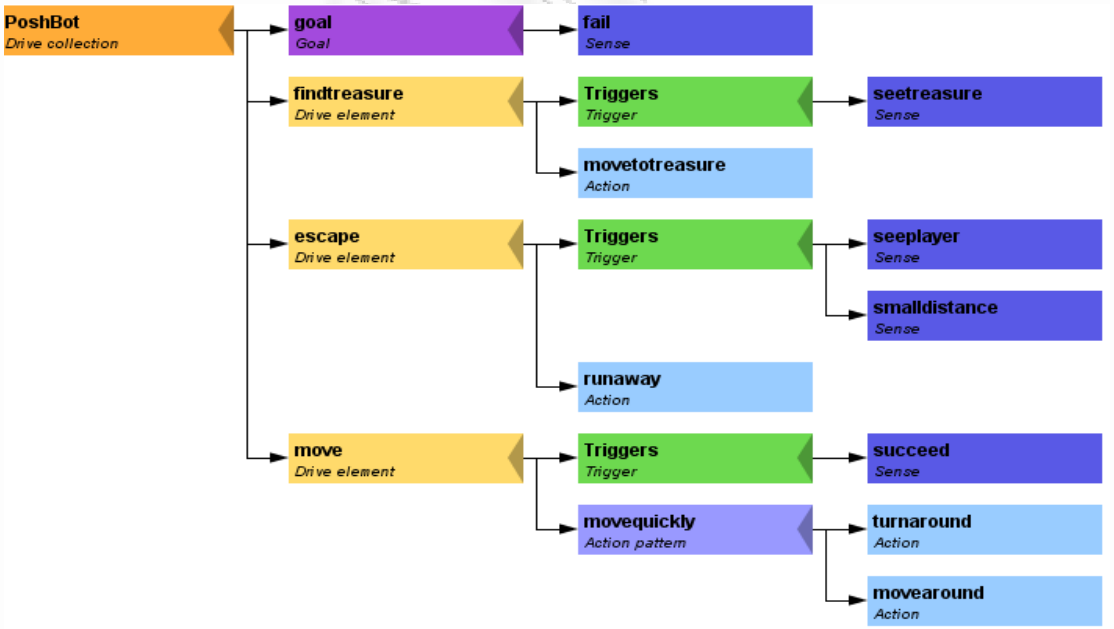
Για το δεύτερο σε προτεραιότητα στόχο (escape) ορίσαμε ότι εκτελείται, όταν βρεθεί κάποιος παίχτης στο οπτικό πεδίο του πράκτορα (seeplayer) και όταν η απόσταση αυτού και

του αντιπάλου είναι μικρή (smalldistance). Όταν συμβούν αυτά, ο πράκτορας θα προσπαθήσει να ξεφύγει εκτελώντας ένα απλό action, το runaway. Έτσι το πλάνο γίνεται ως εξής:



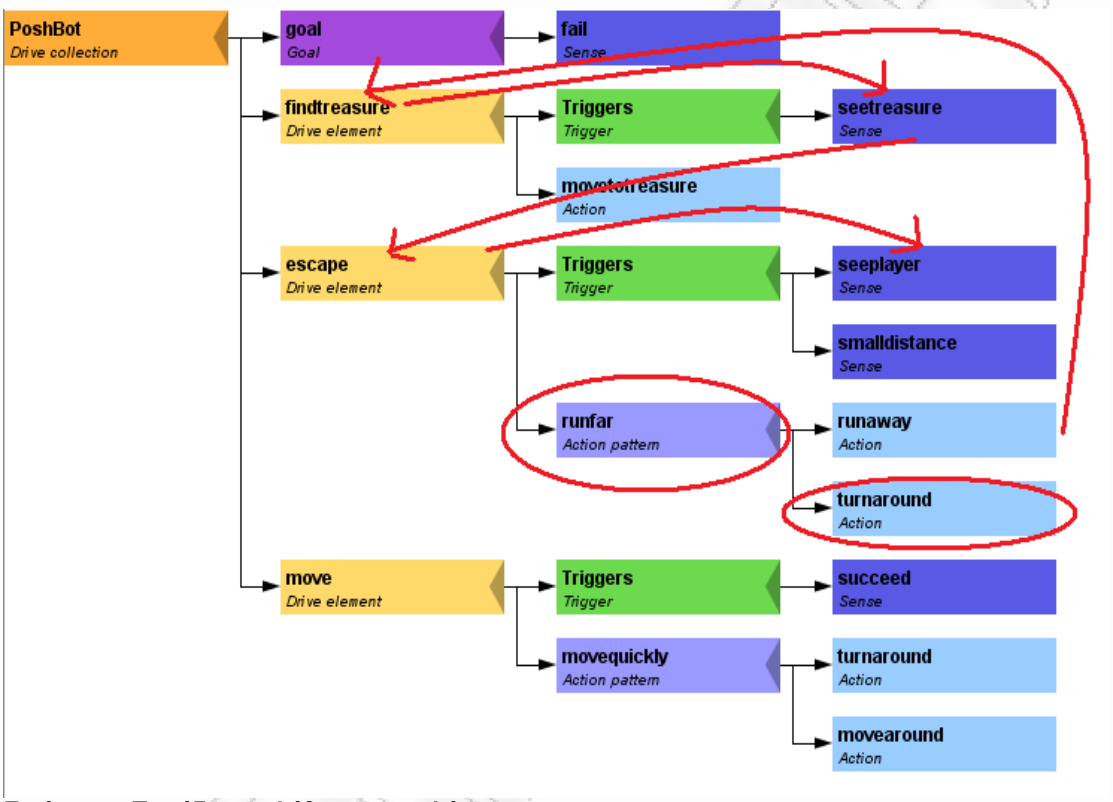
Εικόνα 15: Διαδικασία δημιουργίας πλάνου για την πρώτη συμπεριφορά του κλέφτη.Βήμα3.

Τέλος υπάρχει ένας στόχος (drive element) ο οποίος εκτελείται πάντα, εάν κανένας άλλος δεν μπορεί να εκτελεστεί (δεν ικανοποιούνται οι προϋποθέσεις τους). Αυτός ο στόχος είναι ο move και επειδή εκτελείται χωρίς προϋποθέσεις, η αίσθηση έχει την τιμή succeed (δηλαδή επιστρέφει πάντα την τιμή true). Όταν ενεργοποιηθεί αυτό το drive element, καλεί σαν ενέργεια ένα action pattern (movequicky) που εκτελεί διαδοχικά τις ενέργειες turnaround και movearound. Έτσι το τελικό πλάνο γίνεται ως εξής:



Εικόνα 16 : Διαδικασία δημιουργίας πλάνου για την πρώτη συμπεριφορά του κλέφτη.Βήμα4.

Γενικά όσο πιο απλό είναι το πλάνο,, τόσο πιο εύκολα και πιο σωστά εκτελείται η συμπεριφορά. Επίσης απαιτείται κατανόηση στο πως εκτελείται το πλάνο. Για παράδειγμα μια άλλη προσπάθεια για τη δημιουργία ενός πλάνου της ίδιας συμπεριφοράς ήταν η εξής:
Στο 2ο drive element (escape) αρχικά αντί για το action runaway, είχε χρησιμοποιηθεί ένα Action Pattern με πρώτη ενέργεια το runaway και δεύτερη το turntoplayer το οποίο έκανε τον πράκτορα να γυρίσει και να κοιτάξει τον άλλο παίχτη(σχήμα παρακάτω). Όμως όταν εκτελείται το runaway που κάνει τον πράκτορα να τρέχει μακριά από τον αντίπαλο, το πλάνο θα ξανακοιτάξει αν εκτελείται ο βασικός του στόχος (findtreasure) και στη συνέχεια ο δεύτερος(escape). Αν ο πρώτος δεν μπορεί να ενεργοποιηθεί, η λογική λέει ότι ούτε ο δεύτερος θα ενεργοποιηθεί, αφού με την εκτέλεση του runaway ο παίχτης δεν θα βλέπει πια τον παίχτη(seeplayer). Άρα η turnaround δεν θα πραγματοποιηθεί ποτέ.



Εικόνα 17: Παράδειγμα λάθους στο πλάνο.

Όταν ολοκληρωθεί το πλάνο, αυτό που χρειάζεται είναι να υλοποιηθούν μόνο τα senses και τα actions. Για τη δήλωση της μεθόδου του action πρέπει η μέθοδος να είναι :

- Public
- Χωρίς παραμέτρους
- Πρέπει να είναι σχολιασμένη με @SPOSHAction
- Και πρέπει να επιστρέφει τιμή (οποιαδήποτε εκτός από void)

Η επιστρεφόμενη τιμή δηλώνει πότε η ενέργεια είναι η επιτυχής ή όχι. Εάν η τιμή αυτή είναι false η μηχανή καταλαβαίνει ότι η ενέργεια αυτή απέτυχε.
Η αίσθηση είναι και αυτή μια μέθοδος public, χωρίς παραμέτρους, σχολιασμένη με @SPOSHSense και επιστρέφει τιμή. Η επιστρεφόμενη τιμή δηλώνει αν η αίσθηση ήταν επιτυχής ή όχι. Για παράδειγμα:

```
@SPOSHSense
  public boolean seetreasure() {

     if( items.getVisibleItems(ItemType.MINI_HEALTH_PACK).size()>0)
     {
            return true;  // Βλέπω το θησαυρό
     }

    return false;  // Δεν βλέπω θησαυρό
  }
```

Type= MINI_HEALTH_PACK αφού αυτό θεωρούμε σαν θησαυρό

Η ενέργεια movetotreasure υλοποιείται ως εξής:

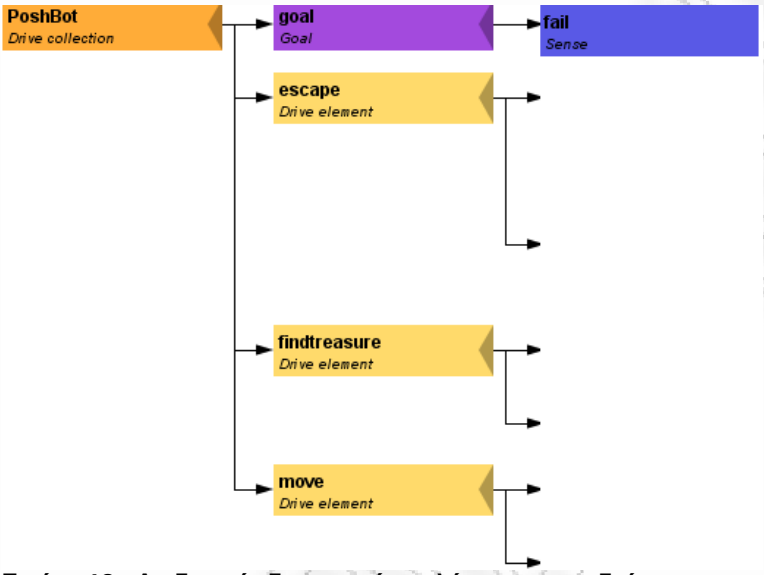
```
@SPOSHAction

  public boolean movetotreasure() {

    //σταματά την εκτέλεση οποιουδήποτε pathexecution αν γίνεται. *

    pathExecutor.stop();

    //βρίσκει όλους τους θησαυρούς τους οποίους έχουμε και θεωρήσει σαν mini health packs.

    Item[] treasures = items.getVisibleItems(ItemType.MINI_HEALTH_PACK).values().toArray(new Item[]{});

    if (treasures.length == 0) {

       user.severe("Δεν υπάρχει κανένας θησαυρός στο χάρτη");

       return false;

       }

    //Κινήσου γρήγορα

    move.setRun();

    //Βρες το μονοπάτι από τον εαυτό σου έως το θησαυρό.. Ξέρουμε ότι είναι ένας γιαυτό και treasures[0]

    pathExecutor.followPath(pathPlanner.computePath(bot, treasures[0].getLocation()));

    return true;

  }
```

* Μερικά actions δρουν συνέχεια γι' αυτό και πρέπει ο ίδιος ο χρήστης να τα τερματίζει έτσι ώστε να πραγματοποιηθούν άλλα. Για παράδειγμα αν δεν τερματιστεί το action shoot ο

πράκτορας θα συνεχίζει να πυροβολεί μέχρι να του δηλωθεί ότι πρέπει να σταματήσει. Το ίδιο συμβαίνει και με την pathexecutor. Αν δεν τερματιστεί, ο πράκτορας θα συνεχίζει να κινείται και κανένα άλλο action δε θα μπορεί να πραγματοποιηθεί κάτι που θα προκαλέσει πρόβλημα στη σωστή εκτέλεση του πλάνου.
Τα υπόλοιπα actions και senses του παραπάνω πλάνου καθώς και το που υλοποιούνται βρίσκονται στο παράρτημα 10.

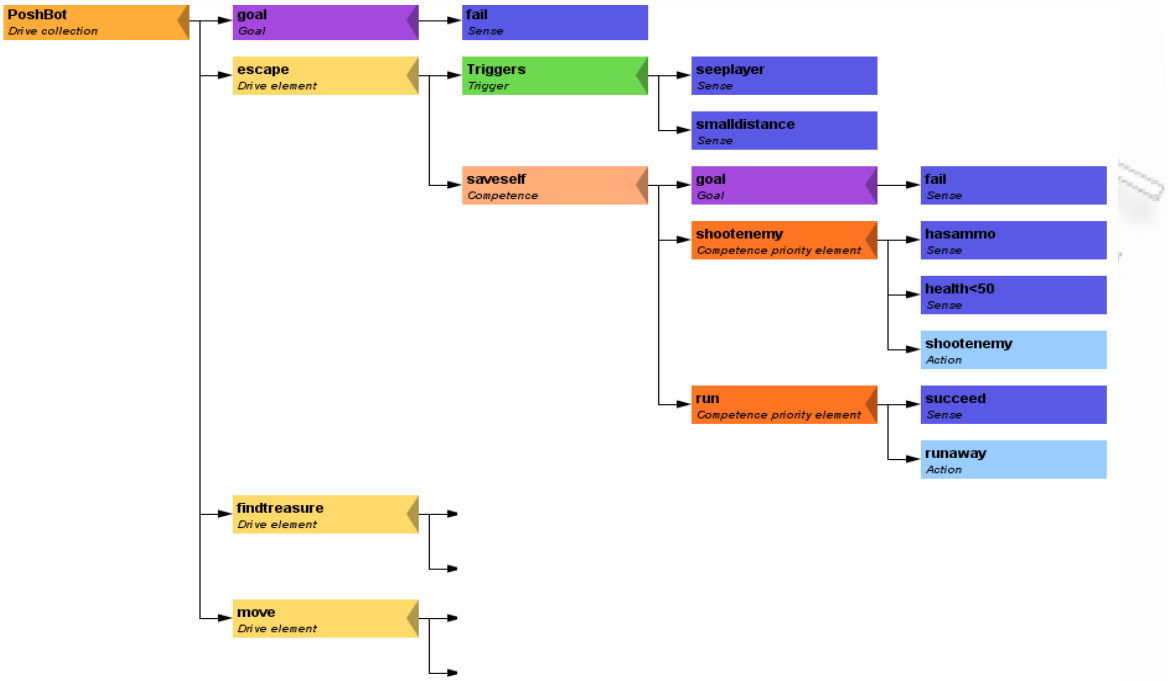
7.4.Ρόλος κλέφτη, 2η συμπεριφορά

Όπως αναφέραμε και προηγουμένως, στο πείραμα δοκιμάζονται δύο διαφορετικές συμπεριφορές για τον κάθε ρόλο. Έτσι δοκιμάζεται μια δεύτερη συμπεριφορά για τον κλέφτη που τον κάνει πιο προσεκτικό όσον αφορά τη σωματική του ακεραιότητα. Πιο προσεκτικός στη γλώσσα του POSH σημαίνει ότι στόχος με τη μεγαλύτερη προτεραιότητα είναι να είναι ασφαλής ο κλέφτης. Επίσης προσθέτουμε και την ικανότητα στον κλέφτη να πυροβολήσει, αν νιώθει ότι απειλείται η σωματική του ακεραιότητα. Άρα το προηγούμενο πλάνο τροποποιείται ως εξής:



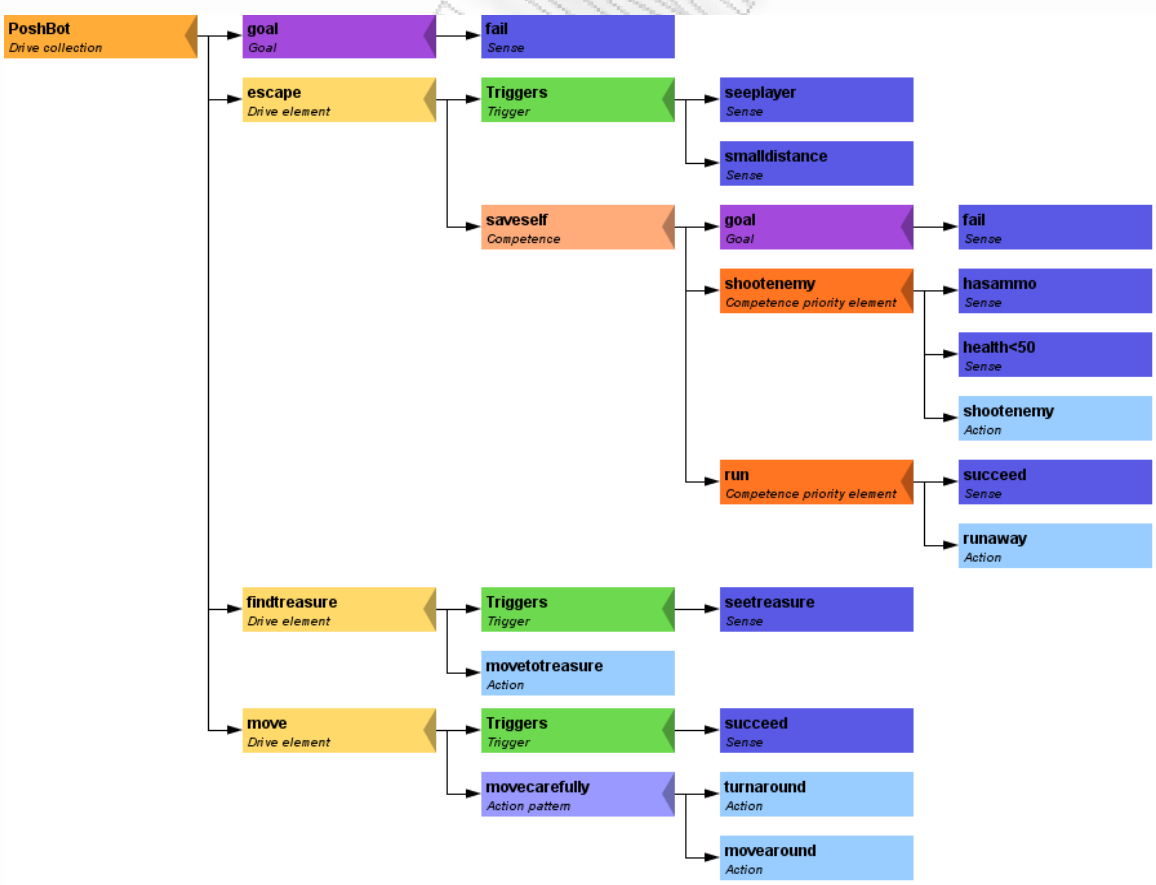
Εικόνα 18 : Διαδικασία δημιουργίας πλάνου για την δεύτερη συμπεριφορά του κλέφτη.Βήμα1.

Το πρώτο drive element (escape) έχει αλλάξει, αφού τώρα προστέθηκε η ικανότητα στον κλέφτη να μπορεί να πυροβολήσει, όταν απειλείται η σωματική του ακεραιότητα. Το escape ενεργοποιείται, όταν ο πράκτορας δει έναν άλλο, δηλαδή όταν η αίσθηση seeplayer επιστρέψει την τιμή true και όταν η απόσταση από αυτόν είναι μικρή (smalldistance). Τώρα η ενέργεια που θα εκτελέσει δεν θα είναι ένα απλό action αλλά ένα competence (saveself) που πάντα θα μπορεί να ενεργοποιηθεί αφού ο στόχος του είναι fail. Το πρώτο competence element (shootenemy) που ελέγχεται είναι, αν ο κλέφτης δεν έχει χτυπηθεί δηλαδή αν η ενέργειά του είναι ίση με την αρχική του (έστω ότι η αρχική του ενέργεια είναι 50), δηλαδή health<50 και αν έχει πυρομαχικά, δηλαδή hasammo. Αν η ενέργειά του είναι μικρότερη της αρχικής και έχει πυρομαχικά τότε θα πυροβολήσει τον αντίπαλο, δηλαδή θα εκτελέσει το συγκεκριμένο competence element που είναι ένα απλό action, το shootenemy, αλλιώς θα ενεργοποιηθεί το δεύτερο competence element (run) που είναι ένα action το runaway. Δηλαδή το πλάνο έχει ως εξής:



Εικόνα 19 : Διαδικασία δημιουργίας πλάνου για την δεύτερη συμπεριφορά του κλέφτη.Βήμα2.

Οι άλλο δύο στόχοι είναι ίδιοι με το προηγούμενο, δηλαδή το ολοκληρωμένο πλάνο είναι ως εξής:



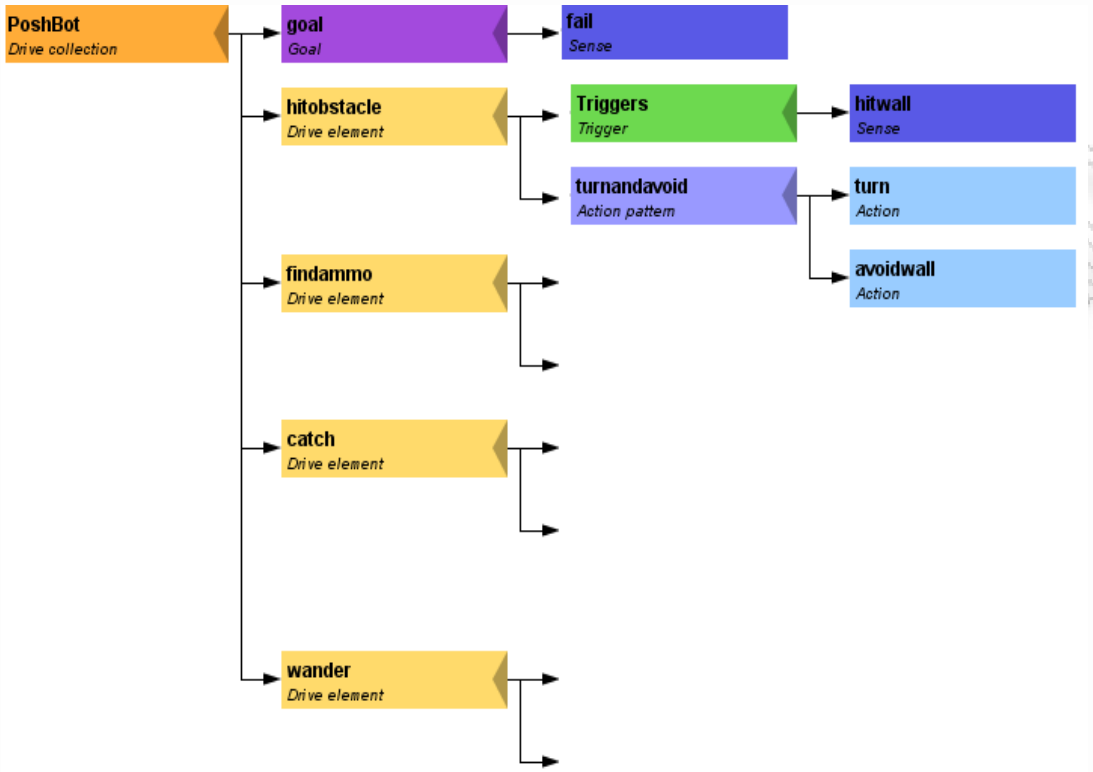
Εικόνα 20 : Διαδικασία δημιουργίας πλάνου για την δεύτερη συμπεριφορά του κλέφτη.Βήμα3.

7.5.Ρόλος αστυνόμου, 1η συμπεριφορά

Ο ρόλος του αστυνομικού είναι να προστατέψει το θησαυρό που βρίσκεται στο χάρτη. Γι' αυτό το λόγο ο αστυνόμος περιπολεί το χώρο γύρω από το θησαυρό. Η πρώτη συμπεριφορά που υλοποιήθηκε έχει τέσσερις βασικούς στόχους. Αρχικός στόχος είναι ο έλεγχος για το αν υπάρχει κάποιο εμπόδιο στο δρόμο του αστυνόμου με αποτέλεσμα να εμποδίζεται η περιπολία του (hitobstacle). Έπειτα ελέγχει, αν χρειάζεται να ψάξει για πυρομαχικά (findammo) και ύστερα αν υπάρχει κάποιος ύποπτος στο χώρο για να κυνηγήσει (catch). Τελευταίος στόχος του είναι η περιπολία του (wander). Και σε αυτή τη συμπεριφορά ξεκινάμε τη σχεδίαση του πλάνου με ένα drive collection του οποίου τα drive elements είναι οι στόχοι που περιγράψαμε παραπάνω. Το πλάνο του έχει ως εξής :

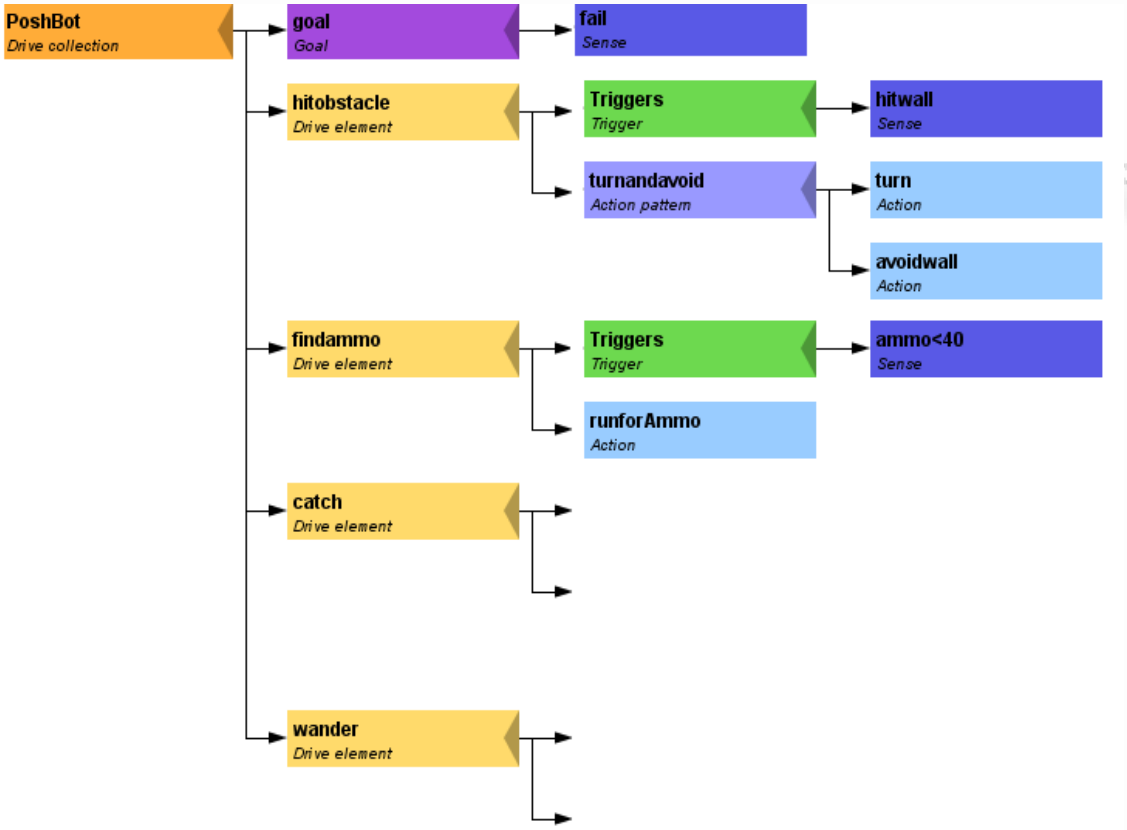
Εικόνα 21: Διαδικασία δημιουργίας πλάνου για την πρώτη συμπεριφορά του αστυνόμου.Βήμα1.

Ο πρώτος και βασικότερος στόχος της συγκεκριμένης συμπεριφοράς είναι ο έλεγχος για το αν κάποιο εμπόδιο (πχ ένας τοίχος) εμποδίζει την πλοήγηση του πράκτορα. Αυτό το drive element (hitobjactle) ικανοποιείται με την αίσθηση (hitwall) και ενεργοποιεί ένα action pattern (turnandavoid). Όπως καταλαβαίνουμε και από το όνομα του action pattern, οι ενέργειες που υλοποιούνται διαδοχικά είναι η turn, η οποία περιστρέφει τον πράκτορα κατά 180^{o}, και η avoidwall σύμφωνα με την οποία ο πράκτορας διαλέγει ένα κοντινό και ορατό σημείο στο χάρτη (navpoint) για να πάει. Έτσι στο πλάνο προστίθενται τα εξής:



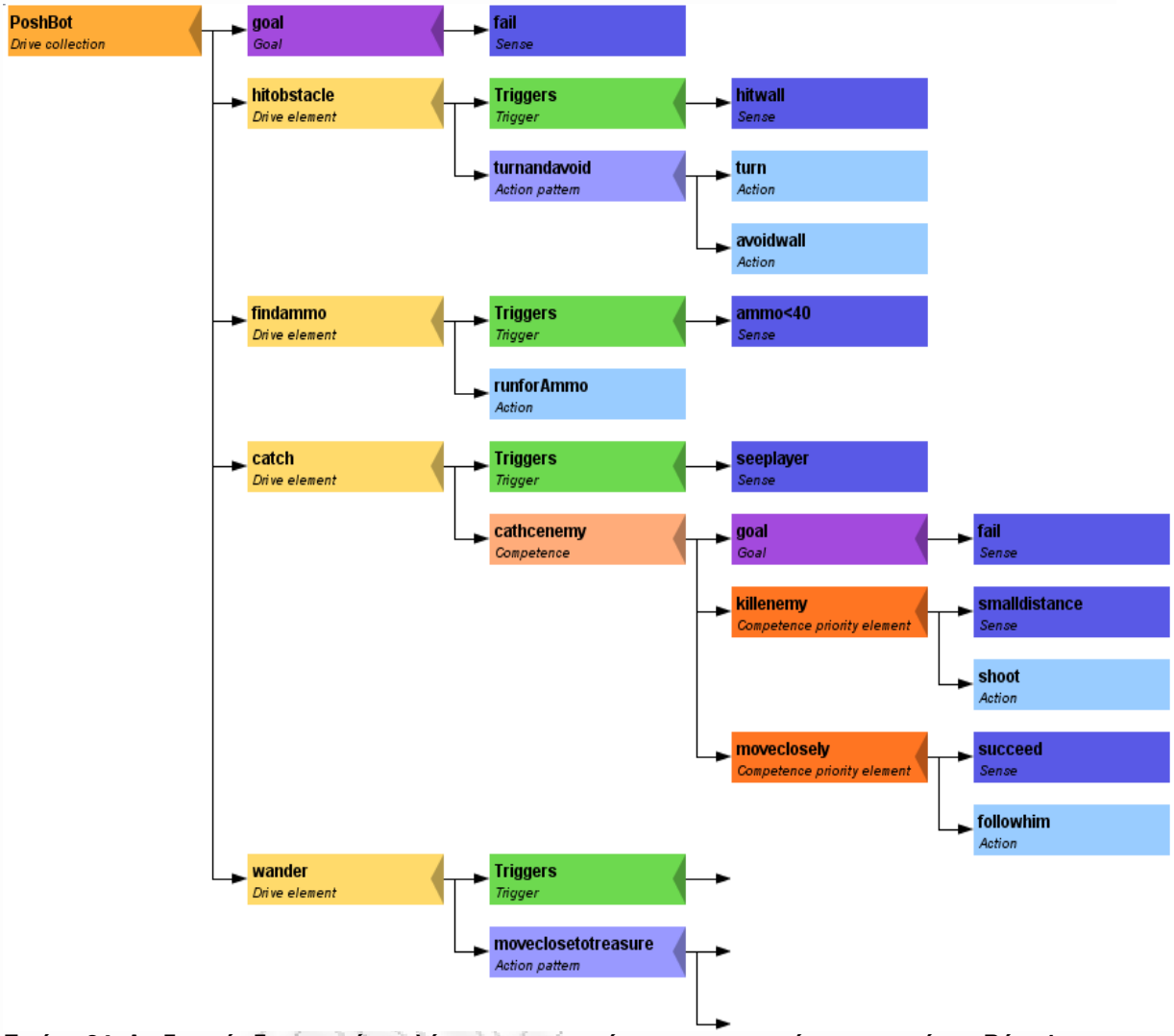
Εικόνα 22: Διαδικασία δημιουργίας πλάνου για την πρώτη συμπεριφορά του αστυνόμου.Βήμα2.

Ο δεύτερος σε προτεραιότητα στόχος του πράκτορα είναι ο έλεγχος για το αν έχει διαθέσιμα πυρομαχικά. Το drive element (findammo) ενεργοποιείται, αν η ποσότητα των πυρομαχικών του είναι μικρότερη του 40 (ammo<40). Στην περίπτωση αυτή το sense δεν επιστρέφει true ή false, αλλά έναν ακέραιο αριθμό και η σύγκριση με το 40 γίνεται από το ίδιο το Posh plan. Το drive element αυτό ενεργοποιεί ένα action, το runforAmmo, κάνοντας τον πράκτορα να ψάχνει για πυρομαχικά που βρίσκονται διάσπαρτα στην πίστα-χάρτη. Έτσι το πλάνο συμπληρώνεται ως εξής:



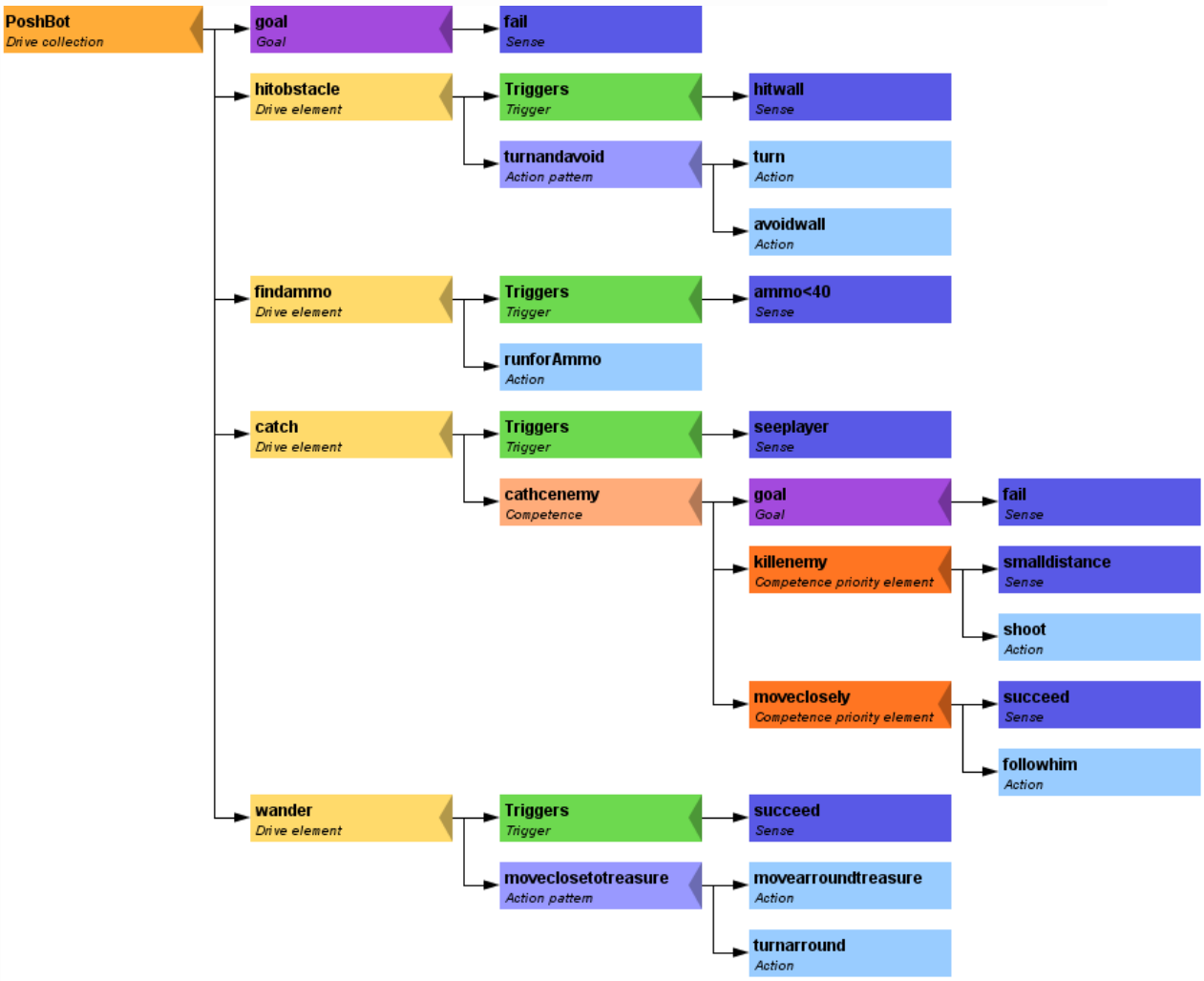
Εικόνα 23: Διαδικασία δημιουργίας πλάνου για την πρώτη συμπεριφορά του αστυνόμου.Βήμα3.

Ο τρίτος σε προτεραιότητα στόχος είναι το να πιάσει τον κλέφτη (catch). Αυτό το Drive element ενεργοποιείται, όταν βρεθεί κάποιος στο οπτικό πεδίο του πράκτορα (seeplayer). Αν αυτό συμβεί τότε ενεργοποιείται ένα competence (catchenemy) σύμφωνα με το οποίο αν ο «εχθρός» βρίσκεται σε κοντινή απόσταση (smalldistance), τότε ενεργοποιείται το πρώτο competence element (killenemy) που οδηγεί σε ένα action (shoot) και κάνει τον πράκτορα να πυροβολεί τον αντίπαλο. Αν η απόσταση δεν είναι κοντινή, τότε ενεργοποιείται το επόμενο competence element (moveclosely) που οδηγεί σε ένα άλλο action (followhim) το οποίο κάνει τον πράκτορα να κινηθεί προς τη θέση του αντιπάλου. Έτσι το πλάνο συμπληρώνεται ως εξής:



Εικόνα 24: Διαδικασία δημιουργίας πλάνου για την πρώτη συμπεριφορά του αστυνόμου.Βήμα4.

Τελευταίος στόχος του πράκτορα είναι η πλοήγησή του (wander), ο οποίος ενεργοποιείται πάντα, εφόσον κανένας άλλος στόχος δεν μπορεί να πραγματοποιηθεί. Το τελευταίο λοιπόν drive element ενεργοποιεί ένα action pattern (moveclosetotreasure) το οποίο με τη σειρά του ενεργοποιεί δύο διαδοχικές ενέργειες (actions), το movearoundtreasure και το turnaround. Το πρώτο αφορά την περιπολία του αστυνόμου γύρω από τον θησαυρό και το δεύτερο τον περιστρέφει κατά 180 μοίρες κάθε φορά, έτσι ώστε να παρακολουθεί τι γίνεται γύρω του. Έτσι το ολοκληρωμένο πλάνο έχει την εξής μορφή :

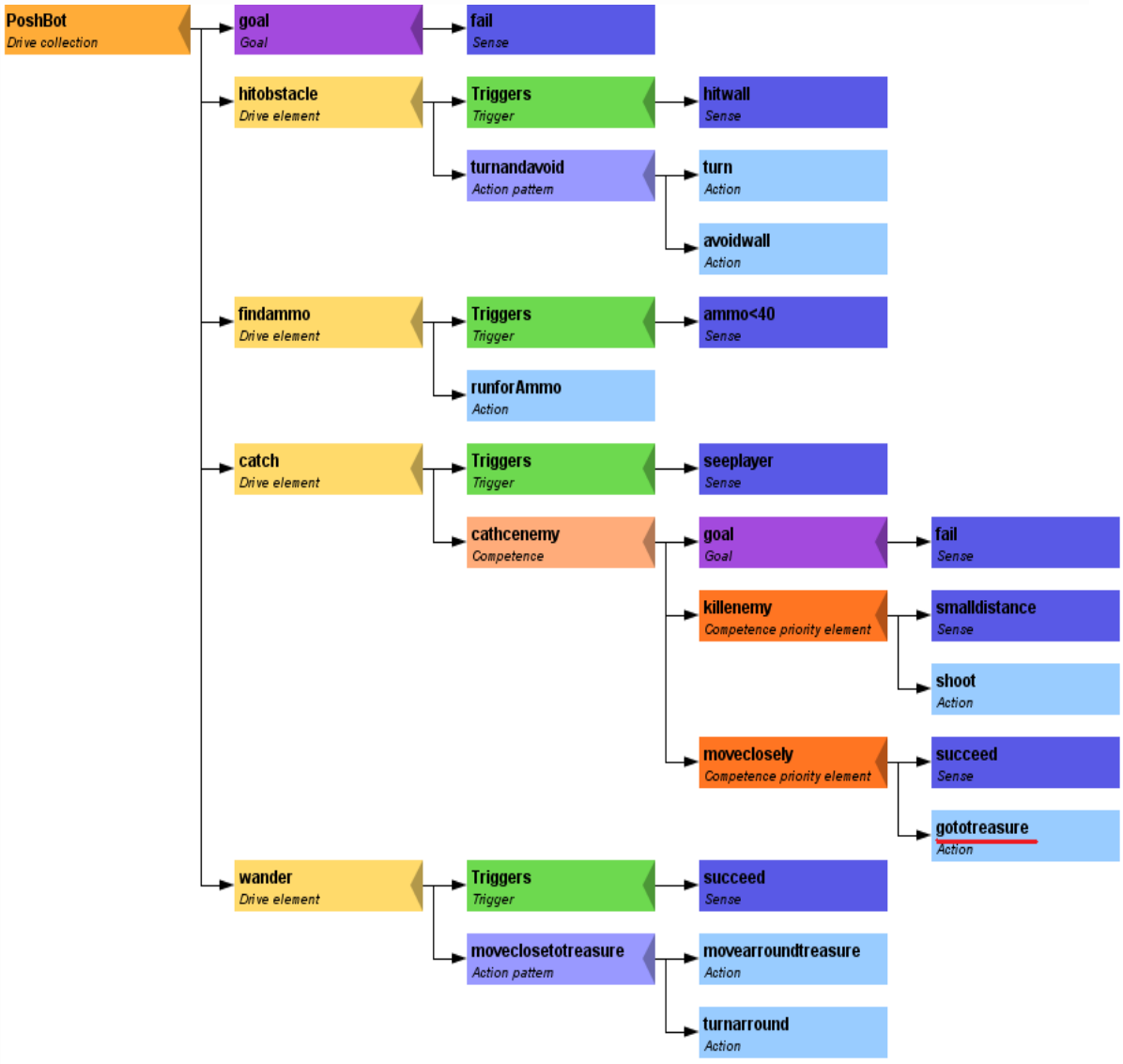


Εικόνα 25 : Διαδικασία δημιουργίας πλάνου για την πρώτη συμπεριφορά του αστυνόμου.Βήμα5.

Η συμπεριφορά που υλοποιήθηκε δημιουργεί έναν «θαρραλέο» αστυνομικό ο οποίος θα κυνηγήσει τον κλέφτη, αφήνοντας όμως με αυτό τον τρόπο απροστάτευτο το θησαυρό. Τι γίνεται όμως στην περίπτωση που ο αστυνομικός έχει συμπεριφορά φύλακα και όχι κυνηγού;

7.6.Ρόλος αστυνόμου, 2η συμπεριφορά.

Σε αυτή τη συμπεριφορά το μοναδικό πράγμα που αλλάχτηκε είναι το action followhim στο competence cathenemy. Ο αστυνόμος τώρα, όταν δει τον κλέφτη σε μακρινή απόσταση δεν τον κυνηγάει, αλλά επιλέγει ένα σημείο που βρίσκεται κοντά στο θησαυρό για να τον προστατεύσει. Το action αυτό είναι το gototreasure.



Εικόνα 26: Πλάνο της δεύτερης συμπεριφοράς του αστυνόμου

Η δημιουργία της συμπεριφοράς ενός πράκτορα είναι μια διαδικασία ελεύθερη ανάλογα με τις προτιμήσεις του καθενός. Μπορεί για παράδειγμα κάποιος να αλλάξει την προτεραιότητα του στόχου findammo ή να την αφαιρέσει τελείως ή να δημιουργήσει άλλους στόχους. Μια άλλη πιθανή συμπεριφορά για τον αστυνόμο είναι να φωνάζει ενισχύσεις, όταν δει ότι απειλείται, ή να αλλάζει το όπλο του με ένα καλύτερο. Την ευχέρεια αυτή, την παρέχει η πλατφόρμα του Pogamut, αφού μπορούμε εύκολα να φτιάχνουμε πλάνα με τη χρήση σχημάτων, χωρίς να χρειάζεται να τα προγραμματίσουμε. Τον κώδικα τον δημιουργεί από μόνο του. Για παράδειγμα ο κώδικας για το παραπάνω πλάνο είναι ο εξής:

```
(

        (C catch (goal ((fail)))
                (elements
                        ( (speak (trigger ((smalldistance))) speak))
                        ( (followhim (trigger ((succeed))) followhim))
                )
        )
        (C cathcenemy (goal ((fail)))
                (elements
                        ( (killenemy (trigger ((smalldistance))) shoot))
                        ( (moveclosely (trigger ((succeed))) gototreasure))
                )
        )
        (AP shoothim  (turntoplayer))

        (AP turnandavoid  (turn avoidwall))

        (AP moveclosetotreasure  (turnarround movearroundtreasure))

        (AP follow  (turntoplayer))

        (SDC PoshBot (goal ((fail)))
                (drives
                        ((hitobstacle (trigger ((hitwall))) turnandavoid ) )
                        ((findammo (trigger ((ammo 40 <))) runforAmmo ) )
                        ((protecttreasure (trigger ((seeplayer))) cathcenemy ) )
                        ((wander (trigger ((succeed))) moveclosetotreasure ) )
                )
        )
)
```

Κώδικας πλάνου POSH (.lap)

8. Συμπεράσματα πειράματος

Εκτελώντας το πείραμα αρκετές φορές συνδυάζοντας διαφορετικές συμπεριφορές κάθε φορά, καταλήξαμε στα εξής αποτελέσματα:

Οι πιθανότητες να είναι κάποιος νικητής είναι περίπου οι ίδιες. Όσον αφορά τον αστυνομικό που ακολουθεί τον κλέφτη αρκετές είναι οι φορές που κερδίζει πυροβολώντας τον κλέφτη, όμως αρκετές είναι και οι φορές που δεν μπορεί να προφυλάξει το θησαυρό επειδή απομακρύνεται από αυτόν. Έτσι, καθώς ακολουθεί τον κλέφτη, υπάρχουν πιθανότητες να χάσει τα ίχνη του κατά τη διάρκεια του κυνηγητού και ο κλέφτης να προλάβει να βρει και να πάρει το θησαυρό. Αυτό γίνεται κυρίως στην περίπτωση όπου αντίπαλος είναι ο γενναίος ληστής, ο οποίος βασίζεται στην ταχύτητά του για να ξεφύγει χωρίς να ανταποδίδει τα πυρά στον αστυνόμο. Ο κλέφτης που δεν διστάζει να σηκώσει το όπλο του ($2^{η}$ συμπεριφορά για τον κλέφτη) σταματά να ψάχνει για το θησαυρό και αμύνεται μόλις αντιληφθεί ότι κινδυνεύει η σωματική του ακεραιότητα. Και σε αυτή την περίπτωση μπορεί ο κλέφτης να κερδίσει εφόσον προλάβει πρώτος να σκοτώσει τον αντίπαλο. Όσον αφορά τον αστυνομικό που φρουρεί τον θησαυρό αρκετές είναι οι φορές που χάνει επειδή τρέχοντας προς τον θησαυρό, όταν αντιληφθεί την παρουσία του κλέφτη, χάνει αρκετό χρόνο στο να τον σημαδέψει και να τον πυροβολήσει. Από την άλλη πλευρά υπάρχουν περιπτώσεις όπου ο αστυνομικός καταφέρνει να προστατέψει το θησαυρό προλαβαίνοντας να πυροβολήσει τον κλέφτη που κατευθύνεται προς τον θησαυρό. Αυτό συμβαίνει κυρίως με τον προστατευτικό κλέφτη ο οποίος θέλει να ανταποδώσει τα πυρά και δεν τρέχει αμέσως προς το θησαυρό. Αντίθετα, ο γενναίος κλέφτης προλαβαίνει πιο εύκολα να πιάσει το θησαυρό και να λήξει το παιχνίδι. Τα αποτελέσματα του

κάθε παιχνιδιού διέφεραν κάθε φορά και γι' αυτό δεν είναι εύκολο να διακρίνουμε ποια από τις δύο συμπεριφορές για τον κάθε ρόλο είναι η καλύτερη. Τα αποτελέσματα αυτά ίσως να είναι και διαφορετικά αν οι συμπεριφορές αυτές πραγματοποιηθούν σε άλλο πιο περίπλοκο χάρτη.

Γενικότερα με την παρούσα εργασία συμπεραίνουμε ότι η παραγωγή συμπεριφοράς ενός πράκτορα είναι ένα πολύ ενδιαφέρον ερευνητικό πεδίο με το οποίο εύκολα μπορεί κάποιος να ασχοληθεί και να πειραματιστεί. Ένα χρήσιμο «μέσο» για την παραγωγή της συμπεριφοράς του πράκτορα όσον αφορά την επιλογή κίνησης αποδείχθηκε ότι είναι το POSH. Με αυτό επιτυγχάνουμε μια αρκετά αληθοφανή συμπεριφορά του πράκτορα, διότι παρέχει συνοπτικά τα εξής πλεονεκτήματα :

- Υποστηρίζει τα basic reactive plans (BRP), που επιτρέπουν ένα ευέλικτο αλλά και συγκεντρωμένο στους στόχους μηχανισμό επιλογής ενέργειας.
- Υποστηρίζει τον ψευδοπαραλληλισμό και την εναλλαγή της προσοχής σε υψηλότερες προτεραιότητες, δηλαδή ο πράκτορας μπορεί και επανεξετάζει τους στόχους του ανάλογα με τις ανάγκες του και τις επιρροές από το περιβάλλον του.
- Περιορίζει την υπερβολική ανάπτυξη στοίβας, εφόσον επιτρέπει την ύπαρξη κύκλων στην ιεραρχία του. Το Slip-stack κάνει τον πράκτορα να ανταποκρίνεται γρηγορότερα στα ερεθίσματα που δέχεται από το περιβάλλον του αλλά και από την εσωτερική του κατάσταση, εφόσον του δίνει τη δυνατότητα να επισκέπτεται το ιστορικό των αποφάσεών του συχνά περιορίζοντας με αυτό τον τρόπο το μέγεθος της στοίβας. Σε κάθε κύκλο του μηχανισμού επιλογής ενέργειας POSH γίνεται έλεγχος των drives έτσι ώστε να ελεγχθεί, αν πρέπει να στραφεί η προσοχή του πράκτορα αλλού.
- Επανεκκινεί το πλάνο ιεραρχίας από την ρίζα του, αν αυτό τερματιστεί.

Η επιλογή της πλατφόρμας του Pogamut βασίστηκε περισσότερο στη δυνατότητα που παρέχει για εύκολη δημιουργία πλάνων POSH, εφόσον παρέχει την ευχέρεια δημιουργίας πλάνων με τη χρήση σχημάτων χωρίς να χρειάζεται ο προγραμματισμός τους. Ο προγραμματισμός των πλάνων γίνεται αυτόματα, ενώ δίνεται και η δυνατότητα παρέμβασης σε αυτόν. Έτσι το μόνο που χρειάζεται να κάνει ο χρήστης είναι να σχεδιάσει το πλάνο της συμπεριφοράς του πράκτορα και να προγραμματίσει τις πρωταρχικές ενέργειες και αισθήσεις. Βέβαια, η διαδικασία δημιουργίας ενός πλάνου χρειάζεται ιδιαίτερη προσοχή και υπομονή καθώς και συχνό έλεγχο. Όπως αναφέρθηκε και σε προηγούμενο κεφάλαιο, τα βήματα της δημιουργίας ενός σωστού πλάνου είναι η αρχική αποσύνθεση της συμπεριφοράς, η επαναληπτική ανάπτυξη και η αναθεώρηση των προδιαγραφών.

Τέλος, η επιλογή της μηχανής του Unreal και κυρίως του περιβάλλοντος του Unreal Tournament 2004 ως περιβάλλον για τους πράκτορες, έγινε με διάφορα κριτήρια. Αρχικά, ο βασικός λόγος που επιλέχτηκε το UT2004 και όχι κάποια από τις άλλες εκδόσεις του (όπως το Unreal Tournament 99 ή το UDK), είναι ότι το Pogamut και κυρίως η έκδοση 3 που χρησιμοποιείται, βασικά σχεδιάστηκε για να χρησιμοποιεί αυτό το περιβάλλον. Έπειτα, παρόλο που υπάρχουν και άλλα περιβάλλοντα όμοια με το Unreal Tournament, όπως το Quake, το Crysis κ.α., το Unreal Tournament είναι πιο εύκολα τροποποιήσιμο, εφόσον έχει ένα μέρος του κώδικά του ανοιχτό για τροποποιήσεις, αυτό που είναι γραμμένο σε Unreal Script. Έτσι εύκολα μπορεί κανείς να δημιουργήσει νέους χάρτες (με τη χρήση του Unreal editor), αλλά και να αλλάξει ή να δημιουργήσει καινούρια είδη παιχνιδιών με διαφορετικούς κανόνες (με τη χρήση της Unreal Script). Επίσης η ύπαρξη των Gamebots που ενσωματώνονται στο Unreal Tournament διευκολύνουν τη σύνδεση άλλων εξωτερικών προγραμμάτων, όπως το Pogamut που χρησιμοποιήθηκε στη συγκεκριμένη εργασία.

9.Ανοιχτά Ερευνητικά Πεδία

Η συγκεκριμένη εργασία είναι αρκετά επεκτάσιμη, όσον αφορά την παραγωγή συμπεριφοράς πρακτόρων. Μια αρκετά ενδιαφέρουσα προσέγγιση είναι η παραγωγή

συμπεριφορών πρακτόρων που συνεργάζονται, ανταγωνίζονται και αλληλεπιδρούν μεταξύ τους. Για παράδειγμα στην περίπτωση των δύο ρόλων της εργασίας, αστυνόμου και κλέφτη, ποιά θα ήταν τα αποτελέσματα της ύπαρξης παράλληλα δύο ληστών που συνεργάζονται μεταξύ τους; Για παράδειγμα: ο ένας να είναι το δόλωμα και ο άλλος να έχει σαν στόχο το θησαυρό. Αντίστοιχα τι αποτελέσματα θα είχαμε αν υπήρχαν δύο ή παραπάνω αστυνομικοί που συνεργάζονται μεταξύ τους για την σύλληψη των ληστών;

Επίσης με την ίδια λογική θα μπορούσαν να δημιουργηθούν και άλλα σενάρια, δημιουργώντας άλλους ρόλους και συμπεριφορές πρακτόρων. Για παράδειγμα: Μπορεί να δημιουργηθεί μια προσομοίωση σε περίπτωση ληστείας σε μια τράπεζα. Έτσι εκτός από τους ρόλους του κλέφτη και του αστυνόμου θα πρέπει να δημιουργηθούν και πράκτορες με ρόλους πελάτη, οι οποίοι θα αντιδρούν ο καθένας διαφορετικά σε περίπτωση ληστείας, δηλαδή θα έχουν διαφορετική συμπεριφορά.

Γενικότερα, το POSH έχει χρησιμοποιηθεί για την παραγωγή συμπεριφοράς πρακτόρων για την έρευνα κάποιων καταστάσεων, όπως για παράδειγμα τη μοντελοποίηση των κοινωνικών αλληλεπιδράσεων σε ένα πρωτόγονο οικισμό. Στη μελέτη αυτή κάθε πράκτορας μέσα στον οικισμό έχει τη δική του συμπεριφορά και το δικό του έλεγχο για τις πράξεις του. Προσωρινά όλοι οι πράκτορες μοιράζονται το ίδιο πλάνο POSH, αλλά αντικείμενο μιας μελλοντικής έρευνας θα μπορούσε να είναι η μελέτη ετερογενών κοινοτήτων. Οι πράκτορες έχουν σχεδιαστεί με το μοντέλο συμπεριφοράς των ζώων, τις αλληλεπιδράσεις μεταξύ τους σε έναν πρωτόγονο οικισμό. Το μοντέλο αυτό δείχνει ότι οι πράκτορες ταλαντεύονται μεταξύ δύο επιθυμητών καταστάσεων, την επιθυμία να βρουν ταίρι και την επιθυμία να μείνουν μόνοι τους.

Μια άλλη μελέτη στην οποία γίνεται χρήση των POSH δυναμικών πλάνων είναι αυτή του Tyrell, η οποία αναφέρθηκε αναλυτικά σε προηγούμενο κεφάλαιο. Ο Tyrell [1993] δημιούργησε ένα περιβάλλον προσομοίωσης στο οποίο κατοικεί ένα τρωκτικό. Το τρωκτικό προσπαθεί να επιβιώσει σε μια σαβάνα, γεμάτη από κινδύνους, παθητικούς και ενεργητικούς, ψάχνοντας να βρει τροφή, καταφύγιο και ευκαιρίες αναπαραγωγής. Με τη χρήση ενός πλάνου POSH μπορεί το τρωκτικό να επιλέξει ποια θα είναι η επόμενή του κίνηση με βάση τους στόχους που έχει, την εσωτερική του κατάσταση αλλά και την κατάσταση του περιβάλλοντος γύρω του. Τέτοιες έρευνες διευκολύνει αρκετά η χρήση των POSH πλάνων.

Τέλος τα Posh πλάνα μπορούν να χρησιμοποιηθούν για τη βελτιστοποίηση διαφόρων εφαρμογών όπως για παράδειγμα εφαρμογών στη βιομηχανία. Η μεθοδολογία BOD με τη χρήση του POSH Action Selection θα μπορούν να επιλύσουν το πρόβλημα του διαλόγου διαχείρισης σε ένα σύστημα, όπως τοTRAINS-93 [Allen et al., 1995]. Αυτό το σύστημα είναι μια σημαντική προσπάθεια για την αντιμετώπιση του προβλήματος του διαλόγου. Είναι ένα σύστημα ικανό να σχεδιάζει πλάνα και ενέργειες καθώς και να συζητά τα πλάνα και τους στόχους προφορικά. Το σύστημα TRANS θα μπορούσε να εξελιχθεί σε βοηθό διαχειριστή που επιχειρεί να κάνει διανομές εμπορευμάτων, όπως μπανάνες και χυμούς πορτοκαλιού, σε διάφορες πόλεις. Επίσης, στην περίπτωση πόλεων που παράγουν διαφορετικά προϊόντα και συνδέονται μεταξύ τους με διαφορετικά συγκοινωνιακά μέσα, όπως τρένα, αυτοκίνητα κ.λπ., θα μπορούσε να κάνει τον προγραμματισμό των μεταφορών τόσο σε χρόνο, όσο και σε χώρο. Αυτό το σύστημα μπορεί να εξελιχθεί με τη χρήση του POSH μηχανισμού επιλογής ενέργειας, δηλαδή με τη χρήση reactive πλάνων.

10. Βιβλιογραφία

Bryson J.Joanna, “Intelligence by Design:Principles of Modularity and Coordination for Engineering Complex Adaptive Agents”, MASSACHUSETTS INSTITUTE OF TECHNOLOGY, September 2001.

Bryson J.Joanna, **“**Action Selection and Individuation in Agent Based Modelling”, University of Bath**,** Department of Computer Science Artificial models of natural Intelligence ,April 26 2004.

Bryson J.Joanna, **“**The Behavior-Oriented Design of Modular Agent Intelligence”, University of Bath, Department of Computer Science Bath BA2 7AY, United Kingdom.

Bryson J.Joanna, Brendan McGonigle, “Agent Architecture as Object Oriented Design”, Laboratory for Cognitive Neuroscience and Intelligent Systems, Edinburgh University Edinburgh EH8 9JZ, UK.

Bryson J.Joanna, Lynn Andrea Stein, “Architectures and Idioms: Making Progress in Agent Design”, Artificial Intelligence Laboratory, MIT 545 Technology Square, Cambridge MA 02139,USA.

Kwong Andy, “A Framework for Reactive Intelligence through Agile Component-Based Behaviors”, University of Bath, September 2003.

Bryson J.Joanna, Tristan J. Caulfield , Jan Drugowitsch, “Integrating Life-Like Action Selection into Cycle-Based Agent Simulation Environments”, University of Bath**,** Department of Computer Science Artificial models of natural Intelligence (AmonI) , July 30 2006.

Estall Mary, “Creating an Assistive Intelligent Environment using Behaviour Orientated Design (BOD)”, Computer Science , University of Bath, April 2007.

Bryson J.Joanna, Kristinn R. Th´orisson, “Dragons, Bats & Evil Knights: A Three-Layer Design Approach to Character Based Creative Play”, Artificial Intelligence Laboratory, MIT 545 Technology Square, Cambridge MA 02139, USA.

Bryson J.Joanna, Tristan J. Caulfield, Jan Drugowitsch, **“**Integrating Life-Like Action Selection into Cycle-Based Agent Simulation Environments”, , University of Bath, Department of Computer Science Artificial models of natural Intelligence (AmonI) ,July 30, 2006.

Doran Jim, “Agent Design for Agent-Based Modelling”, Department of Computer Science, University of Essex.

Partington, S. J., **“**A critical analysis of Behaviour-Oriented Design (BOD), based on experiences in using it to create an Unreal Tournament Capture-the-Flag (CTF) team. Technical Report”, Department of Computer Science ,University of Bath, (CSBU-2005-05), 2005.

Davies N.P., Q.H.Mehdi, “BDI for Intelligent Agents in Computer Games”, Computer Games Centre School of Computing and Information Technology, University of Wolverhampton, Wolverhampton, UK.

Hindriks V.Koen, Birna van Riemsdijk, Tristan Behrens, Rien Korstanje, Nick Kraayenbrink, Wouter Pasman, Lennard de Rijk, “Unreal Goal Bots Connecting Agents to Complex Dynamic Environments”, Delft University of Technology, Mekelweg 4, 2628 CD, Delft, The Netherlands , Clausthal University of Technology, Julius-Albert-Straÿe 4, 38678 Clausthal, Germany.

Davies N. P., Q. H. Mehdi, N. E. Gough, “TOWARDS INTERFACING BDI WITH 3D GRAPHICS ENGINES”, Games Simulation and AI Centre Research Institute for Advanced Technologies

School of Computing and Information Technology University of Wolverhampton Wolverhampton, UK.

DiWang, Budhitama Subagdja, Ah-Hwee Tan, Gee-Wah Ng, “Creating Human-like Autonomous Players in Real-time First Person Shooter Computer Games”, Intelligent Systems Centre Nanyang Technological University Nanyang Drive, Singapore, School of Computer Engineering Nanyang Technological University Nanyang Avenue, Singapore, DSO National Laboratories 20 Science Park Drive Singapore.

Gemrot Jakub, Cyril Brom, Rudolf Kadlec, Michal Bvda, Ondřej Burkert, Michal Zemčak, Radek Pvbil, Tomaš Plch, “Pogamut 3 – Virtual Humans Made Simple”, Charles University in Prague, Faculty of Mathematics and Physics, Department of Software and Computer Science Education. Prague, Czech Republic.

Wang Jijun, Stephen Balakirsky, “USARSim V3.1.3: A Game-based Simulation of mobile robots”, University of Pittsburgh

Whiting S.Jeffrey, “COGNITIVE AND BEHAVIORAL MODEL ENSEMBLES FOR AUTONOMOUS VIRTUAL CHARACTERS”, Brigham Young University, Computer Science, August 2007

Σπυρίδωνας Βοσινάκης, «ΕΥΦΥΕΙΣ ΠΡΑΚΤΟΡΕΣ ΣΕ ΕΙΚΟΝΙΚΑ ΠΕΡΙΒΑΛΛΟΝΤΑ», Πειραιάς, 2003

Δικτυακός ιστότοπος με πληροφορίες για το μηχανισμό επιλογής ενέργειας POSH από το πανεπιστήμιο του Bath, "POSH Action Selection": http://www.cs.bath.ac.uk/~jjb/web/posh.html

Δικτυακός ιστότοπος με πληροφορίες για την αρχιτεκτονική BOD από το πανεπιστήμιο του Bath, "BOD Reactive Plans": http://www.cs.bath.ac.uk/~jjb/web/BOD/AgeS02/node9.html

Δικτυακός ιστότοπος με πληροφορίες για την υλοποίηση του μηχανισμού επιλογής ενέργειας POSH από το πανεπιστήμιο του Bath," Introduction to the Strict Slip-Stack POSH Implementation ": http://www.cs.bath.ac.uk/~jjb/web/BOD/sposh.html

Δικτυακός ιστότοπος με πληροφορίες για μηχανισμούς επιλογής ενέργειας, " Action Selection and Individuation in Agent Based Modelling ": http://www.cs.bath.ac.uk/~jjb/web/as4iabm/

Δικτυακός ιστότοπος με πληροφορίες για την πλατφόρμα, "Pogamut 3 CookBook": http://diana.ms.mff.cuni.cz/pogamut_files/latest/doc/tutorials/

Δικτυακός ιστότοπος με πληροφορίες για τα GameBots: http://diana.ms.mff.cuni.cz/pogamut_files/latest/doc/gamebots/ch01.html

Δικτυακός ιστότοπος με πληροφορίες για τα τον Unreal Editor: http://www.angelmapper.com/gamedev/tutorials/beginner1.htm

Δικτυακός ιστότοπος με πληροφορίες για Unreal Script, " Legacy UnrealScrip "http://wiki.beyondunreal.com/Legacy:UnrealScript

11.Παράρτημα : Οδηγίες εγκατάστασης της εφαρμογής

Κατά την διάρκεια της εκπόνησης της εργασίας μια από τις κύριες δυσκολίες που συναντήθηκαν ήταν η ένωση των προγραμμάτων που απαιτούνταν για την υλοποίηση του πειράματος και γι' αυτό το λόγο παρακάτω ακολουθούν οι οδηγίες της εγκατάστασης αυτών.

1.Οδηγίες Εγκατάστασης Pogamut

Για την εγκατάσταση του Pogamut απαραίτητα προγράμματα είναι :

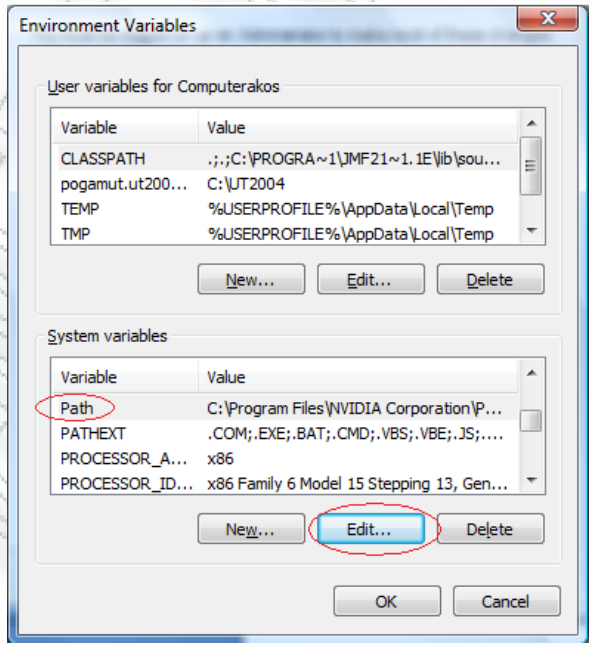
- Netbeans 6.9.1 (http://netbeans.org) την έκδοση JAVA SE
- JDK 1.6 και επόμενες εκδόσεις
- Maven 3.0.2 (http://archive.apache.org/dist/maven/binaries)
- UT2004
- Και το Pogamut 3.2.0 Standalone (http://diana.ms.mff.cuni.cz/main/tiki-index.php?page=Download)

Αρχικά εγκαθιστούμε το Netbeans 6.9.1 και στη συνέχεια το UT2004 που δεν έχουν κάποια ιδιαίτερη δυσκολία.

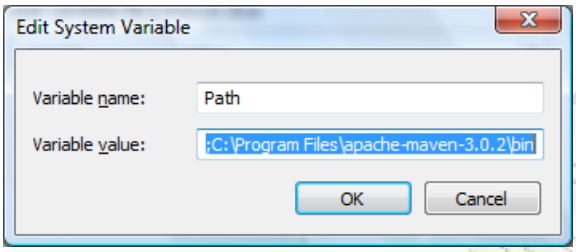
Στη συνέχεια εγκαθιστούμε το Pogamut (PogamutInstaller-full-3.2.0.jar). Το μοναδικό που προσέχουμε είναι τα directories που μας ζητάει να είναι τα σωστά.

Τέλος έχει σειρά το maven. Αρχικά κάνουμε unzip το αρχείο apache-maven-3.0.2.zip και τοποθετούμε το φάκελο apache-maven-3.0.2 στο Program Files . Στη συνέχεια προσθέτουμε το bin directory στο PATH. Αυτό γίνεται ως εξής:

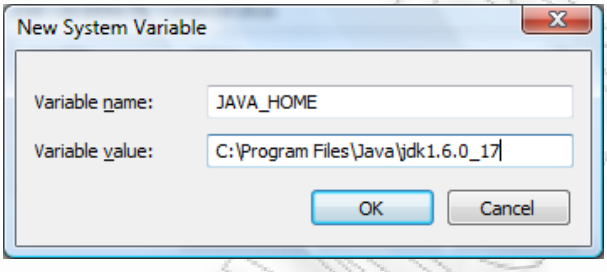
Στο System properties (WinKey + Pause) των WIndows στο tab Advance επιλέγουμε το Enviromental Varibles.



Στο System Variables επιλέγουμε το path και κάνουμε edit. Εκεί προσθέτουμε το directory του apache-maven-3.0.2\bin.

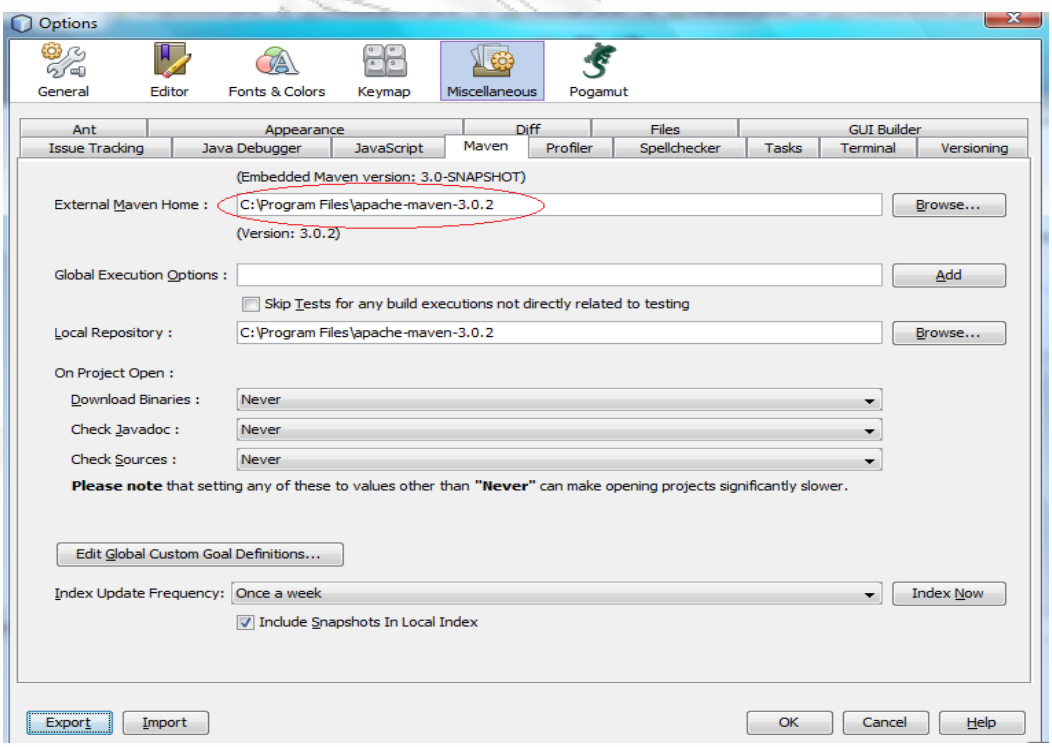


Επίσης στο System Variables προσθέτουμε άλλη μια μεταβλητή την JAVA_HOME επιλέγοντας το New.. Στο Variable name βάζουμε JAVA_HOME και στο value το directory που βρίσκεται το JDK. Για παράδειγμα:



Έπειτα Ok και Apply. Καλό θα ήταν στη συνέχεια να γίνει ένα Reboot στο Pc. Για να δούμε ότι το maven έχει εγκατασταθεί σωστά ανοίγουμε το command prompt και πληκτρολογούμε τα εξής: mvn –version. Αν αυτό τρέξει τότε είμαστε έτοιμοι να προχωρήσουμε.

Ανοίγουμε το Netbeans. Ακολουθούμε τη διαδρομή : (NetBeans->Tools->Options->Miscellaneous->sheet Maven και προσθέτουμε στο External Maven Home το directory που βρίσκεται το maven :



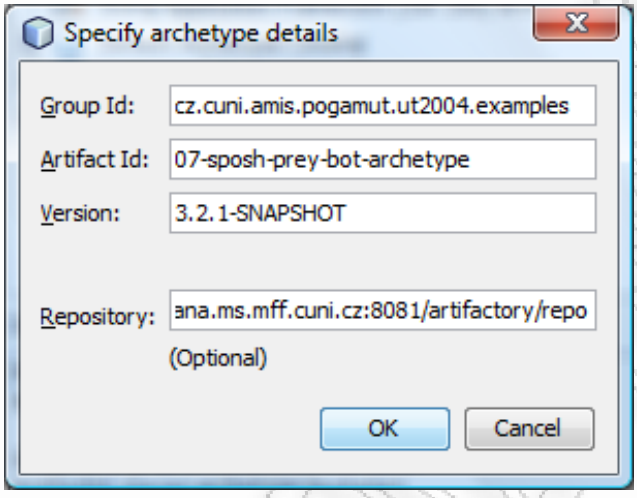
2. Δημιουργία SPOSH Agent

Τώρα όλα είναι έτοιμα ώστε να μπορέσουμε να δημιουργήσουμε το νέο μας Sposh agent. Επειδή αυτή η έκδοση του pogamut είναι πολύ πρόσφατη υπάρχουν κάποιες ελλείψεις (τουλάχιστον μέχρι στιγμής) και ο μοναδικός τρόπος για να ξεκινήσεις έναν sposh agent είναι φορτώνοντας το αρχικό παράδειγμα που σου παρέχει το πρόγραμμα. Από εκεί και πέρα μπορεί το παράδειγμα αυτό να αλλαχτεί σύμφωνα με τις προτιμήσεις του καθενός. Έτσι για τη δημιουργία ενός sposh agent ακολουθούμε τα εξής βήματα:

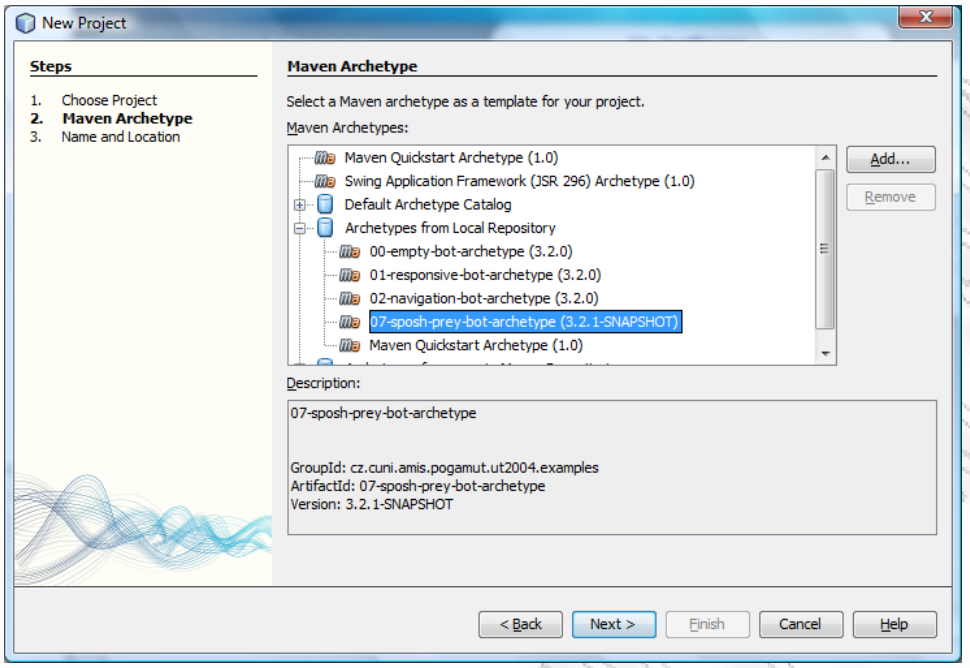
File → New Project→ Maven→ Maven Project → Next → Add

Εδώ μας ζητείται να προσθέσουμε τα χαρακτηριστικά του αρχέτυπου που είναι τα εξής:

Group Id: cz.cuni.amis.pogamut.ut2004.examples
Artifact Id: 07-sposh-prey-bot-archetype
Version: 3.2.1-SNAPSHOT
Repository: http://diana.ms.mff.cuni.cz:8081/artifactory/repo



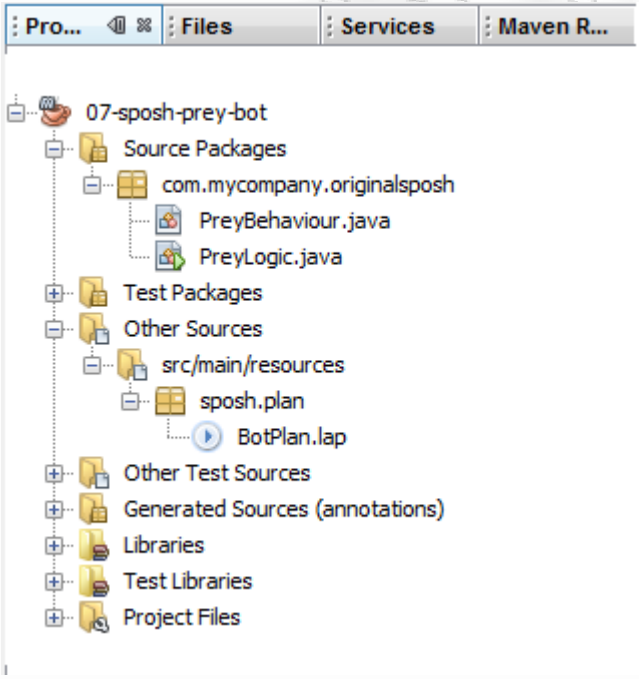
Προσθέτοντας αυτά δημιουργείται το 07-sposh-prey-bot-archetype που αποθηκεύεται στο Local Repository. Έτσι δε χρειάζεται να ακολουθήσουμε την ίδια διαδικασία αν θέλουμε να δημιουργήσουμε και άλλον sposh agent στο μέλλον.



Έπειτα πατώντας Next και στη συνέχεια Finish δημιουργείται το Project.

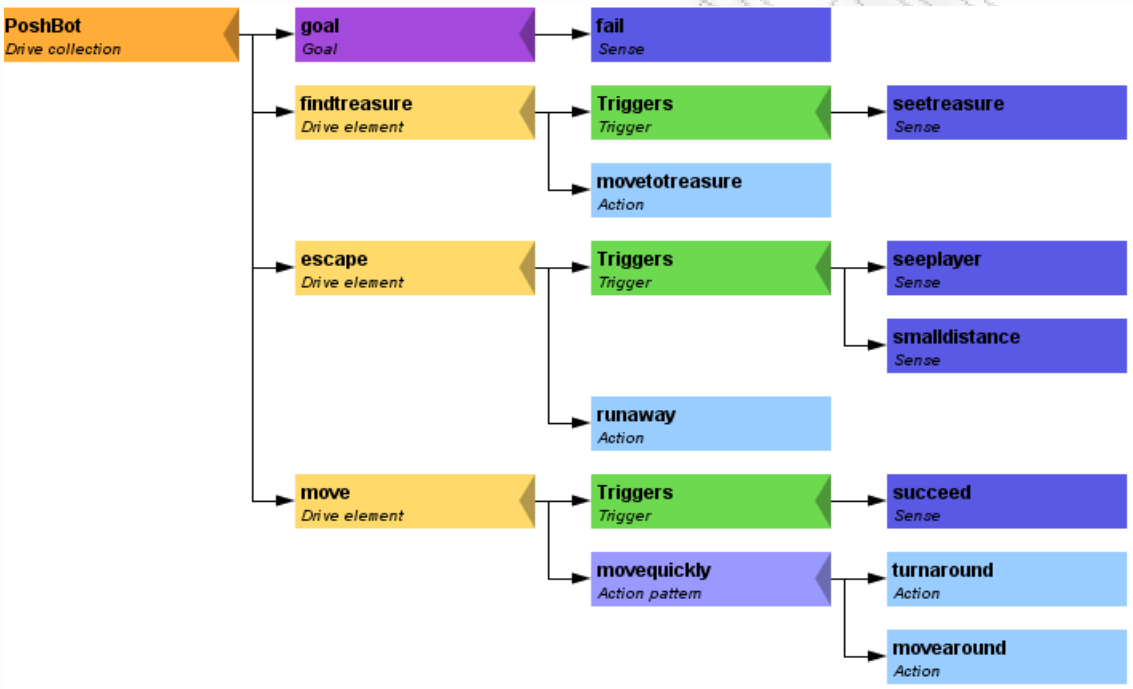
Προσοχή! Αν κατά τη δημιουργία του project το project μας είναι Badly-formed maven project απλά κάνουμε ένα Build στο project μας και ίσως και ένα restart στο Netbeans. Αυτό είναι πρόβλημα του Netbeans . Παρουσιάζεται μόνο την πρώτη φορά της δημιουργίας ενός Maven Project κατά αυτόν τον τρόπο.

Το Project εμφανίζεται ως εξής:



Το πλάνο του Posh που είναι της μορφής .lap βρίσκεται στο Other Sources→ src/main/resources→sposh.plan→ BotPlan.lap όπως φαίνεται και παραπάνω. Οι συμπεριφορά του agent βρίσκεται στο PreyBehaviour.java. Στο PreyLogic.java είναι υλοποιημένη η main και άλλες βασικές λειτουργίες όπως για παράδειγμα η διαδικασία «μετάφρασης» του πλάνου (parsing). Εμείς θα ασχοληθούμε κυρίως με το BotPlan.lap και με το PreyBehaviour.java. Τα ονόματα των κλάσεων αυτών στη συγκεκριμένη εργασία άλλαξαν έτσι για παράδειγμα η PreyBehaviour.java μετονομάστηκε σε Thief1Beahaviour.java και η PreyLogic.java σε Thief1Logic.java όσον αφορά τη συμπεριφορά του γενναίου ληστή.

Για να ξεκινήσουμε τη διαδικασία υλοποίησης της συμπεριφοράς ενός πράκτορα ξεκινάμε από το πλάνο (BotPlan.lap). Έστω ότι θέλουμε να υλοποιήσουμε την συμπεριφορά του γενναίου κλέφτη που παρουσιάστηκε αναλυτικά παραπάνω. Σύμφωνα με αυτή έχουμε το εξής πλάνο:



Στο παραπάνω πλάνο παρατηρείται ότι τα πρωταρχικά senses του πράκτορα είναι το να αντιληφθεί την παρουσία ενός θησαυρού (seetreasure), του αντιπάλου (seeplayer) και την απόσταση μεταξύ τους (smalldistance), τα οποία πυροδοτούν τις ενέργειες «πήγαινε στο θησαυρό» (movetotreasure) και «φύγε μακριά από τον αντίπαλο» (runaway) αντίστοιχα. Οι δύο τελευταίες ενέργειες είναι η περιστροφή γύρω από τον εαυτό σου (turnaround) και η περιπολία γύρω από το θησαυρό (movearound). Άρα το πλάνο μας πιο συνοπτικά περιγράφεται ως εξής:

See treasure → Move to Treasure

See Player και Small Distance → Runaway

Τίποτα από τα παραπάνω → Turn around και Move around

Άρα τα actions και τα senses που πρέπει να υλοποιηθούν μέσα στην Thief1Behaviour.java είναι τα εξής :

Actions: movetotreasure, runaway, turnaround και movearound

Senses: seetreasure , seeplayer και smalldistance

Τα senses succeed και fail καθώς και το action doNothing είναι είδη υλοποιημένα στο πρόγραμμα.

Αναλυτικά η υλοποίησή τους:

```
@SPOSHSense
  public boolean smalldistance() {
       //Βρίσκει ποιό είναι το Location του.
       Location myLoc= bot.getLocation();
       //Ελέγχει αν υπάρχει κάποιος παίχτης σε μικρή απόσταση από αυτόν
       if ((players.getNearestVisiblePlayer().getLocation().getDistance(myLoc) < UnrealUtils.CHARACTER_COLLISION_RADIUS * 25))
      {
          return true; //Έχω μικρή απόσταση
       }
       return false; //Έχω μεγάλη απόσταση
  }
```

```
@SPOSHSense
  public boolean seeplayer() {
       //Ελέγχει αν βλέπει κάποιον
       if(players.getNearestVisiblePlayer()!=null )
       {
          return true; // Βλέπω κάποιον
       }
       return false; // Δεν βλέπω κάποιον
  }
```

```
@SPOSHSense
  public boolean seetreasure() {
     if( items.getVisibleItems(ItemType.MINI_HEALTH_PACK).size()>0)
     {
            return true;  // Βλέπω το θησαυρό
     }
            return false;  // Δεν  βλέπω το θησαυρό
```

```
@SPOSHAction
  public boolean movetotreasure() {
    //σταματά την εκτέλεση οποιουδήποτε pathexecution αν γίνεται. *
    pathExecutor.stop();
    //βρίσκει όλους τους θησαυρούς τους οποίους έχουμε και θεωρήσει σαν mini health packs.
    Item[] treasures = items.getVisibleItems(ItemType.MINI_HEALTH_PACK).values().toArray(new Item[]{});
    if (treasures.length == 0) {
       user.severe("Δεν υπάρχει κανένας θησαυρός στο χάρτη");
       return false;
     }
    //Κινήσου γρήγορα
    move.setRun();
    //Βρες το μονοπάτι από τον εαυτό σου έως το θησαυρό.. Ξέρουμε ότι είναι ένας γιαυτό και treasures[0]
    pathExecutor.followPath(pathPlanner.computePath(bot, treasures[0].getLocation()));
    return true;
}
```

```
@SPOSHAction
   public boolean movearound() {
         //σταματά όποιον pathexecutor τρέχει
         pathExecutor.stop();
        //Βρίσκει όλα τα navpoints του χάρτη
         Collection<NavPoint> navPoints = tabooNavPoints.filter(getWorldView().getAll(NavPoint.class).values());
        //Επιλέγει ένα τυχαίο navpoint
         NavPoint chosen = MyCollections.getRandom(navPoints);
        //Ορίζεται η κίνησή του σε τρέξιμο
         move.setRun();
         //Βρίσκει το path για αυτό το τυχαίο navpoint και πηγαίνει προς αυτό.
         IPathFuture<ILocated> pathHandle = pathPlanner.computePath(info.getLocation(), chosen);
         pathExecutor.followPath(pathHandle);
         return true;
}
```

```
@SPOSHAction
   public boolean turnaround() {
         //Γυρίζει γύρω από τον εαυτό του κατά 180 μοίρες
         move.turnHorizontal(180);
         return true;
   }
```

```
@SPOSHAction
   public boolean runaway() {
         pathExecutor.stop();
         //Βρίσκει το Location του αντιπάλου
         Location plLoc = players.getNearestVisiblePlayer().getLocation();
         NavPoint chosenone = null;
         double maxdistance = 0.0;
         //Βρίσκει το navpoint με τη μεγαλύτερη απόσταση από τον αντίπαλο
         for (NavPoint navpoints : bot.getWorldView().getAll(NavPoint.class).values()) {
            double x =navpoints.getLocation().getDistance(plLoc);
            if (x>maxdistance){
               chosenone = navpoints;
               maxdistance=x;
            }
         }
         //Ορίζεται η κίνησή του σε τρέξιμο
         move.setRun();
         System.out.println("To navpoint me ti megaluteri apostasi pou diale3a einai " +chosenone);
         //Βρίσκει το path για αυτό το navpoint και πηγαίνει προς αυτό
         IPathFuture<ILocated> pathHandle1 = pathPlanner.computePath(info.getLocation(), chosenone);
         pathExecutor.followPath(pathHandle1);
         return true;
      }
}
```

Η κλάση Thief1Logic.java περιέχει την main κλάση και άλλες λειτουργίες όπως τη διαδικασία «μετάφρασης» του πλάνου (parsing), την αρχικοποίηση του πράκτορα, και είναι η εξής:

```
public class Thief1Logic extends SposhLogicController<UT2004Bot> {
   private String SPOSH_PLAN_RESOURCE = "sposh/plan/BotPlan.lap";
   private Thief1Behaviour behaviour;
   @Override
   public void initializeController(UT2004Bot bot) {
      super.initializeController(bot);
      behaviour = new Thief1Behaviour("Thief1Behaviour", bot);
   }
   @Override //διαβάζει το plan και το κάνει parsing
   protected String getPlan() throws IOException {
      return getPlanFromResource(SPOSH_PLAN_RESOURCE);
   }
   @Override
   protected IWorkExecutor createWorkExecutor() {
      return new BehaviorWorkExecutor(behaviour);
   }
   @Override
   public Initialize getInitializeCommand() {
// Δήλωση της θέσης που θα ξεκινάει ο πράκτορας που είναι το PlayerStart 4 επειδή αυτό βρίσκεται πιο κοντά στον θησαυρό.
      String Base = "PlayerStart4";
      Location Loc= null;
      for (NavPoint playerstart : bot.getWorldView().getAll(NavPoint.class).values()) { //από όλα τα navpoints
         if (playerstart.getId().getStringId().contains(Base)) {//αν κάποιο έχει το όνομα PlayerStart4
            Loc= playerstart.getLocation();//βρες το location του
         }
      }
```

συνέχεια..

```
  this.getAct().act(new SetSkin().setSkin("HumanMaleA.MercMaleA")); //δήλωση του skin του πράκτορα όμως σε αυτή την έκδοση δεν πραγματοποιείται.

  return new Initialize().setLocation(Loc).setName("THIEF BRAVE"); //δήλωση του ονόματος με το οποίο θα εμφανίζεται ο πράκτορας στο παιχνίδι και της τοποθεσίας.

   }

  @Override

    public void botInitialized(GameInfo gameInfo, ConfigChange currentConfig,

                        InitedMessage init) {

      behaviour.botInitialized(gameInfo, currentConfig, init);

    }

    @Override

    public void botSpawned(GameInfo gameInfo, ConfigChange currentConfig,

                        InitedMessage init, Self self) {

      behaviour.botSpawned(gameInfo, currentConfig, init, self);

      //Αλλαγή του health του κλέφτη από 100 σε 30 κατά την εκκίνησή του.

      this.getAct().act(new ChangeAttribute().setHealth(30));

    }

    @Override

    public void botKilled(BotKilled event) {

      behaviour.botKilled(event);

    }

    // H main.

    public static void main(String[] args) throws PogamutException {

        new UT2004BotRunner(Thief1Logic.class, "SPOSH-Prey").setMain(true).startAgent();

    }

}
```

3. Σύνδεση στο UT2004

Για τη σύνδεση του Bot μας στο UT2004 έχει απομείνει ένα βήμα ακόμα:

Στα Services (στο Netbeans) υπάρχει ένας κόμβος UT2004. Πάνω σε αυτόν και με δεξί κλικ (add server) δημιουργούμε μια δική μας σύνδεση. Σε αυτή βάζουμε ένα όνομα π.χ Local UT και στο URI το localhost:3001.

Τώρα τρέχουμε το UT2004 και δημιουργούμε HOST GAME. Στα GAME TYPES επιλέγουμε οποιοδήποτε από τα GAMEBOTS π.χ. Gamebots DeathMatch. Στη συνέχεια διαλέγουμε οποιαδήποτε πίστα και στο Server Rules προσέχουμε το όνομα του Server να είναι το ίδιο με αυτό που είχαμε βάλει προηγουμένως (Local UT). Όταν είμαστε έτοιμοι πατάμε το LISTEN. Καθώς το παιχνίδι δημιουργηθεί εκτελούμε το project στο Netbeans και ο ιδού ο πράκτορας!!

12.Παράρτημα : Δημιουργία νέου τύπου παιχνιδιού στο UT2004

Για τη δημιουργία του νέου gametype ακολουθήθηκε η εξής διαδικασία:

Αρχικά στο UT2004 δημιουργείται ένας νέος φάκελος (KaterinaGBScenario) ο οποίος περιέχει έναν υποφάκελο με όνομα Classes. Εκεί προστίθενται όλες οι κλάσεις που θα χρησιμοποιηθούν. Στην περίπτωσή μας έχουμε μια κλάση, την KaterinaPogamutScenario.uc. Βασική ιδιότητα αυτής της κλάσης είναι να κληρονομεί τα χαρακτηριστικά ενός DeathMatch. Έτσι έχουμε τα εξής:

```
class KaterinaPogamutScenario extends BotDeathMatch
```

Στη συνέχεια αυτό που μας ενδιαφέρει να τροποποιήσουμε στη deathmatch είναι οι κανόνες τερματισμού του παιχνιδιού. Έτσι αυτό που θέλουμε να αλλάξουμε είναι η συνάρτηση CheckEndGame() η οποία ελέγχει πότε το παιχνίδι τελειώνει. Η CheckEndGame είναι η εξής:

```
function bool CheckEndGame(PlayerReplicationInfo Winner, string Reason)
{
    local Controller P, NextController;
    local PlayerController Player;
    local bool bLastMan;
        if ( bOverTime )                                    ..συνέχεια
```

```
{
		if ( Numbots + NumPlayers == 0 )
			return true;
		bLastMan = true;
		for ( P=Level.ControllerList; P!=None; P=P.nextController )
			if ( (P.PlayerReplicationInfo != None) && !P.PlayerReplicationInfo.bOutOfLives )
			{
				bLastMan = false;
				break;
			}
		if ( bLastMan )
			return true;
	}

  bLastMan = ( Reason ~= "LastMan" );
	if (treasure> 0)// Εάν η μεταβλητή treasure έχει τιμή μεγαλύτερη του 0 τότε τερμάτισε το παιχνίδι
  {
		GotoState('MatchOver');
  }

  if ( !bLastMan && (GameRulesModifiers != None) && !GameRulesModifiers.CheckEndGame(Winner, Reason) )
	{
	    return false;
	}
	if ( Winner == None )
	{
		// find winner
		for ( P=Level.ControllerList; P!=None; P=P.nextController )
			if ( P.bIsPlayer && !P.PlayerReplicationInfo.bOutOfLives
			    && ((Winner == None) || (P.PlayerReplicationInfo.Score >= Winner.Score)))
			{
				Winner = P.PlayerReplicationInfo;
			}
	}
```

..συνέχεια

```
// check for tie
if ( !bLastMan )
{
        for ( P=Level.ControllerList; P!=None; P=P.nextController )
        {
                if ( P.bIsPlayer &&
                        (Winner != P.PlayerReplicationInfo) &&
                        (P.PlayerReplicationInfo.Score == Winner.Score)
                && !P.PlayerReplicationInfo.bOutOfLives )
                {
                        if ( !bOverTimeBroadcast )
                        {
                                StartupStage = 7;
                                PlayStartupMessage();
                                bOverTimeBroadcast = true;
                        }
                        return false;
                }
        }
    }
EndTime = Level.TimeSeconds + EndTimeDelay;
GameReplicationInfo.Winner = Winner;
    log("winner= " $ GameReplicationInfo.Winner);
if ( CurrentGameProfile != None )
    {
        CurrentGameProfile.bWonMatch = (PlayerController(Winner.Owner) != None);
    }
EndGameFocus = Controller(Winner.Owner).Pawn;
```

..συνέχεια

```
 if ( EndGameFocus != None )
                    EndGameFocus.bAlwaysRelevant = true;
 for ( P=Level.ControllerList; P!=None; P=NextController )
  {
    Player = PlayerController(P);
    if ( Player != None )
    {
      if ( !Player.PlayerReplicationInfo.bOnlySpectator )
                PlayWinMessage(Player, (Player.PlayerReplicationInfo == Winner));
      Player.ClientSetBehindView(true);
      if ( EndGameFocus != None )
      {
        Player.ClientSetViewTarget(EndGameFocus);
        Player.SetViewTarget(EndGameFocus);
      }
      Player.ClientGameEnded();
    }
    NextController = P.NextController;
    P.GameHasEnded();
  }
  return true;
}
```

Η μεταβλητή treasure ορίζεται και αρχικοποιείται στην κλάση BotDeathMatch.uc, η οποία βρίσκεται στο φάκελο GameBots2004, ως εξής:

```
var config int treasure;
```

Η τιμή της αλλάζει κάθε φορά που ένας πράκτορας θα συλλέξει ένα *MiniHealthPack,* το οποίο και θεωρείται ο θησαυρός. Η αλλαγή της τιμής της ορίζεται στην κλάση RemoteBot.uc που βρίσκεται και αυτή στο φάκελο GameBots2004 και συγκεκριμένα στη συνάρτηση

HandlePickup η ενημερώνει το πότε ο πράκτορας σύλλεξε κάποιο αντικείμενο. Εκεί έχει προστεθεί το εξής:

```
if (pickup.IsA('MiniHealthPack')) {    // Όταν ο πράκτορας συλλέξει ένα MiniHealthPack(θησαυρός)
                    log("Pick helath up");
                    //αυξάνεται η τιμή της μεταβλητής treasure κατά 1, η οποία βρίσκεται στην κλάση
                    // BotDeathMatch (Gamebots2004 directory)
                    BotDeathMatch(Level.Game).treasure++;
                    //Στη συνέχεια καλείται η συνάρτηση EndGame, η οποία με τη σειρά της καλεί τη
                    //συνάρτηση CheckEndGame (που έχουμε ήδη τροποποιήσει κατάλληλα)
                    Level.Game.EndGame(None,"LastMan");
          }
```

Για να εκτελεστούν τα παραπάνω πρέπει να πραγματοποιηθούν άλλα τρία βήματα.

Αρχικά πρέπει στο αρχείο UT2004.ini, που βρίσκεται στο φάκελο UT2004\System, να προστεθεί το εξής:

EditPackages=KaterinaGBScenario

Έπειτα θα πρέπει να σβηστούν τα αρχεία GameBots2004.ucl και GameBots2004.u που βρίσκονται στο UT2004\System.

Τέλος, μέσα από το command prompt θα πρέπει στο directory UT2004\System να εκτελεστεί η εντολή ***ucc make.***

Στη συνέχεια ξεκινάμε το Unreal Tournament και ιδού το νέο μας παιχνίδι!!