Πανεπιστήμιο Ιωαννίνων
Τμήμα Πληροφορικής και Τηλεπικοινωνιών

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

Υλοποίηση Τεχνικών Μηχανικής Μάθησης σε περιβάλλον Μοντελοποίησης και Προσομοίωσης Κυβερνο-φυσικών Συστημάτων

Σωτήριος Τζαμάρας

Επιβλέπων: Αδάμ Σταύρος
Επίκουρος Καθηγητής

Άρτα, Νοέμβριος, 2020

Implementing Machine Learning Techniques in a Modeling and Simulation Environment for Cyber-Physical Systems

Εγκρίθηκε από τριμελή εξεταστική επιτροπή
Άρτα, 5 Νοεμβρίου 2020

ΕΠΙΤΡΟΠΗ ΑΞΙΟΛΟΓΗΣΗΣ

1. Επιβλέπων καθηγητής
   Σταύρος Αδάμ,
   Επίκουρος Καθηγητής

2. Μέλος επιτροπής
   Γεωργία Φουτσιτζή,
   Καθηγήτρια

3. Μέλος επιτροπής
   Walid Taha,
   Καθηγητής

Δήλωση μη λογοκλοπής

Δηλώνω υπεύθυνα και γνωρίζοντας τις κυρώσεις του Ν. 2121/1993 περί Πνευματικής Ιδιοκτησίας, ότι η παρούσα πτυχιακή εργασία είναι εξ ολοκλήρου αποτέλεσμα δικής μου ερευνητικής εργασίας, δεν αποτελεί προϊόν αντιγραφής ούτε προέρχεται από ανάθεση σε τρίτους. Όλες οι πηγές που χρησιμοποιήθηκαν (κάθε είδους, μορφής και προέλευσης) για τη συγγραφή της περιλαμβάνονται στη βιβλιογραφία.

Τζαμάρας, Σωτήριος

Υπογραφή

Ευχαριστίες

Πολλοί άνθρωποι έχουν συνεισφέρει στο Acumen, στη θεωρία του και στις εφαρμογές που περιγράφονται σε αυτή τη πτυχιακή. Ο σχεδιασμός και η εφαρμογή της γλώσσας καθοδηγήθηκαν από τον Walid Taha. Η αρχική εργασία στο γραφικό περιβάλλον χρήστη και ο traditional διερμηνέας υλοποιήθηκε από τον Paul Brauner. Η αρχική εργασία στον optimized διερμηνέα έγινε από τον Kevin Atkinson. Ο τρέχων traditional και enclosure διερμηνέας, η τρισδιάστατη οπτικοποίηση και οι υποκείμενες βιβλιοθήκες αναπτύχθηκαν από τους Yingfu Zeng, Adam Duracz, Ferenc A. Barth και Fei Xu.

Σε μια πιο προσωπική σημείωση, υπάρχουν πολλά άτομα που θα ήθελα να ευχαριστήσω, χωρίς τους οποίους δεν θα μπορούσα να ολοκληρώσω αυτήν τη πτυχιακή. Αρχικά, θα ήθελα να εκφράσω την ειλικρινή ευγνωμοσύνη μου στον επιβλέποντα καθηγητή μου Σταύρο Αδάμ, για την εμπιστοσύνη που μου έδειξε, την καθοδηγησή του, τις πολύτιμες συμβουλές και τις αμέτρητες μας συζητήσεις. Στον σύμβουλο καθηγητή μου Walid Taha, για τη συνεχή υποστήριξη, την υπομονή, τα κίνητρα και τις τεράστιες γνώσεις του. Δεν θα μπορούσα να φανταστώ να είχα καλύτερους μέντορες για αυτή τη πτυχιακή.

Τέλος, θα ήθελα να ευχαριστήσω την οικογένειά μου: τους γονείς μου και τον αδερφό μου για την ανεπιφύλακτη υποστήριξή τους σε αυτό το ταξίδι και γενικά στη ζωή μου.

Περίληψη

Τα κυβερνο-φυσικά συστήματα είναι συζευγμένα συστήματα, αποτελούμενα από ψηφιακές και φυσικές οντότητες που αλληλεπιδρούν μεταξύ τους. Οι προσομοιώσεις και οι δοκιμές της επιθυμητής συμπεριφοράς τέτοιων υβριδικών συστημάτων, τα οποία παρουσιάζουν διακριτή αλλά και συνεχή δυναμική συμπεριφορά, είναι δύσκολες και συχνά επικίνδυνες. Μερικά παραδείγματα τέτοιων συστημάτων είναι τα αυτόνομα αυτοκίνητα, οι ρομποτικές εφαρμογές και οι ιατρικές συσκευές παρακολούθησης. Η ικανότητα μοντελοποίησης τέτοιων συστημάτων απαιτεί ανάλυση φυσικών φαινομένων και εφαρμογή εκτεταμένων δοκιμών για απρόσμενα συμβάντα.

Σε αυτή τη πτυχιακή, δείχνουμε πώς μέσω της χρήσης μεθόδων μηχανικής μάθησης μπορούμε να συμβάλουμε στην ενίσχυση των κυβερνο-φυσικών συστημάτων με την επίτευξη ευφυούς συμπεριφοράς. Οι αυτόνομοι πράκτορες μέσω της αξιοποίησης μιας προσέγγισης που ονομάζεται state-action-reward-state, είναι σε θέση να μάθουν πώς να βελτιστοποιήσουν τη συμπεριφορά τους και να δράσουν έξυπνα σε ένα άγνωστο περιβάλλον. Τέλος, παρέχουμε επίσης μια εφαρμογή μάθησης με ενίσχυση που δείχνει την προσέγγισή μας.

Λέξεις-κλειδιά: κυβερνο-φυσικά συστήματα, προσομοιώσεις, μοντελοποίηση, πράκτορες, έξυπνη συμπεριφορά.

Abstract

Cyber-physical systems are coupled systems of digital and physical entities that interact with each other. Simulating and testing the desired behavior of such hybrid systems, which exhibit both discrete and continuous dynamic behavior, is hard and often dangerous. Some examples are autonomous automobiles, robotics and medical monitoring devices. The ability to model such systems requires analysis of physical phenomena and extended testing for unexpected events.

In this thesis, we demonstrate how utilization of machine learning can be used to enhance cyber-physical systems to achieve intelligent behavior. Creating autonomous agents and utilizing a state-action-reward-state approach, they are capable of learning how to optimize their behavior and react to an unknown environment. We also provide a Reinforcement learning implementation demonstrating our approach.

Keywords: cyber-physical systems, simulations, modeling, agents, intelligent behavior.

Περιεχόμενα

Κατάλογος Σχημάτων

Κατάλογος Πινάκων

Πίνακας Συντομογραφιών

AI Artificial Intelligence - Τεχνητή Νοημοσύνη

AST Abstract Syntax Tree - Αφηρημένο Συντακτικό Δέντρο

BTA Binding Time Analysis - Ανάλυση Χρόνου Δέσμευσης

CLI Command Line Interface - Διεπαφή Γραμμής Εντολών

CPS Cyber-physical Systems - Κυβερνο-φυσικά Συστήματα

DSL Domain-specific Language - Γλώσσα Συγκεκριμένου Πεδίου Ορισμού

DAE Discrete-Algebraic Equations - Διακριτές-Αλγεβρικές Εξισώσεις

FMDP Finite Markov Decision Process - Πεπερασμένη Μαρκοβιανή Διεργασία Απόφασης

FRP Functional Reactive Programming - Λειτουργικός Προγραμματισμός Αντίδρασης

GE Gaussian Elimination - Απαλοιφή Gauss

GUI Graphical User Interface - Γραφικό Περιβάλλον Χρήστη

IDE Integrated Development Environment - Ολοκληρωμένο Περιβάλλον Ανάπτυξης

IoT Internet of Things - Διαδίκτυο των Πραγμάτων

JRE Java Runtime Environment - Περιβάλλον εκτέλεσης Java

JVM Java Virtual Machine - Εικονική Μηχανή Java

MDP Markov Decision Process - Μαρκοβιανή Διεργασία Απόφασης

ML Machine Learning - Μηχανική Μάθηση

MBD Model-based Design - Σχεδίασης βάσει Μοντέλου

ODE Ordinary Differential Equations - Συνήθεις Διαφορικές Εξισώσεις

RL Reinforcement Learning - Μάθηση με Ενίσχυση

SD Symbolic Differentiation - Συμβολική Παραγώγιση

1 Εισαγωγή

Αυτή η πτυχιακή αποσκοπεί να δείξει πώς σε μια Domain-specific Language - Γλώσσα Συγκεκριμένου Πεδίου Ορισμού (DSL) για προσομοίωση βάσει μοντέλων μπορούν να εφαρμοστούν τεχνικές μηχανικής εκμάθησης κατά τη λήψη αποφάσεων και στον έλεγχο προβλημάτων σε Cyber-physical Systems - Κυβερνο-φυσικά Συστήματα (CPS). Αυτό το τμήμα παρουσιάζει τα κίνητρα αυτής της πτυχιακής και περιγράφει τις συνεισφορές της. Το υπόλοιπο κείμενο της πτυχιακής οργανώνεται ως εξής: Η ενότητα 2 παρέχει μια επισκόπηση της γλώσσας Acumen και της σύνταξης της. Στην Ενότητα 3 περιγράφουμε πώς το περιβάλλον Acumen δημιουργεί μια προσομοίωση. Η Ενότητα 4 εισάγει τις γλωσσικές δυνατότητες που απαιτούνται για τη δημιουργία έξυπνης συμπεριφοράς και παρουσιάζει συγκεκριμένες συνεισφορές σε αλλαγές κώδικα και προσθήκες στην εφαρμογή της γλώσσας. Η Ενότητα 5 παρέχει μια μελέτη τόσο στη σύνταξη της γλώσσας Acumen όσο και της Python και κάνει τη συγκρίση μεταξύ τους. Τέλος, η Ενότητα 6 παρουσιάζει τα συμπεράσματα και κάποιες προτάσεις για μελλοντική έρευνα και ανάπτυξη.

1.1 Κίνητρο

Οι τεχνολογικές εξελίξεις στους τομείς της αρχιτεκτονικής των υπολογιστών και συγκεκριμένα των μικροελεγκτών, επέτρεψαν μειώσεις σε επίπεδο αρκετά μικρό ώστε να ενσωματώνονται σε σχεδόν οποιοδήποτε ηλεκτρονικό ή μηχανικό εξάρτημα. Τα συζευγμένα συστήματα που διαθέτουν υπολογιστικές δυνατότητες, αντιπροσωπεύοντας τον κυβερνοχώρο, όσο και ανατροφοδότηση δεδομένων σε πραγματικό χρόνο από φυσικά εξαρτήματα είναι γνωστά ως CPS [11]. Αυτή η γενιά ψηφιακών συστημάτων εισήγαγε μια ποικιλία χρήσεων στις καθημερινή μας ζωή, που απεικονίζονται καλύτερα με παραδείγματα όπως αυτόνομα χαρακτηριστικά σε μοντέρνα αυτοκίνητα, ρομποτικές εφαρμογές και έξυπνες πόλεις. Ο σχεδιασμός τέτοιων συστημάτων αποδείχθηκε εξαιρετικά περίπλοκος, επειδή υπάρχει ένα μεγάλο εύρος καταστάσεων στις οποίες μπορούν να βρεθούν. Για παράδειγμα, η δοκιμή ενός νέου αυτόνομου βοηθητικού χαρακτηριστικού σε ένα αυτοκίνητο που γίνεται σε κάποιο συνωστισμένο δρόμο, μπορεί να είναι εξαιρετικά επικίνδυνη και η διαδικασία κατασκευής πολύ δαπανηρή, κάνοντας ένα τέτοιο σύστημα ακατάλληλο για δοκιμές υπό ορισμένα σενάρια.

Ένας τρόπος αντιμετώπισης αυτού του εύρους πολυπλοκότητας είναι η χρήση μιας προσέγγισης Model-based Design - Σχεδίασης βάσει Μοντέλου (MBD), ενός μαθηματικού και οπτικού τρόπου αντιμετώπισης σύνθετων συστημάτων. Τέτοια μοντέλα χρησιμοποιούνται ως μαθηματική αναπαράσταση οποιουδήποτε συστήματος. Για παράδειγμα, οι ιδιότητες ενός συστήματος

μπορούν να καθοριστούν μέσω της χρήσης μεταβλητών ενώ κάποια μοντέλα μπορεί να περιέχουν εξισώσεις για την απεικόνιση της θέσης ορισμένων φυσικών εξαρτημάτων [12]. Τα εργαλεία μοντελοποίησης μπορούν στη συνέχεια να χρησιμοποιήσουν αυτές τις περιγραφές μοντέλου για τη δημιουργία προσομοιώσεων και τη μελέτη της συμπεριφοράς τους σε οποιαδήποτε από τις πιθανές καταστάσεις τους. Αυτή η ικανότητα παρέχει έναν ευκολότερο και φθηνότερο τρόπο δοκιμών, σε σύγκριση με την ανάπτυξη πραγματικών πρωτοτύπων στο φυσικό κόσμο.

Επιπρόσθετα, με τη συνεχή αύξηση των υπολογιστικής ισχύος και των πλέον πρόσφατων τεχνολογικών εξελίξεων, όλο και περισσότερες συσκευές μπορούν να επικοινωνούν μεταξύ τους, γεγονός που αυξάνει ακόμη περισσότερο την πολυπλοκότητα της μοντελοποίησης. Μία αρχιτεκτονική συστημάτων που επιτρέπει την επικοινωνία πολλών συσκευών μεταξύ τους είναι γνωστή ως Internet of Things - Διαδίκτυο των Πραγμάτων (IoT). Ένα άλλο ταχέως αναπτυσσόμενο πεδίο είναι η Artificial Intelligence - Τεχνητή Νοημοσύνη (AI) και συγκεκριμένα ένα από το πλέον σύγχρονο σύνολο τεχνολογιών του, το οποίο είναι η Machine Learning - Μηχανική Μάθηση (ML). Τέτοιες τεχνολογίες αποτελούν ήδη μέρος της καθημερινότητάς μας, με χρήσεις που κυμαίνονται από ¨έξυπνες¨ συσκευές όπως οικιακοί αυτοματισμοί και κινητά τηλέφωνα έως συσκευές παρακολούθησης υγείας και συστήματα μεταφοράς.

Τώρα περισσότερο από ποτέ, οι άνθρωποι βασίζονται στη σωστή λειτουργία τέτοιων συσκευών και μηχανημάτων, οπότε η απόδειξη της ορθής λειτουργίας τους πρέπει να είναι κάτι παραπάνω από υποχρεωτική. Όταν η ML χρησιμοποιείται σε συνδυασμό με τα CPS και το μοντέλο IoT, αναμένονται νέες και έξυπνες συμπεριφορές από αυτά τα συστήματα. Αυτό οφείλεται στο γεγονός ότι η ML είναι σε θέση να αναλύει τεράστιους όγκους δεδομένων, καθώς και να εξερευνά καταστάσεις οι οποίες ήταν αδύνατο να υπολογιστούν. Τέτοιες εφαρμογές μπορούν να χρησιμοποιηθούν για την παρατήρηση της συμπεριφοράς των CPS με αυτόνομο τρόπο και να οδηγήσουν σε έξυπνη λήψη αποφάσεων χωρίς ρητό προγραμματισμό του συστήματος.

1.2 Συνεισφορά

Αυτή η πτυχιακή επικεντρώθηκε σε μια διεξοδική διερεύνηση της γλώσσας μοντελοποίησης Acumen, παρουσιάζοντας τα ποσοστά υποστήριξης της γλώσσας για αλγόριθμους ML, χρησιμοποιώντας επεξεγηματικά παραδείγματα. Η συνεισφορά αυτής της εργασίας συνοψίζεται στα ακόλουθα αντικείμενα:

- Μια παρουσίαση των περιορισμών της γλώσσας (Ενότητα 4.3), όσον αφορά τη δυνατότητα υλοποίησης τεχνικών ML.
- Ένα σύνολο από αλλαγές στον κώδικα (Ενότητες 4.3.1 και 4.3.2), που εισάγουν διορθώσεις σφαλμάτων κατά την ανάθεση τιμών σε δομές δεδομένων όπως λίστες και πίνακες.
- Μια συλλογή από προσθήκες στην λειτουργικότητα δομών δεδομένων (π.χ. λίστες, πίνακες, κ.α) και στην αναφορά σφαλμάτων κατά την εκτέλεση των αντίστοιχων ενεργειών (Ενότητα 4.4).
- Μια περίπτωση μελέτης που χρησιμοποιεί τον αλγόριθμο Q-learning για την προσομοίωση ενός έξυπνου πράκτορα που πλοηγείται σε ένα λαβύρινθο (Ενότητα 5.2).
- Μια συλλογή από συγκρίσεις σε κώδικα (Ενότητα 5.3) μεταξύ του περιβάλλοντος Acumen και Python.

2 Acumen: Γλώσσα Μοντελοποίησης Υβριδικών Συστημάτων

Το Acumen [2] είναι μια πειραματική DSL γραμμένη σε Scala [20], μια γλώσσα προγραμματισμού που εκτελείται επάνω στην Java Virtual Machine - Εικονική Μηχανή Java (JVM). Ο σκοπός της γλώσσας είναι να μοντελοποιεί και να προσομοιώνει CPS, δυναμικά συστήματα που μπορούν να παρουσιάσουν διακριτή, συνεχή ή υβριδική (διακριτή και συνεχή) συμπεριφορά. Αν και μικρή σε μέγεθος, η γλώσσα παρέχει μια ποικιλία εκφράσεων στο χρήστη επιτρέποντας την προσομοίωση μεγαλύτερων συστημάτων, δίνοντας έμφαση στην ικανότητα μοντελοποίησης συστημάτων που περιέχουν πολλές μεταβλητές που αλλάζουν δυναμικά με την πάροδο του χρόνου.

Το Acumen αποτελεί ένα Integrated Development Environment - Ολοκληρωμένο Περιβάλλον Ανάπτυξης (IDE) που παρέχει μία ευρεία γκάμα από εργαλεία στον χρήστη, όπως επεξεργαστή κώδικα, αναφορά σφαλμάτων και ρεαλιστικές τρισδιάστατες κινούμενες εικόνες. Αυτή η ενότητα περιγράφει το πλαίσιο επινόησης και ανάπτυξης της γλώσσας, κάνει μια λεπτομερή εισαγωγή στη σύνταξη του Acumen, και παρουσιάζει το ενσωματωμένο Graphical User Interface - Γραφικό Περιβάλλον Χρήστη (GUI).

2.1 Κίνητρα και Προηγούμενη Εργασία

Το Acumen δημιουργήθηκε έχοντας ως κίνητρο την απουσία ισχυρών εργαλείων ανάπτυξης για μοντελοποίηση CPS. Υπάρχει ένα ακαθόριστο κενό μεταξύ εργαλείων επαλήθευσης και των εκτελέσιμων προσομοιωτών [36] και το Acumen χτίστηκε ως μια πειραματική γέφυρα μεταξύ των δύο. Ειδικότερα, ο πρωταρχικός στόχος της γλώσσας είναι να εξερευνήσει τρόπους σχεδιασμού ενός ¨*αυστηρού αλλά πρακτικού*¨ εργαλείου για προγραμματιστές CPS, ενώ ενσωματώνει μαθηματική ορθότητα, πρακτικότητα και επεκτασιμότητα [25].

Οι πρώτες υλοποιήσεις της γλώσσας υιοθέτησαν την ιδέα του Functional Reactive Programming - Λειτουργικός Προγραμματισμός Αντίδρασης (FRP), όπου τα διακριτά και τα συνεχή συμβάντα συνδυάζονται για να μοντελοποιήσουν ένα αντιδραστικό σύστημα [36]. Πρόσφατες υλοποιήσεις χρησιμοποιούν έναν πλήρη επανασχεδιασμό [12], με νέες προσθήκες στη γλώσσα. Το Acumen έχει χρησιμοποιηθεί για την διδασκαλία μαθημάτων [24, 23, 27] και περιλαμβάνεται επίσης σε ένα υπό έκδοση βιβλίο για τα CPS [31].

2.2 Διερμηνείς και Σημασιολογία

Στο Acumen υποστηρίζονται δύο τύποι σημασιολογίας και τρείς διαφορετικοί τύποι διερμηνέων [12]. Οι πρώτοι δύο τύποι διερμηνέων, αναφοράς (reference) και βελτιστοποιημένος (optimized), υλοποιούν την παραδοσιακή σημασιολογία (traditional semantics), η οποία χρησιμοποιεί αριθμούς κινητής υποδιαστολής (floating point numbers) για την αναπαράσταση πραγματικών αριθμών. Η συγκεκριμένη σημασιολογία επιλύει Ordinary Differential Equations - Συνήθεις Διαφορικές Εξισώσεις (ODE) κάνοντας χρήση παραδοσιακών μεθόδων όπως για παράδειγμα την μέθοδο Runge-Kutta τετάρτου βαθμού. Η διαφορά μεταξύ των δύο διερμηνέων είναι ότι ο διερμηνέας reference κάνει την δοκιμή νέων χαρακτηριστικών σημασιολογίας πιο εύκολη και αποτελεί σημείο αναφοράς για τις δοκιμές της σωστής λειτουργία του διερμηνέα optimized.

Ο τρίτος τύπος διερμηνέα, με το όνομα περιφραστικός (enclosure), υλοποιεί την περιφραστική σημασιολογία (enclosure semantics) και μέσω αυτής το Acumen είναι ικανό να δημιουργεί αυστηρές προσομοιώσεις. Αυτός ο τύπος σημασιολογίας χρησιμοποιεί ένα υποσύνολο της γλώσσας του Acumen και ο τρόπος λειτουργίας του περιγράφεται από παραγωγή προσεγγίσεων με τη χρήση άνω και κάτω ορίων για όλες τις προσομοιώσεις.

2.3 Σύνταξη

Σε αυτή την υπο-ενότητα παρουσιάζουμε τη σύνταξη της γλώσσας που θα χρησιμοποιηθεί στο υπόλοιπο κείμενο αυτής της πτυχιακής. Υποθέτουμε τη χρήση της έκδοσης *2016.08.30*[1] του Acumen καθώς και την επιλογή σημασιολογίας Reference 2015 από το μενού *Semantics/Traditional*. Το Acumen υποστηρίζει τις ακόλουθες δομές δεδομένων στη σύνταξη του:

- Τύποι Μεταβλητών
  - Ακέραιοι (π.χ. 1, 2, κ.α)
  - Πραγαματικοί αριθμοί (π.χ. 2.5, -9.8, κ.α)
  - Διαστήματα (π.χ. [0...10])
  - Δεδομένα αληθείας (true, false)
  - Χαρακτήρες (π.χ. "Acumen", "node", κ.α)
  - Κατοχυρωμένες σταθερές (π.χ pi, children, red, κ.α)

[1] Αυτή η έκδοση μπορεί να βρεθεί στην ιστοσελίδα του Acumen κάνοντας κλικ στο κουμπί Download. Διεύθυνση: ¨`http://www.acumen-language.org/`¨.

- Συλλογές δεδομένων (π.χ. λίστες, πίνακες, πλειάδες, κ.α)
- Τελεστές (π.χ. +, -, κ.α)
- Ενσωματωμένες συναρτήσεις (π.χ. sin, print, length, κ.α)
- Περιγραφές μοντέλων
  - Ορισμός (model Ball(x) = ...)
  - Δημιουργία και τερματισμό μοντέλων (create, terminate)
  - Τομείς (initially, always)
- Παράγωγοι μεταβλητών ως προς το χρόνο (π.χ. x', x", κ.α)
- Δηλώσεις υπό συνθήκες (if, match)
- Μεταβλητές σκηνής 3D
  - Στατικές ή δυναμικές λίστες με 3D αντικείμενα (π.χ. _3D = (Ball size=1))
  - Θέση κάμερας (π.χ. _3DView = ((0,0,0),(0,1,1)))
  - 3D αντικείμενα (π.χ. Ball, Obj, Cone, κ.α)
- Υποθέσεις (π.χ. hypothesis "Gravity remains constant" g==-9.8)

Επίσης το Acumen παρέχει υποστήριξη για σχόλια με τη χρήση των χαρακτήρων ASCII '//' όπου αγνοείται το υπόλοιπο της γραμμής, ή εμπεριέχοντας κείμενο μεταξύ των χαρακτήρων '/*' και '*/'.

2.3.1 Συναρτήσεις

Η προσομοίωση ενός υβριδικού συστήματος επιτυγχάνει την παρατήρηση της συμπεριφορά του με την πάροδο του χρόνου. Συνήθως, αυτή εκφράζεται χρησιμοποιώντας διαφορικές εξισώσεις. Υπάρχουν δύο τύποι εξισώσεων στο Acumen, διακριτές και συνεχείς. Ας υποθέσουμε ότι θέλουμε να μοντελοποιήσουμε ένα χρονόμετρο το οποίο μετράει πόσες φορές ένα συγκεκριμένο χρονικό πλαίσιο έχει παρέλθει. Αυτό είναι ένα συνεχές σύστημα του οποίου οι μεταβλητές αλλάζουν με την πάροδο του χρόνου. Για να διαμορφώσουμε ένα τέτοιο σύστημα πρέπει πρώτα να δημιουργήσουμε μια μεταβλητή της οποίας η παράγωγος σε σχέση με τον χρόνο είναι ίση με 1.

```
t'= 1
```

Σε αυτό το παράδειγμα δημιουργήσαμε μια συνεχή εξίσωση. Στη γλώσσα του Acumen η δημιουργία συνεχών εξισώσεων γίνετε με τη χρήση του τελεστή ισότητας (=). Στη συνέχεια, πρέπει να αλλάξουμε την κατάσταση του συστήματός μας. Ας υποθέσουμε ότι μια μεταβλητή με το όνομα *iterator* χρησιμοποιείται για να μετρήσει πόσες φορές επαναφέρουμε το χρονόμετρο στο 0 εάν περάσουν 2 δευτερόλεπτα. Για να το κάνουμε αυτό πρέπει να υλοποιήσουμε δύο διακριτά συμβάντα, ένα για να αυξήσουμε τη μεταβλητή *iterator* και ένα για να επαναφέρουμε την τιμή του χρονομέτρου.

```
if t> 2 then
  iterator + = iterator + 1,
  t + = 0
noelse
```

Παρατηρήστε ότι με τη χρήση του τελεστή διακριτής εξίσωσης (+=) το μοντέλο αντιλαμβάνεται ότι η επόμενη τιμή του *iterator* ισούται με την τρέχουσα τιμή συν 1. Επιπλέον, η επόμενη τιμή του χρονοδιακόπτη t είναι μηδέν.

2.3.2 Συλλογές Δεδομένων

Στο Acumen και για λόγους απλότητας, δεν υπάρχει θεμελιώδης διαφορά στη δημιουργία των διαφόρων τύπων δομών δεδομένων. Για να καλύψουμε τόσο την περίπτωση της θεωρητικής δομής (καρτεσιανό γινόμενο) όσο και της αναπαράστασής της στη μνήμη (δομές δεδομένων), χρησιμοποιούμε τον όρο *συλλογές δεδομένων* στο υπόλοιπο του κειμένου αυτής της πτυχιακής. Αναλυτικότερα, δεν υπάρχει περιορισμός στην ύπαρξη διαφορετικών τύπων μεταβλητών στην ίδια συλλογή. Αυτό σημαίνει ότι ο προγραμματιστής μπορεί να εκμεταλλευτεί το γεγονός, ότι για παράδειγμα η ίδια σύνταξη χρησιμοποιείται τόσο για λίστες όσο και για πλειάδες (tuples). Στο Acumen υπάρχουν διάφοροι τρόποι δημιουργίας συλλογών δεδομένων. Για παράδειγμα η δημιουργία απλών λιστών και πινάκων ακολουθεί την παρακάτω σύνταξη:

```
x = (1, "node", -9.8)    // Απλή λίστα
x = ((1, 2),             // Απλός πίνακας
     (3, 4))
```

Μια λίστα μπορεί επίσης να δημιουργηθεί χρησιμοποιώντας ένα εύρος τιμών. Η δημιουργία εύρους γίνετε χρησιμοποιώντας τη σύνταξη *αρχή:τέλος* ή *αρχή:βήμα:τέλος*. Για παράδειγμα:

```
x = (0:2:10)
```

Υπάρχουν επιπρόσθετοι τρόποι αρχικοποίησης λιστών οποιασδήποτε διάστασης, που διατίθενται μέσω της χρήσης των λέξεων-κλειδιών *ones* και *zeros*:

```
x = ones(5,2)      // Λίστα δύο διαστάσεων
y = zeros(5,2,2)   // Λίστα τριών διαστάσεων
```

Οι υποστηριζόμενες λειτουργίες στις συλλογές δεδομένων περιλαμβάνουν αριθμητικούς τελεστές (+, -, *), την αντιστροφή (*inv*), την αναστροφή (*trans*) και την ορίζουσα (*det*). Η λέξη-κλειδί *length* μπορεί να χρησιμοποιηθεί για την εύρεση του μήκους οποιασδήποτε συλλογής ενώ όταν μια συλλογή περιγράφεται από περισσότερες διαστάσεις, η ίδια εντολή μπορεί να χρησιμοποιηθεί για να βρεθεί το μήκος οποιασδήποτε διάστασης.

Επιπρόσθετα, η εύρεση και η επιστροφή ενός συγκεκριμένου στοιχείου μιας συλλογής, επιτυγχάνεται με την λειτουργία αναζήτησης. Αυτή γίνεται μέσω της χρήσης απλών παρενθέσεων μετά το όνομα της συλλογής. Για παράδειγμα, η σύνταξη για την επιστροφή ενός στοιχείου που βρίσκεται στη δεύτερη σειρά και τρίτη στήλη ενός πίνακα είναι ως εξής: $x(2,3)$. Επίσης, υπάρχει ο τεμαχισμός πινάκων slicing ως επιλογή στη γλώσσα. Ένας υποπίνακας μπορεί να εξαχθεί από έναν υπάρχοντα πίνακα όπως φαίνεται στο παρακάτω παράδειγμα:

```
model Main(simulator) =
initially
  I3 = ((1 ,0 ,0),
        (0 ,1 ,0),
        (0 ,0 ,1)),
  I2 = ((0 ,0) ,
        (0 ,0))
always
  I2 = I3 (0:1 ,0:1) // Πρώτες δύο γραμμές και στήλες του I3
```

2.3.3 Έλεγχος Ροής

Οι δηλώσεις υπό συνθήκες μπορούν να χρησιμοποιηθούν στο Acumen για να αλλάξουν τη συμπεριφορά ενός συστήματος, συχνά χρησιμοποιώντας τόσο διακριτά όσο και συνεχή γεγονότα. Αυτή είναι κοινή πρακτική για υβριδικά συστήματα. Υπάρχουν δύο τύποι τέτοιων δηλώσεων στο Acumen. Οι δηλώσεις ¨if ... else¨ και ¨match ... with¨.

2.3.3.1 Συνθήκη Διακλάδωσης If

Οι δηλώσεις if χρησιμοποιούνται για τον καθορισμό τμημάτων κώδικα που ενεργοποιούνται υπό διαφορετικές συνθήκες. Η σύνταξη για αυτόν τον τύπο δηλώσεων ορίζεται ως εξής:

```
if t <= 5 then
  x' = 1
else
  x'' = -1
```

Σε αυτό το παράδειγμα, όσο η μεταβλητή t είναι μικρότερη ή ίση με 5 τότε η πρώτη συνεχής ανάθεση βρίσκεται σε ισχύ. Το αποτέλεσμα είναι ότι η μεταβλητή x αυξάνεται σε σχέση με το χρόνο. Επειδή η μεταβλητή t επίσης αυξάνεται συνεχώς, η πάροδος 5 δευτερολέπτων επιτρέπει την ενεργοποίηση της δεύτερης εξίσωσης. Ως αποτέλεσμα, η μεταβλητή x' μειώνεται.

Ενσωματώνοντας πολλές αναθέσεις εντός παρενθέσεων διαχωρισμένες με κόμματα, επιτρέπει την εισαγωγή πολλών δηλώσεων μέσα στη δήλωση *else*. Θα πρέπει επίσης να σημειωθεί ότι όταν δεν χρειάζεται εναλλακτική ενέργεια, μπορεί να χρησιμοποιηθεί η λέξη-κλειδί *noelse*.

2.3.3.2 Συνθήκη Διακλάδωσης Match

Η δήλωση match είναι ο δεύτερος τύπος δήλωσης υπό συνθήκες που υποστηρίζεται από το Acumen και αποτελεί μία γενίκευση της δήλωσης if. Ο τρόπος που λειτουργεί είναι ότι ταιριάζει μια συγκεκριμένη έκφραση με πολλαπλές διαφορετικές περιπτώσεις. Το παρακάτω παράδειγμα εξηγεί την κατασκευή δήλωσης match:

```
match direction with [
  "left"  -> x += x - 1
| "up"    -> y += y + 1
| "right" -> x += x + 1
| "down"  -> y += y - 1
]
```

Η αντιστοίχιση απαιτεί μια σταθερή τιμή, και στο προηγούμενο παράδειγμα η μεταβλητή *direction* μπορεί να αναπαριστά μόνο τέσσερις τιμές. Σημειώστε ότι μόνο μία περίπτωση μπορεί να είναι ενεργή ανά πάσα στιγμή και μόνο η πρώτη περίπτωση θα ληφθεί υπόψη, εάν υπάρχουν περισσότερες που αντιπροσωπεύουν την ίδια τιμή.

2.3.4 Βρόγχοι Επανάληψης

Η γλώσσα επιτρέπει την επανάληψη σε συλλογές δεδομένων ή εύρη τιμών χρησιμοποιώντας τη λέξη-κλειδί *foreach*. Η επανάληψη μπορεί να γίνει με τη χρήση ενός εύρους τιμών όπως εξηγείται στην ενότητα 2.3.2. Η σύνταξη αυτής της επαναληπτικής δήλωσης απεικονίζεται στα ακόλουθα παραδείγματα:

```
foreach i in 0:2:10 do x = y * i
```

και

```
foreach i in 0:(length(v)-1) do x = y * v(i)
```

Το δεύτερο παράδειγμα απεικονίζει τον τρόπο επανάληψης σε μια συλλογή δεδομένων. Στον προγραμματισμό, απαιτείται αρκετά συχνά η επανάληψη σε μία συλλογή από τιμές όπως μια λίστα. Αυτό γίνεται καλύτερα δυναμικά χρησιμοποιώντας τη συνάρτηση *λενγτη* όπως αυτή εξηγείται στην ενότητα 2.3.2. Μια άλλη χρήση επαναληπτικών δηλώσεων είναι η άθροιση σειρών. Η σύνταξη της απεικονίζεται στο ακόλουθο παράδειγμα:

```
x += sum i*i for i in 1:10 if i%2 == 0
```

Αυτό το παράδειγμα δείχνει πώς χρησιμοποιείται η δήλωση *sum* για την εύρεση του αθροίσματος ενός εύρους τιμών, λαμβάνοντας υπόψιν την αξιολόγηση μιας κατάστασης. Αξίζει επίσης να σημειωθεί ότι η δήλωση *if* είναι προαιρετική και μπορεί να παραληφθεί.

2.3.5 Δημιουργία Μοντέλων

Ένα εκτελέσιμο πρόγραμμα Acumen μπορεί να περιέχει πολλαπλές δηλώσεις μοντέλων. Θα αναφερόμαστε σε ένα τέτοιο σύνθετο πρόγραμμα με τον όρο '*πλήρες μοντέλο*'. Προκειμένου να γίνει ένα πλήρες μοντέλο εκτελέσιμο στο περιβάλλον Acumen, απαιτείται η δήλωση ενός μοντέλου με όνομα ¨*Main*¨. Η δήλωση του μοντέλου *Main* πρέπει να έχει ακριβώς μία παράμετρο, που είναι προκαθορισμένη με το όνομα simulator. Σε γενικές γραμμές οι ορισμοί μοντέλων αποτελούνται από το όνομα και από καμία ή περισσότερες παραμέτρους ακολουθούμενες από το σύμβολο της ισότητας ('='). Σε κάθε μοντέλο μπορούν να συνυπάρχουν δύο ειδικά τμήματα, το τμήμα *initially* και το τμήμα *always*. Η χρήση του δεύτερου τμήματος είναι προαιρετική. Για παράδειγμα ένα απλό μοντέλο έχει την ακόλουθη μορφή:

```
model Node (pos_x, pos_y) =
initially
  x = pos_x, y = pos_y
always
  // Υπόλοιπο κείμενο δηλώσεων
```

Το παραπάνω παράδειγμα δείχνει τη δήλωση ενός μοντέλου και πώς διαιρείται στους δύο τομείς. Στο τομέα *initially*, εκχωρούνται αρχικές τιμές σε όλες τις μεταβλητές που είναι τοπικά ορισμένες στο μοντέλο. Οι παράμετροι του μοντέλου μπορούν επίσης να χρησιμοποιηθούν ως ορισμοί, ενώ οι μεταβλητές που δημιουργούνται σε αυτήν την ενότητα μπορούν να αναφερθούν μόνο από τον τομέα *always*. Ο τομέας *always* περιγράφει τη συμπεριφορά του συστήματος κατά την πάροδο του χρόνου. Μια πολύ σημαντική πτυχή αυτού του τομέα είναι ότι **όλες οι δηλώσεις εκτελούνται ταυτόχρονα, και η σειρά με την οποία εμφανίζονται δεν έχει σημασία**.

Η δημιουργία στιγμιότυπου (instance)[1] ενός μοντέλου είναι δυνατή και στις δύο ενότητες. Η δημιουργία στιγμιότυπου στον τομέα *initially* θεωρείται ως *¨στατική¨*, ενώ η δημιουργία στον τομέα *always* θεωρείται ως *¨δυναμική¨*.

```
model Main (simulator) =
initially
  t = 0, t' = 0,
  b = create Node(0, 0)  // Στατικό στιγμιότυπο
always
  t' = 1,
  if t > 5 then
    create Node(1, 1),   // Δυναμικό στιγμιότυπο
    t+=0
  noelse
```

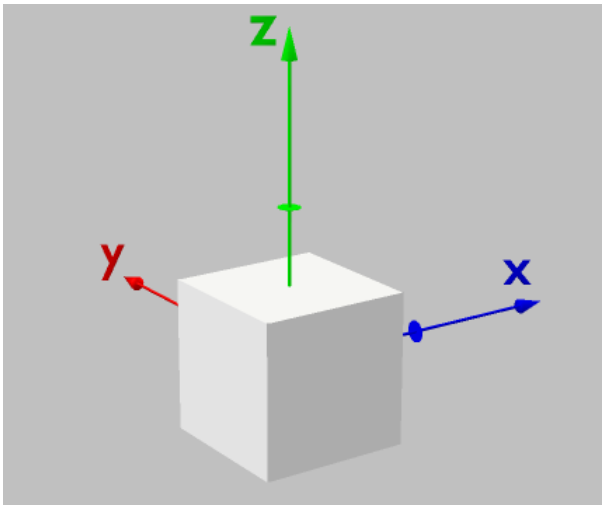
Αναφερόμαστε σε ένα μοντέλο ως *γονέα* όταν αυτό δημιουργεί νέα στιγμιότυπα μοντέλων είτε στατικά είτε δυναμικά, ενώ όλα τα στιγμιότυπα που ανήκουν σε αυτό το γονέα αναφέρονται ως *παιδιά*. Μια πολύ χρήσιμη λειτουργία είναι η επανάληψη στη συλλογή παιδιών ενός μοντέλου. Αυτή η λειτουργία επιτυγχάνεται μέσω της λειτουργίας επανάληψης:

```
foreach c in children do c.x += 15
```

[1] Η έννοια του στιγμιότυπου έχει την ίδια σημασιολογία όπως και στις αντικειμενοστραφείς γλώσσες προγραμματισμού.

2.3.6 Οπτικοποίηση και Κινούμενες Εικόνες

Το περιβάλλον χρήστη ενισχύεται περαιτέρω με την βιβλιοθήκη γραφικών jPCT [4], ικανή για την δημιουργία κινούμενων εικόνων σε πραγματικό χρόνο κατά τη διάρκεια μιας προσομοίωσης [34]. Η προσομοίωση κάποιου υβριδικού συστήματος κατά την πάροδο του χρόνου είναι ένας αντιπροσωπευτικός τρόπος για να κατανοήσουμε καλύτερα τη συμπεριφορά του μέσα από ένα ελκυστικό οπτικό αποτέλεσμα που είναι αλληλεπιδράσιμο.



Σχήμα 1: Οπτικοποίηση ενός απλού κουτιού από το Acumen

Στο Acumen, πρέπει να χρησιμοποιηθεί η ειδική μεταβλητή **_3D** για τη δημιουργία τρισδιάστατων αντικειμένων. Η βιβλιοθήκη γραφικών παρέχει μια συλλογή από κοινά σχήματα (π.χ. Sphere, Cylinder, Box, κ.α.) όπως και αντικείμενα κειμένου (Text). Για την διευκόλυνση του χρήστη, τα αντικείμενα περιέχουν προκαθορισμένες τιμές. Για παράδειγμα για τη δημιουργία της σκηνής που απεικονίζεται στο Σχήμα 1 χρησιμοποιείται η ακόλουθη σύνταξη:

```
_3D = (Box                  // Όνομα αντικειμένου
       size = (1,1,1))      // Μέγεθος στους άξονες x,y,z
```

Η σύνταξη σε Acumen παρέχει επίσης υποστήριξη για φόρτωση τρισδιάστατων πλεγμάτων (3D meshes) αποθηκευμένα ως αρχεία **OBJ**[1]. Η δημιουργία τέτοιων τρισδιάστατων αντικειμένων γίνεται με τη χρήση του προσαρμοσμένου αντικειμένου (Obj) της βιβλιοθήκης γραφικών. Για παράδειγμα:

[1]Η διεύθυνση: `https://en.wikipedia.org/wiki/Wavefront_.obj_file` περιέχει περισσότερες πληροφορίες σχετικά με αυτό τον τύπο αρχείου.

```
_3D = (Obj
       content =  "Car.obj"  // Όνομα αρχείου OBJ
```

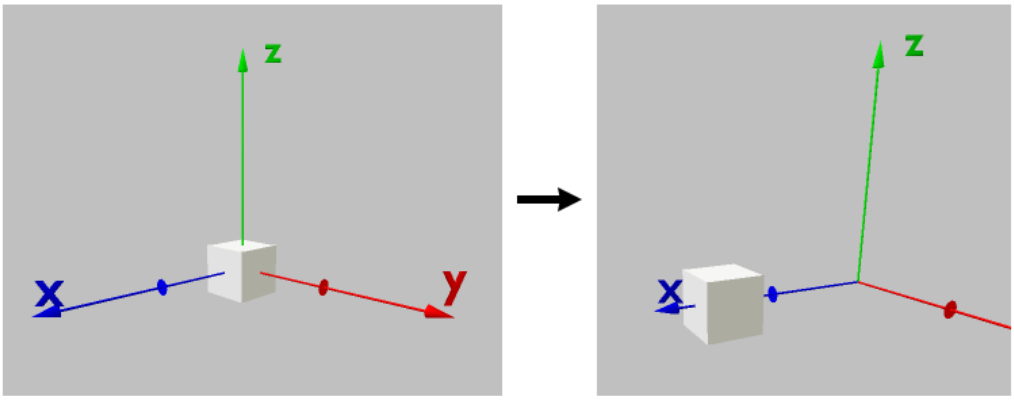
Η επιθυμία ύπαρξης κινουμένων εικόνων σε μια προσομοίωση υποδηλώνει την ύπαρξη δυναμικών τρισδιάστατων μεταβλητών. Στην πραγματικότητα η απεικόνιση της συμπεριφοράς ενός μοντέλου γίνεται με τη συνεχή ανάθεση τιμών στις δυναμικές μεταβλητές _3D στον τομέα *always* αυτού του μοντέλου. Ωστόσο, αυτό προϋποθέτει ότι το ειδικό πεδίο _3D έχει ήδη οριστεί στο τομέα *initially*. Στη συνέχεια, η απεικόνιση θα πραγματοποιηθεί κατά την διάρκεια της προσομοίωσης του μοντέλου. Το παρακάτω παράδειγμα δείχνει πώς επιτυγχάνεται η μετακίνηση του αντικειμένου που απεικονίζεται στο Σχήμα 1, κατά το μήκος του άξονα x.

```
model Main (simulator) =
initially
  x = 0, x' = 0,
  _3D = ()
always
  x' = 1,
  _3D = (Box size = (1,1,1) center = (x, 0, 0))
```

Η τρισδιάστατη σκηνή του Acumen παρέχει τη δυνατότητα χειρισμού της κάμερας. Κατά τη διάρκεια μιας προσομοίωσης το πάτημα του αριστερού πλήκτρου του ποντικιού περιστρέφει τη σκηνή 3D, το δεξί πλήκτρο μετακινεί το κέντρο της προβολής και η χρήση του τροχού του ποντικιού επιτυγχάνει τη λειτουργία μεγένθυσης. Ο χειροκίνητος έλεγχος της σκηνής είναι επίσης δυνατός απευθείας μέσα σε ένα μοντέλο. Η λέξη-κλειδί **_3DView** μπορεί να χρησιμοποιηθεί για την αλλαγή δύο παραμέτρων, του σημείου της κάμερας και του σημείου εστίασης. Ομοίως, και ανάλογα σε ποιό τομέα του μοντέλου θα οριστεί αυτή η μεταβλητή, μπορεί να υλοποιηθεί στατική ή δυναμική συμπεριφορά της κάμερας. Ο κώδικας για ένα πολύ ενδιαφέρον μοντέλο με δυναμικό χειρισμό κάμερας είναι ο ακόλουθος:

```
model Main (simulator) =
initially
  x=0, x'=1,
  _3D = (), _3DView = ()
always
  x'=1,
  _3D = (Box size = (0.5,0.5,0.5)
         color = white
         center = (x,0,0)),
  _3DView = ((5,5,2),(x,0,0))
```

Σε αυτό το παράδειγμα οι συντεταγμένες της κάμερας είναι (5,5,2) ενώ η θέση εστίασης πάνω στον άξονα x αυξάνεται δυναμικά με την πάροδο του χρόνου. Παρατηρήστε ότι το αντικείμενο 3D που εμπεριέχεται σε αυτήν τη σκηνή, το οποίο είναι ένα λευκό κουτί, κινείται επίσης κατά μήκος του άξονα x. Το κινούμενο σχέδιο που προκύπτει είναι ότι η κάμερα ακολουθεί το αντικείμενο καθ' όλη τη διάρκεια της προσομοίωσης. Αυτή η διαμόρφωση αποδίδει τη σκηνή που απεικονίζεται στο Σχήμα 2. Η εικόνα στα αριστερά απεικονίζει την κατάσταση της σκηνής στην αρχή της προσομοίωσης ενώ στα δεξιά είναι η ίδια σκηνή μετά από 1.5 δευτερόλεπτα.



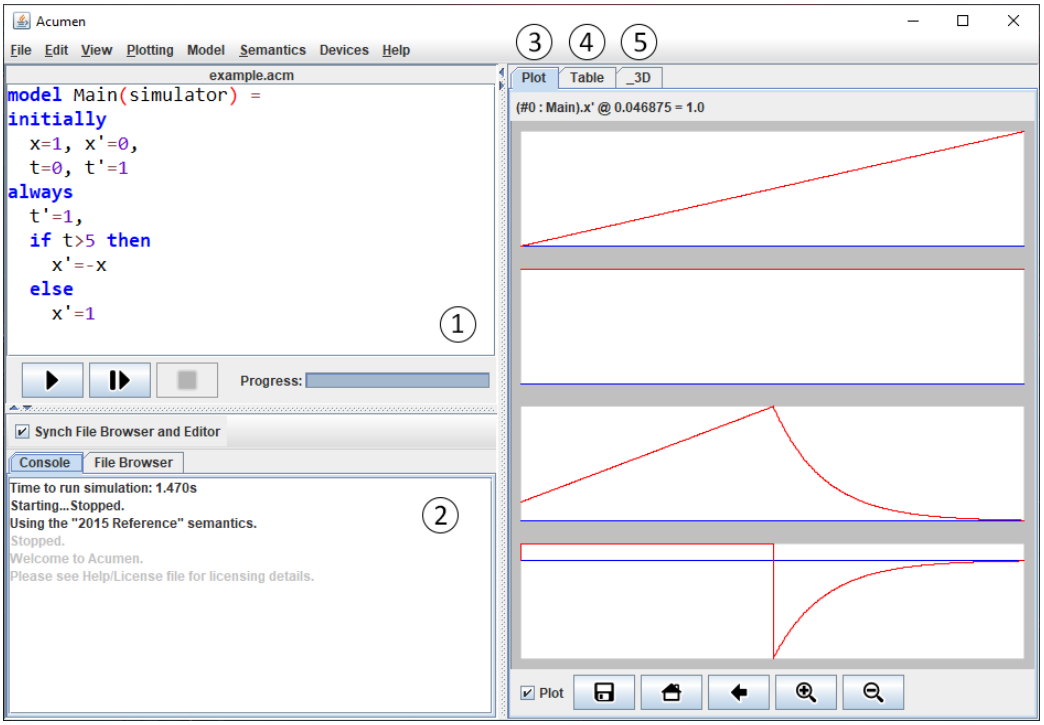
Σχήμα 2: Σκηνή με δυναμικό σημείο προβολής κάμερας

Τέλος, υπάρχει επίσης η δυνατότητα ¨προσάρτησης¨ αντικειμένων στην προοπτική του θεατή. Για παράδειγμα, το ακόλουθο αντικείμενο έχει μια πρόσθετη παράμετρο *coordinates*, η οποία κάνει τέτοια αντικείμενα να φαίνεται στατικά, ακόμα και αν υπάρχει κίνηση της κάμερας οποιουδήποτε είδους.

```
_3D = (Text center = (0,0,0)
       coordinates = "camera"
       content = "Machine Learning")
```

Η προσομοίωση ενός μοντέλου που περιέχει αυτό το αντικείμενο, διασφαλίζει ότι το αντικείμενο είναι στατικό ακριβώς στο κέντρο της σκηνής, κοιτώντας τον θεατή. Οποιεσδήποτε κινήσεις, περιστροφές, μεγενθύνσεις και κάθε είδους ρυθμίσεις κάμερας δεν τροποποιούν το αντικείμενο με κανένα τρόπο. Η παρουσίαση αυτού του μοντέλου είναι δύσκολο να γίνει μέσα από κάποιο σχήμα, για αυτό προτρέπουμε τον αναγνώστη να εκτελέσει την προσομοίωση του μοντέλου για την κατανόηση αυτής της στατικής συμπεριφοράς.

2.4 Γραφικό Περιβάλλον Χρήστη



Σχήμα 3: Το Γραφικό Περιβάλλον Χρήστη του Acumen

Το IDE του Acumen συνδυάζεται με ένα διαισθητικό GUI [25] όπως φαίνεται στο Σχήμα 3. Είναι μια εφαρμογή που δεν χρειάζεται εγκατάσταση, τρέχει πάνω στο Java Runtime Environment - Περιβάλλον εκτέλεσης Java (JRE)[1] [10] και είναι οργανωμένο σε δύο τμήματα. Στο αριστερό τμήμα υπάρχει ο επεξεργαστής κώδικα, υλοποιημένος με τη βιβλιοθήκη RSyntaxTextArea [7]. Επίσης υπάρχουν δύο καρτέλες (tabs) οι οποίες υλοποιούν διαφορετικές όψεις ανάλογα την επιλογή (2). Η πρώτη περιέχει την τερματική κονσόλα (Console) ενώ η δεύτερη την όψη περιήγησης αρχείων (File Browser). Ο επεξεργαστής κώδικα παρέχει τα μέσα για τη δημιουργία και την τροποποίηση μοντέλων, ενώ η παραγωγικότητα του επεξεργαστή ενισχύεται με την υποστήριξη επισήμανσης σύνταξης, την αυτόματη συμπλήρωση κώδικα και τις ενδείξεις σε γραμμές όπου παρουσιάζονται σφάλματα κατά τη διάρκεια της προσομοίωσης (1). Η τερματική κονσόλα είναι υπεύθυνη για την εμφάνιση σφαλμάτων και ενημερωτικών μηνυμάτων.

[1] Για την εκτέλεση της εφαρμογής του Acumen απαιτείται η εγκατάσταση του περιβάλλοντος εκτέλεσης Java (JRE) και συγκεκριμένα η έκδοση 8.

Στο δεξί τμήμα του GUI υπάρχουν τρεις διαφορετικές καρτέλες για οπτικοποίηση δεδομένων. Η καρτέλα σχεδιασμού διαγραμμάτων (Plot) εμφανίζει τον τρόπο με τον οποίο εξελίσσεται η προσομοίωση των μεταβλητών με την πάροδο του χρόνου, κάθε μια στο δικό της διάγραμμα (3). Η καρτέλα πίνακας (Table) εμφανίζει κάθε μεταβλητή που υπάρχει στο μοντέλο (ως στήλες), και τις τιμές τους σε κάθε βήμα της προσομοίωσης (4). Τέλος, η καρτέλα _3D [34, 33] εμφανίζει τρισδιάστατες σκηνές και κινούμενες εικόνες που περιέχουν 3D αντικείμενα ορισμένα στα μοντέλα (5).

Είναι σημαντικό να αναφέρουμε ότι η εκτέλεση του Acumen μπορεί να πραγματοποιηθεί επίσης μέσα από μία Command Line Interface - Διεπαφή Γραμμής Εντολών (CLI), η οποία δέχεται επιπλέον παραμέτρους. Στα πλαίσια αυτής της πτυχιακής δεν γίνετε χρήση αυτού του τρόπου εκκίνησης του περιβάλλοντος Acumen. Γενικά, η χρήση της CLI προορίζεται για την εκτέλεση προσομοίωσης σε ένα τερματικό παράθυρο ή από ένα script.

3 Δημιουργία Προσομοιώσεων

Το Acumen, ως μία διερμηνευόμενη γλώσσα, χρειάζεται να μετατρέψει τους ορισμούς των μοντέλων από δεδομένα κειμένου σε αυστηρές προσομοιώσεις. Υποστηρίζει διάφορους επιλυτές διαφορικών εξισώσεων που χρησιμοποιούνται για την προσομοίωση, με βάση τη επιλεγμένη σημασιολογία. Σε αυτήν την πτυχιακή κάνουμε χρήση της επιλογής *Reference 2015* από τον κατάλογο σημασιολογίας που απλώς εφαρμόζει την μέθοδο Runge-Kutta τετάρτου βαθμού και ένα σταθερό χρονικό βήμα για την ολοκλήρωση. Η ακόλουθη ενότητα εξηγεί πώς εφαρμόζεται η συντακτική ανάλυση στο πλήρες μοντέλο για τη δημιουργία ενός Αφηρημένου Συντακτικού Δέντρου (AST) και με ποιους τρόπους αυτό τροποποιείται περαιτέρω πριν από την ερμηνεία. Επιπλέον, εξηγούμε πώς λειτουργεί ο βρόχος προσομοίωσης του Acumen και πώς ο διερμηνέας διασχίζει το AST για να εκτελέσει την προσομοίωση.

3.1 Συντακτική Ανάλυση

Αφού καθοριστεί το πλήρες μοντέλο από τον χρήστη, πρέπει να αναλυθεί και να επεξεργαστεί. Η ανάλυση είναι ένα απαραίτητο πρώτο βήμα για τη δημιουργία οποιασδήποτε προσομοίωσης, καθώς μετατρέπει το κείμενο σε μία δομή δεδομένων, συνήθως ένα AST, το οποίο αργότερα ερμηνεύεται από τη γλώσσα. Η μέθοδος ανάλυσης του Acumen ήταν δυνατή χρησιμοποιώντας τη βιβλιοθήκη Scala Parser Combinators [19]. Η ανάλυση γίνεται χρησιμοποιώντας συναρτήσεις ανάλυσης που συμπεριφέρονται παρόμοια με τις κανονικές εκφράσεις, αναγνωρίζοντας συγκεκριμένες λέξεις ή φράσεις και μετατρέποντας τις σε συγκεκριμένους τύπους δεδομένων, δημιουργώντας τελικά ένα AST.

Με την επιτυχή συντακτική ανάλυση του πλήρους μοντέλου, απαιτείται η ανάλυση δύο επί πλέον διαμορφώσεων. Αυτά τα πρόσθετα βήματα ανάλυσης αφορούν τους κρυφούς ορισμούς των μοντέλων με ονόματα *Device* και *Parameters*. Το μοντέλο Device χρησιμοποιείται για ρεαλιστική προσομοίωση [33] απομακρυσμένων συσκευών από τη βιβλιοθήκη απόδοσης 3D του Acumen. Ο κώδικας αρχικοποίησης που αναλύεται και χρησιμοποιείται για την περιγραφή του μοντέλου Device είναι ως εξής:

```
model Device(id) =
initially
  ax=0, ay=0, az=0,
  alpha=0, beta=0, gamma=0,
  compassHeading=0
```

Η δεύτερη διαμόρφωση αφορά το μοντέλο Parameters και περιέχει μεταβλητές που εισάγονται από τη CLI. Εάν το Acumen εκτελεστεί με το GUI, το μοντέλο Parameters αρχικοποιείται χωρίς καμία παράμετρο. Ο ορισμός του μοντέλου φαίνεται παρακάτω:

```
model Parameters()=
```

Τα αποτελέσματα από την ανάλυση των μοντέλων Parameters και Device, καθώς και την ανάλυση των ορισμών που υπάρχουν στο πλήρες μοντέλο, στη συνέχεια εισάγονται σε μια λίστα. Η λίστα που προκύπτει, η οποία τώρα αποτελείται από τρεις ορισμούς μοντέλων, αντιπροσωπεύει το παραγόμενο AST που χρησιμοποιείται κατά την διάρκεια της ερμηνείας (interpretation phase).

3.2 Επί πλέον Περάσματα

Πριν από το βήμα της ερμηνείας, το AST πρέπει να τροποποιηθεί. Αυτό επιτυγχάνεται μέσω της χρήσης ειδικών μεθόδων που ονομάζονται περάσματα (passes). Ο αριθμός και ο τύπος των περασμάτων διαφέρει ανάλογα με την επιλεγμένη σημασιολογία του Acumen, ενώ η σειρά με την οποία εφαρμόζονται είναι αυστηρά προκαθορισμένη. Οποιασδήποτε επιλογή από το μενού παραδοσιακής σημασιολογίας (*traditional*) 2015 εκτελεί την εφαρμογή δύο περασμάτων με την ακόλουθη σειρά:

1. Binding Time Analysis - Ανάλυση Χρόνου Δέσμευσης (BTA) [35]
2. Desugarer (Local Inlining)

3.2.1 Ανάλυση Χρόνου Δέσμευσης

Η BTA αποτελεί το πρώτο πέρασμα κατά την τροποποίηση του AST. Αποτελεί μία σημαντική στατική ανάλυση που απαιτείται από *μερικούς αξιολογητές* (partial evaluators). Σε ένα πρόγραμμα, η BTA αξιολογεί ποια μέρη του μπορούν να εκτελεστούν κατά τη διάρκεια της μεταγλώττισης και ποιά κατά τη διάρκεια της προσομοίωσης, επισυνάπτοντας τα με μία από δύο πιθανές ετικέτες που ονομάζονται ¨δεσμευτικοί χρόνοι¨. Το πρώτο μέρος επισημαίνεται ως ¨στατικό¨ και χρησιμοποιείται για εκτελέσεις κατά την διάρκεια της μεταγλώττισης ενώ το δεύτερο, που επισημαίνεται ως ¨δυναμικό¨, κατά την διάρκεια της προσομοίωσης. Στα πλαίσια του Acumen, η υλοποίηση της BTA υποστηρίζει τη χρήση μερικών παραγώγων, η οποία ενισχύει την εκφραστικότητα της γλώσσας. Ένα χρήσιμο αποτέλεσμα αυτής της προσθήκης είναι η ικανότητα έκφρασης της εξίσωσης Euler-Lagrange [35, 36].

Μαθηματική Σύνταξη	**Acumen Σύνταξη**
m=5 g=9.81 l=3 $I = m * l^2$	m=5, g=9.81, l=3, I=m*l ^ 2
$T = \frac{1}{2} * I\dot{\theta}^2$	T=1/2*I*theta'^2,
$V = m * g * l * (1 - \cos(\theta))$	V=m*g*l*(1-cos(theta)),
$L = T - V$	L = T - V
$\frac{\partial}{\partial t}(\frac{\partial L}{\partial \dot{\theta}})$ - $\frac{\partial L}{\partial \theta}$ = 0	L'[theta']' - L'[theta] = 0

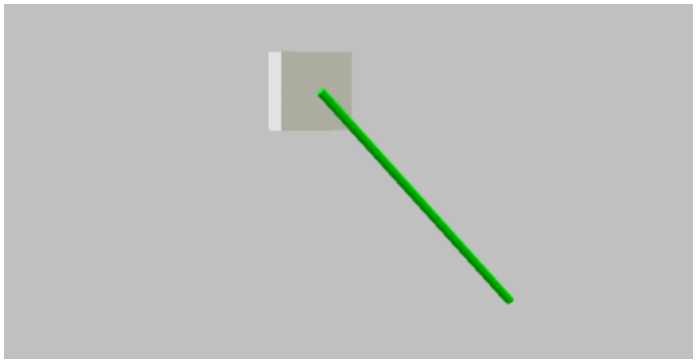
Πίνακας 1: Διαμόρφωση εκκρεμούς Lagrange

Ο Πίνακας 1 δείχνει τη μαθηματική σύνταξη για μερικούς παραγώγους συγκριτικά με την ίδια σύνταξη στο Acumen, κατά την μοντελοποίηση ενός απλού εκκρεμούς. Εδώ, η συνάρτηση *Lagrange* L είναι ίση με την κινητική ενέργεια T μείον την δυναμική ενέργεια V. Η δήλωση L'[theta] είναι η σύνταξη σε Acumen για το $\frac{\partial L}{\partial \theta}$ και η δήλωση L'[theta']' για το $\frac{\partial}{\partial t}(\frac{\partial L}{\partial \dot{\theta}})$. Ένα πλήρες παράδειγμα[1] προσομοίωσης ενός απλού εκκρεμούς στο Acumen μπορεί να υλοποιηθεί ως εξής:

```
model Main(simulator)=
initially
  theta = 3*pi/2, theta' = 0, theta'' = 0,
  L = 0, T = 0, V = 0, I = 0,
  m = 5, l = 3, g = 0,
  _3D = (), _3DView=()
always
  g = 9.8, m=5, l=2, I = m*l^2,
  T = (1/2)*I*theta'^2,
  V = m*g*l*(1-cos(theta)),
  L = T - V,
  L'[theta']' - L'[theta]=0,
  _3DView= ((5,10,0),(0,0,0)),
  _3D = (Cylinder radius = 0.03 length=l
    center=(l*sin(theta)/2,0,-l*cos(theta)/2)
    rotation = (-pi/2,-theta,0) color = green,
    Box size=(0.5,0.2,0.5) center=(0,-0.15,0)
    color=white)
```

[1]Αυτό το απόσπασμα κώδικα καθώς και επιπρόσθετα παραδείγματα που κάνουν χρήση της BTA βρίσκονται στο φάκελο examples/04_experimental/04_BTA.

Το γραφικό αποτέλεσμα κατά τη διάρκεια προσομοίωσης αυτού του μοντέλου απεικονίζεται στο Σχήμα 4 παρακάτω:



Σχήμα 4: Οπτικοποίηση ενός απλού εκκρεμούς

Ο μετασχηματισμός μερικών παραγώγων και εξισώσεων σε μια γλώσσα που υποστηρίζει ODE, είναι μια διαδικασία δύο βημάτων [35, 36]. Το πρώτο βήμα είναι η εφαρμογή της BTA όπως εξηγείται στην αρχή αυτής της ενότητας, ενώ το δεύτερο βήμα ονομάζεται εξειδίκευση (specialization). Όταν η εφαρμογή της BTA ολοκληρωθεί με επιτυχία και εφαρμοστούν ετικέτες δέσμευσης στο AST, το βήμα εξειδίκευσης αναλαμβάνει και είναι υπεύθυνο για την ορθή εκτέλεση των οδηγιών από την BTA. Κατά τη διάρκεια της εξειδίκευσης, η Symbolic Differentiation - Συμβολική Παραγώγιση (SD) εξαλείφει ολικούς και μερικούς παραγώγους χρησιμοποιώντας τον κανόνα της αλυσίδας. Στη συνέχεια το AST, το οποίο πλέον αποτελείται μόνο από υπονοούμενες (implicit) Discrete-Algebraic Equations - Διακριτές-Αλγεβρικές Εξισώσεις (DAE), περνάει μέσω του αλγεβρικού επιλυτή του Acumen. Αυτός ο επιλυτής είναι σε θέση να επιλύει ταυτόχρονα πολλαπλές implicit μορφής ODE σε ρητής (explicit) μορφής εξισώσεων, μια διαδικασία που επιτυγχάνεται χρησιμοποιώντας συμβολική Gaussian Elimination - Απαλοιφή Gauss (GE). Συνοψίζοντας, η εξειδίκευση μετατρέπει τη μορφή του AST σε εξειδικευμένη χωρίς ετικέτες δέσμευσης χρόνου και απαρτιζόμενο μόνο από ODE.

3.2.2 Desugarer (Local Inlining)

Το *Desugarer (Local Inlining)* αποτελεί το δεύτερο και τελευταίο πέρασμα πάνω στο AST. Ο κύριος σκοπός του είναι να μετασχηματίσει εκφράσεις για να απλοποιήσει το AST σε ένα σύστημα εξισώσεων. Αναλυτικότερα, το AST περιέχει μια συλλογή ενεργειών. Μπορεί να συνυπάρχουν επτά διαφορετικά είδη ενεργειών και είναι τα εξής: IfThenElse, Switch, ForEach, Claim, Hypothesis, continuous και discrete. Αυτές οι ενέργειες είναι σύνθετες κα-

τασκευές που περιέχουν ένα ή περισσότερα από τα ακόλουθα: εκφράσεις, πρόσθετες λίστες ενεργειών και περιπτώσεις (clauses) της συνθήκης match (όπως αυτή παρουσιάστηκε στην Ενότητα 2.3.3.2).

Οι εκφράσεις είναι συντακτικές οντότητες που μπορούν να αποτελούνται από ένα ή περισσότερα από τα ακόλουθα: λειτουργίες (operations), δείκτες (indeces), κλήσεις σε άλλες παρουσίες μοντέλου (calls), μεταβλητές (variables) και τιμές (literals). Οι ειδικές κατασκευές clauses περιγράφουν τις περιπτώσεις που εμφανίζονται στις συνθήκες match. Αποτελούνται από μία τιμή, ένα προαιρετικό ισχυρισμό και μια λίστα ενεργειών. Το ακόλουθο παράδειγμα εξηγεί τους διαφορετικούς όρους, οι οποίοι μπορούν να αναγνωριστούν με το ακόλουθο συνδυασμό χρωμάτων: κόκκινο για τις τιμές, πράσινο για ισχυρισμό και μπλε για τη λίστα ενεργειών:

```
match target with [
  "closed" claim (t<=10) ->
  target+ = "open",
  t+ = t+1
| "open" -> // Υπόλοιπο κείμενο δηλώσεων
]
```

Μια άλλη περίπτωση που ακολουθεί αυτό το μοτίβο είναι αυτή της ενέργειας IfThenElse. Εκτός της έκφρασης που περιγράφει μια συνθήκη, περιέχει επίσης δύο επιπλέον λίστες ενεργειών. Η πρώτη λίστα περιέχει δράσεις που ενεργοποιούνται όταν πληρούται η συνθήκη, ενώ η δεύτερη περιέχει αυτές που ενεργοποιούνται από τη συνθήκη else, εφόσον υπάρχει. Σε περίπτωση που χρησιμοποιείται η λέξη-κλειδί noelse, η δεύτερη λίστα είναι κενή.

```
if t>2 then
  x+ = x+1,
  t+ = t+1
else
  t'=1
```

Ομοίως, οι ενέργειες υπόθεσης περιέχουν μια έκφραση συμβολοσειράς και μια λίστα εκφράσεων για αξιολόγηση. Αυτές οι ενέργειες μπορούν να δημιουργηθούν με την ακόλουθη σύνταξη:

```
hypothesis "Gravity remains constant" (g==-9.8)
```

Οι ενέργειες discrete αντιπροσωπεύουν έναν από τους ακόλουθους τύπους: Εκχώρηση (Assign), Δημιουργία (Create), Εξάλειψη (Elim) και Μετακίνηση (Move). Ο πρώτος τύπος περιέχει μια διακριτή ανάθεση με εκφράσεις που αντιπροσωπεύουν την αριστερή και τη δεξιά πλευρά της ανάθεσης. Οι τρεις τελευταίες ενέργειες είναι οδηγίες για τα στιγμιότυπα μοντέλων και χρησιμοποιούνται για τη δημιουργία, τη διαγραφή ή τη μετακίνηση μιας παρουσίας σε ένα νέο μοντέλο-γονέα. Οι ενέργειες Create αποτελούνται από μια προαιρετική έκφραση μιας μεταβλητής, που βασίζεται στο εάν το στιγμιότυπο μοντέλου δημιουργείται στατικά ή δυναμικά, και δύο εκφράσεις για το όνομα του μοντέλου και τις παραμέτρους του. Οι ενέργειες Elim έχουν μία έκφραση για το όνομα του μοντέλου και οι ενέργειες Move περιέχουν εκφράσεις για το όνομα της instance που θα μεταφερθεί καθώς και του νέου γονέα. Παραδείγματα αυτών των ενεργειών δίνονται στις ακόλουθες γραμμές:

```
x = create Ball(1)   // Στατική παρουσία (τομέας initially)
create Ball(1)       // Δυναμική παρουσία (τομέας always)
terinate x           // Τερματισμός παρουσίας μοντέλου x
move x y             // Μετακίνηση του x στο y
```

Μία συνάρτηση του περάσματος Desugarer με το όνομα desugar αξιολογεί όλες τις ενέργειες που αναφέρθηκαν μέχρι στιγμής και τις διασχίζει αναδρομικά έως ότου βρεθεί μία ρητή τιμή (literal value) ή μία μεταβλητή. Σε περίπτωση ρητής τιμής δεν εκτελείται περαιτέρω δράση ενώ στην περίπτωση μεταβλητής, γίνεται αξιολόγησή της[1].

Οι ενέργειες continuous αντιμετωπίζονται με ελαφρώς διαφορετικό τρόπο. Οι εκφράσεις της αριστερής και της δεξιάς πλευράς αξιολογούνται όπως εξηγήθηκε παραπάνω. Εάν η αξιολογημένη αριστερή πλευρά της δράσης έχει αντιστοιχηθεί επιτυχώς σε κάποιο μοντέλο, δημιουργείται μια κατασκευή εξίσωσης τύπου **EquationT** που περιέχει την αριστερή και τη δεξιά πλευρά της συνεχής ανάθεσης. Στη συνέχεια, οι συνεχείς αναθέσεις υψηλότερης τάξης επεκτείνονται σε συστήματα συνεχών αναθέσεων πρώτης τάξης, μια διαδικασία που ενσωματώνει καθεμία από τους χαμηλότερους βαθμούς σε κατασκευές εξίσωσης τύπου **EquationI**. Διαφορετικά, εμφανίζεται ένα σφάλμα στην κονσόλα, που ενημερώνει ότι βρέθηκε κάποια απροσδόκητη εξίσωση. Για παράδειγμα, ο μετασχηματισμός της έκφρασης $x'' = 1$ γίνεται επέκταση ως:

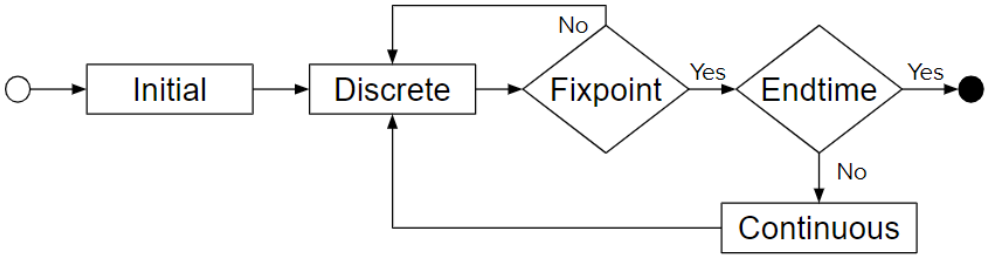
```
List(EquationT(x''), EquationI(x'), EquationI(x))
```

[1]Επειδή η κατανόηση σχετικά με τον ακριβή τρόπο λειτουργίας των αλγορίθμων που περιγράφουν το πέρασμα Desugarer είναι πολύ εξειδικευμένη τεχνικά, προτρέπουμε τον αναγνώστη να επισκεφτεί την σελίδα που στεγάζει τον κώδικα του Acumen στη διεύθυνση: https://github.com/maroneal/acumen

Για λόγους απλότητας, παρουσιάζουμε τις κατασκευές εξισώσεων μόνο σε κορυφαίο επίπεδο και χωρίς τα περιεχόμενά τους, όπως αυτές εμφανίζονται στο AST. Είναι πολύ χρήσιμο να σημειωθεί εδώ ότι οι ενέργειες continuous έχουν πλέον μετατραπεί σε λίστα. Αυτή η λίστα περιέχει τις δομές σε μια προκαθορισμένη σειρά και ο τρόπος με τον οποίο συμπληρώνεται διαμορφώνεται από τον επιλεγμένο διερμηνέα. Για οποιαδήποτε επιλεγμένη παραδοσιακή σημασιολογία του έτους 2015 αυτή η διαμόρφωση είναι προκαθορισμένη ως *local-inline*.

3.3 Διερμηνέας

Η δημιουργία προσομοιώσεων στο Acumen είναι μια διαδικασία πολλαπλών βημάτων. Ένας ελεγκτής καλεί επανειλημμένα μία συνάρτηση **step( )** έως ότου ικανοποιηθεί με επιτυχία κάποια κατάσταση τερματισμού. Κατά τη διάρκεια κάθε βήματος, η κατάσταση προσομοίωσης διατηρείται μέσα σε μία δομή τύπου map με όνομα *store*, η οποία είναι μια αφηρημένη δομή δεδομένων που περιλαμβάνεται σε κάθε διερμηνέα, μαζί με πληροφορίες σχετικά με το εάν η προσομοίωση έχει ολοκληρωθεί.



Σχήμα 5: Βρόχος προσομοίωσης του διερμηνέα Reference στο Acumen

Η προσομοίωση ενός μοντέλου Acumen απεικονίζεται στο Σχήμα 5. Οι παραδοσιακοί διερμηνείς εκτελούν διακριτά (Discrete) και συνεχή (Continuous) βήματα εναλλάξ μέχρι να βρεθεί ο τελικός χρόνος (EndTime)[1]. Αφού ολοκληρωθεί η ανάλυση, ο διερμηνέας εκτελεί το αρχικό βήμα (Initial) μία φορά για να αρχικοποιήσει το αρχικό store της προσομοίωσης χρησιμοποιώντας την κατάσταση όπως περιγράφεται στο τομέα initially του μοντέλου. Στη συνέχεια λαμβάνονται επανειλημμένα διακριτά βήματα μέχρι να βρεθεί ένα βήμα σταθερού χρόνου (Fixpoint), που έχει ως αποτέλεσμα την επεξεργασία διακριτών αναθέσεων και δομικών ενεργειών. Εάν το EndTime επιτευχθεί η προσομοίωση θεωρείται ως ολοκληρωμένη, διαφορετικά πραγματοποιείται ένα συνεχές βήμα.

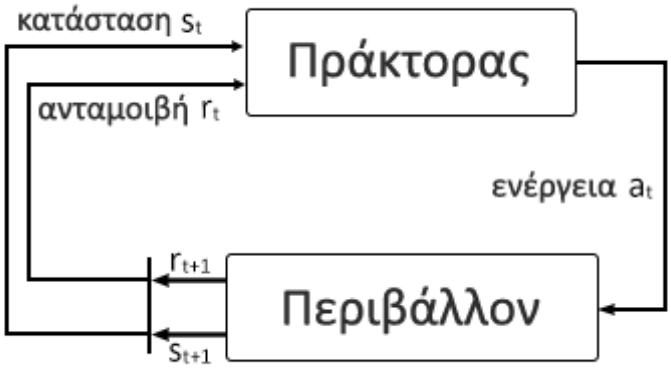
[1] Οι διερμηνείς traditional έχουν προεπιλεγμένη τιμή 10 δευτερολέπτων.

Κατά τη διάρκεια ενός συνεχούς βήματος, εκτελούνται όλες οι συνεχείς αναθέσεις και οι ολοκληρώσεις, αυξάνοντας την τιμή του χρόνου και ξεκινώντας εκ νέου το βρόχου προσομοίωσης εκτελώντας ένα νέο διακριτό βήμα. Οι αναθέσεις που ορίζονται από τις κατασκευές **EquationsT** συλλέγονται και εφαρμόζονται στο store σε κάθε βήμα, ενώ οι κατασκευές **EquationsI** συλλέγονται μόνο μετά από εύρεση βήματος σταθερού χρόνου.

4 Δημιουργία Ευφυούς Συμπεριφοράς στο Περιβάλλον Acumen

Για την ενσωμάτωση μεθόδων μηχανικής μάθησης στο Acumen με στόχο τη δυνατότητα υλοποίησης ευφυούς συμπεριφοράς (intelligent behavior) από τα μοντέλα μελετήσαμε την τεχνική της Reinforcement Learning - Μάθηση με Ενίσχυση (RL). Αυτή η ενότητα κάνει μια εισαγωγή στην RL και παρέχει τα αποτελέσματα μιας μελέτης σχετικά με το εάν το περιβάλλον Acumen μπορεί να υποστηρίξει τέτοιους τύπους αλγορίθμων. Αρχικά, εξηγούμε την λειτουργία ενός αλγορίθμου RL με όνομα *Q-Learning*. Δεύτερον, παρουσιάζουμε τους τρέχοντες περιορισμούς στη συμπεριφορά συλλογών στο Acumen, τρόπους επίλυσης αυτών των προβλημάτων καθώς και ορισμένες προσθήκες σε λειτουργίες των συλλογών δεδομένων.

4.1 Μάθηση με Ενίσχυση



Σχήμα 6: Ο βρόγχος της μάθησης με ενίσχυση

Η RL είναι μια τεχνική εκπαίδευσης της ML, κατά την οποία πράκτορες λογισμικού πειραματίζονται σε ένα άγνωστο προς εκείνους περιβάλλον με τη λήψη αποφάσεων [16]. Κατά τη διάρκεια αυτής της διαδικασίας είναι σε θέση να προσαρμοστούν και να βελτιστοποιήσουν τη συμπεριφορά τους μέσω ενός μηχανισμού ανατροφοδότησης που βασίζεται σε ανταμοιβές [29]. Αυτή η διαδικασία συνιστά ένα τύπο μάθησης που μπορεί να μοντελοποιηθεί ως κάποια Markov Decision Process - Μαρκοβιανή Διεργασία Απόφασης (MDP) η οποία αποτελείται από τα ακόλουθα:

- Ένα σύνολο καταστάσεων, S.
- Ένα σύνολο ενεργειών που επιτρέπεται να κάνει ο πράκτορας, A.
- $P_a(s, s') = Pr(s_{t+1} = s' | s_t = s, a_t = a)$ είναι η πιθανότητα μετάβασης στην κατάσταση s' από την κατάσταση s, χρησιμοποιώντας μία ενέργεια a, σε χρόνο που ισούται με t
- $R_a(s, s')$ είναι η ανταμοιβή που επιστρέφεται μετά τη μετάβαση στην κατάσταση s' κατά τη λήψη μίας δράσης a
- Ένα σύνολο κανόνων που περιγράφουν την παρατήρηση του περιβάλλοντος από τον πράκτορα

4.2 Αλγόριθμος Q-Learning

Το Q-learning είναι ένας αλγόριθμος RL χωρίς μοντέλο, δηλαδή ο πράκτορας δεν χρησιμοποιεί προβλέψεις για την επόμενη κατάσταση ή επιβράβευση. Με τον όρο κατάσταση (state) αναφερόμαστε σε όλους τις πιθανές θέσεων που μπορεί να βρεθεί ο πράκτορας, ενώ οι αποφάσεις που είναι διαθέσιμες σε κάθε κατάσταση είναι γνωστές ως *ενέργειες*. Ο όρος ¨Q¨ αναφέρεται στη συνάρτηση που υπολογίζει την *ποιότητα* (quality) των συνδυασμών κατάστασης-ενέργειας, οι οποίοι είναι υπεύθυνοι για την ενίσχυση της συμπεριφοράς ενός πράκτορα λογισμικού [17]. Οι πράκτορες ακολουθούν συγκεκριμένη πολιτική (policy), η οποία είναι ένας ορισμός της συμπεριφοράς του πράκτορα ανά πάσα στιγμή. Αξίζει να σημειωθεί ότι έαν κάποιο περιβάλλον λήψης αποφάσεων μοντελοποιηθεί χρησιμοποιώντας μία Finite Markov Decision Process - Πεπερασμένη Μαρκοβιανή Διεργασία Απόφασης (FMDP) εγγυάται μια βέλτιστη πολιτική, μεγιστοποιώντας τη συσσωρευτική επιβράβευση[1] [18].

$$Q^{new}(s_t, a_t) \leftarrow Q^{old}(s_t, a_t) + \alpha * (r_t + \gamma * maxQ(s_{t+1}, \alpha) - Q^{old}(s_t, a_t)) \quad (1)$$

Η Εξίσωση 1 παραπάνω περιγράφει τη συνάρτηση του Q-learning. Πιο συγκεκριμένα, ένας πίνακας Q είναι υπεύθυνος για τη διατήρηση των διαφορετικών τιμών q που αντιπροσωπεύουν τις διαφορετικές καταστάσεις του περιβάλλοντος και αρχικοποιείται σε κάποια αυθαίρετη τιμή. Το μέγεθος του πίνακα εξαρτάται από το μέγεθος καταστάσεων πολλαπλασιασμένο με τις διαθέσιμες ενέργειες του πράκτορα. Συνήθως αναφερόμαστε σε μια επανάληψη ως ένα *επεισόδιο* και διαρκεί όσο η νέα κατάσταση s_{t+1} δεν είναι τερματική. Πιο

[1] Κατά την διάρκεια των διαθέσιμων κινήσεων του πράκτορα οι επιστρεφόμενες ανταμοιβές προστίθενται και το αποτέλεσμα αντιπροσωπεύει την συσσωρευτική επιβράβευση σε μια επανάληψη του αλγορίθμου.

αναλυτικά, σε κάθε χρονικό βήμα t ο πράκτορας επιλέγει μια ενέργεια a_t, λαμβάνει μία επιβράβευση r_t και επιστρέφεται μια νέα κατάσταση s_{t+1}. Στη συνέχεια, μέσω της συνάρτησης $\mathcal{Q}$, ενημερώνεται η επιλεγμένη τιμή $\mathcal{Q}$ όπως φαίνεται στην Εξίσωση 1, η οποία εξαρτάται τόσο από την προηγούμενη κατάσταση s_t όσο και από την επιλεγμένη ενέργεια.

Όπως συμβαίνει με τους περισσότερους αλγόριθμους ML, ο συντελεστής α είναι η έννοια του *ποσοστού μάθησης* ή *μεγέθους βήματος* και παίρνει τιμές μεταξύ 0 και 1 $(0 < \alpha \leq 1)$. Αυτή η παράμετρος ελέγχει το βαθμό στον οποίο καινούριες πληροφορίες αντικαθιστούν τις παλιές [13]. Ένα υψηλό ποσοστό εκμάθησης κάνει τον πράκτορα να λαμβάνει υπόψη νέες πληροφορίες σε μεγαλύτερο βαθμό, ενώ ένας συντελεστής στο 0 δεν επιτρέπει για καμία μάθηση. Η επόμενη παράμετρος που υπάρχει στη εξίσωση είναι ο συντελεστής discount γ και αντιπροσωπεύει τη σημασία των μελλοντικών ανταμοιβών. Ένας χαμηλός συντελεστής discount κάνει τον πράκτορα να εκτιμά μόνο τις βραχυπρόθεσμες ανταμοιβές.

Η στρατηγική ϵ-greedy μπορεί επίσης να χρησιμοποιηθεί για την περαιτέρω ενίσχυση της μαθησιακής διαδικασίας, δίνοντας τη δυνατότητα ελέγχου του ποσοστού εξερεύνησης του πράκτορα [32, 28]. Αυτή η στρατηγική υλοποιείται με τη χρήση μιας παραμέτρου ϵ, όπου $(0 < \epsilon < 1)$, και είναι υπεύθυνη για την εξισορρόπηση της *εξερεύνησης-εκμετάλλευσης*. Η Εξίσωση 2 παρουσιάζει την αξιολόγηση της επόμενης ενέργειας a_t του πράκτορα.

$$a_t = \begin{cases} \text{Βήμα εκμετάλλευσης (καλύτερο)}, & 1 - \epsilon \\ \text{Βήμα εξερεύνησης (τυχαίο)}, & \text{αλλιώς} \end{cases} \tag{2}$$

Η παράμετρος ϵ αρχικοποιείται στην μεγαλύτερη τιμή της και κατά την διάρκεια της προσομοίωσης μειώνεται κατά ένα ποσοστό της τρέχουσας τιμής της. Η υιοθέτηση αυτής της στρατηγικής διασφαλίζει μια ιδιαίτερα διερευνητική συμπεριφορά στην αρχή και ιδιαίτερα εκμεταλλευτική συμπεριφορά σε μεταγενέστερα επεισόδια.

4.3 Περιορισμοί Γλώσσας

Η ικανότητα δημιουργίας και τροποποίησης των διάφορων συλλογών δεδομένων είναι ένα σημαντικό ζήτημα των αλγορίθμων μηχανικής μάθησης. Παρόλο που το Acumen παρέχει αρκετούς τρόπους για τη δημιουργία συλλογών δεδομένων, όπως εξηγείται στην ενότητα 2.3.2, τέτοιες κατασκευές συνοδεύονται επίσης από περιορισμούς στις αναθέσεις τιμών τους. Η τροποποίηση των στοιχείων μιας συλλογής δεδομένων με όλους τους δυνατούς τρόπους έχει μεγάλη σημασία, όπως για παράδειγμα την ολική ανάθεση τιμών, αναθέσεις σε όλα τα στοιχεία της με επανάληψη και αναθέσεις σε μεμονωμένα στοιχεία.

```
model Main(simulator) =
initially
  x=(1,2),
  t=0, t'=0
always
  t'=1,
  if t>1 then
    x(0)+ = x(0)+1,
    x(1)+ = x(1)-1,
    t+ = 0
  noelse
```

Το προηγούμενο μοντέλο αποτελεί ένα καλό παράδειγμα τέτοιων περιορισμών, καθώς παρουσιάζει κάποιο πρόβλημα κατά την εκτέλεσή του. Συγκεκριμένα, η προσπάθεια προσομοίωσης αυτού του μοντέλου προκαλεί ένα σφάλμα, το οποίο αν και δεν είναι πολύ ενημερωτικό υποδηλώνει ότι υπάρχει ένα πρόβλημα κατά την προσπάθεια διακριτικής εκχώρησης τιμής στο πρώτο πεδίο της λίστας x. Το ακριβές σφάλμα είναι το εξής:

```
Index(Var(Name(x,0)),List(Lit(GInt(0))))
```

Τέτοιος περιορισμός ισχύει επίσης και για πίνακες οποιασδήποτε διάστασης, όπως οι πίνακες. Ο Πίνακας 2 παρουσιάζει την υποστήριξη του Acumen για τις διαφορετικές επιλογές εκτέλεσης διακριτών αναθέσεων τόσο σε απλές λίστες όσο και σε πίνακες οποιασδήποτε διάστασης.

Διακριτές Εκχωρήσεις	**Λίστα**	**Πίνακας**
Ολόκληρη συλλογή	Ναι	Ναι
Όλα τα στοιχεία με επανάληψη	Ναι	Ναι
Μεμονωμένα στοιχεία	Όχι	Όχι

Πίνακας 2: Υποστήριξη για διακριτές εκχωρήσεις σε συλλογές

Ακόμη χειρότερα, η εκτέλεση συνεχών αναθέσεων στις τιμές των λιστών και πινάκων υποφέρουν επίσης από αντίστοιχους περιορισμούς. Για να παρουσιάσουμε ένα τέτοιο περιορισμό χρησιμοποιούμε το ακόλουθο μοντέλο, που χρησιμοποιεί μία επαναληπτική δήλωση *foreach* για να εκτελέσει τη συνεχή ανάθεση $x'(i) = 1$ σε όλα τα στοιχεία μιας λίστας.

```
model Main(simulator) =
initially
  x=(1,2), x'=(0,0)
always
  foreach i in 0:(length(x)-1) do
    x'(i)=1
```

Η εκτέλεση αυτού του παραδείγματος παράγει το ακόλουθο σφάλμα στην κονσόλα, υποδηλώνοντας ότι ο διερμηνευτής δεν είναι σε θέση να αναλύσει σωστά την αριστερή πλευρά της έκφρασης:

```
The left-hand side of an assignment must be of the form ‘e.x’.
x'(i)=i
```

Αυτός ο περιορισμός ισχύει επίσης για πίνακες οποιασδήποτε διάστασης. Ομοίως δυσλειτουργεί και η εκτέλεση συνεχών αναθέσεων σε μεμονωμένα στοιχεία των συλλογών δεδομένων.

```
model Main(simulator) =
initially
  x=((1,2), (3,4))
always
  x(0,1)=10
```

Σε αυτό το παράδειγμα, η συνεχής ανάθεση του αριθμού 10 στη θέση (0,1) του πίνακα x επιστρέφει σφάλμα παρόμοιο με το μοντέλο που χρησιμοποιεί επανάληψη παραπάνω. Ο Πίνακας 3 παρουσιάζει την τρέχουσα υποστήριξη για συνεχείς αναθέσεις σε οποιοδήποτε είδος λίστας:

Συνεχείς Εκχωρήσεις	**Λίστα**	**Πίνακας**
Ολόκληρη συλλογή	Ναι	Ναι
Όλα τα στοιχεία με επανάληψη	Όχι	Όχι
Μεμονωμένα στοιχεία	Όχι	Όχι

Πίνακας 3: Υποστήριξη για συνεχείς εκχωρήσεις σε συλλογές

Όπως φαίνεται από τον Πίνακα 2 και τον Πίνακα 3 υπάρχουν πολλοί περιορισμοί κατά τη λειτουργία ανάθεσης στους διάφορους τύπους συλλογών. Διεξήγαμε μια έρευνα στην ιστορία της ανάπτυξης του Acumen με την πάροδο του χρόνου και τα αποτελέσματα έδειξαν ότι ζητήματα σχετικά με τις συνεχείς αναθέσεις εμφανίζονται στο πέρασμα Desugarer ενώ ζητήματα σε διακριτές αναθέσεις παρουσιάστηκαν με την προσθήκη του περάσματος BTA.

4.3.1 Εξάλειψη του Προβλήματος Ανάλυσης Χρόνου Δέσμευσης

Η προσθήκη της BTA εισήγαγε επίσης ένα πρόβλημα όπου οι διακριτές εκχωρήσεις σε μεμονωμένα στοιχεία συλλογών οδηγούν σε σφάλματα. Θα ακολουθήσουμε το ίδιο παράδειγμα με την προηγούμενη ενότητα για να αναλύσουμε τη ρίζα του προβλήματος στην προσπάθεια επίλυσής του.

```
model Main(simulator) =
initially
  t=0, t'=1, x=(0,0)
always
  t'= 1,
  if t>1 then
    x(0)+ = x(0)+1,
    x(1)+ = x(1)-1,
    t+ = 0
  noelse
```

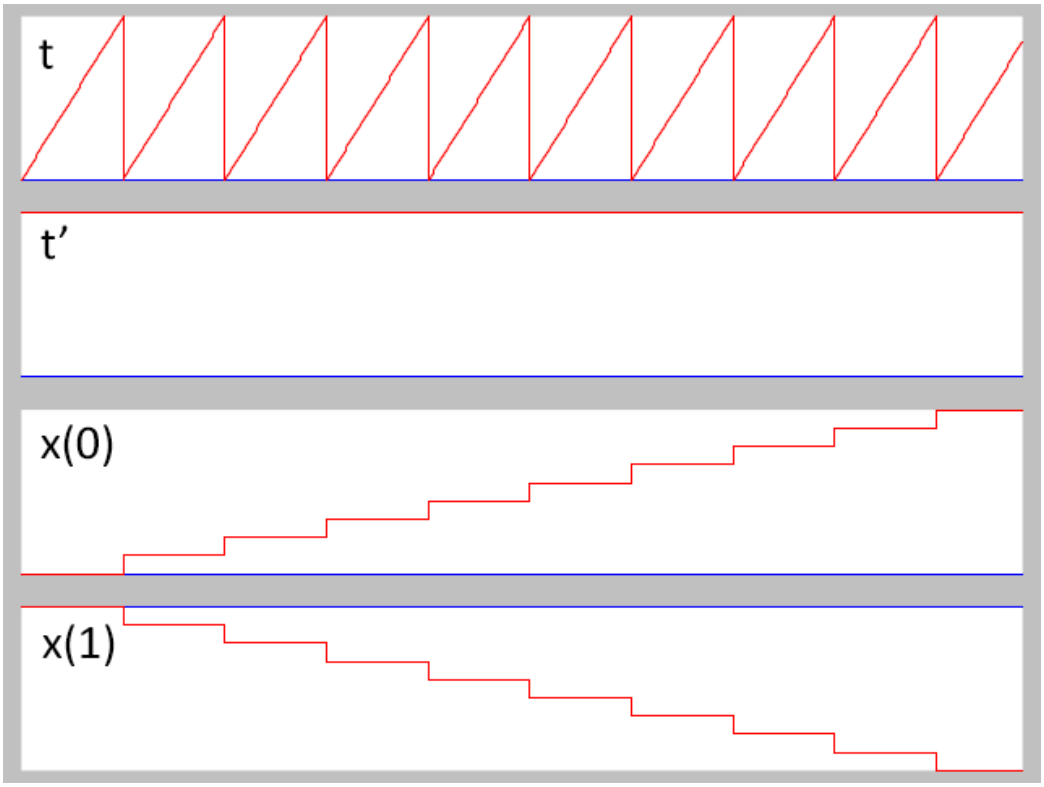
Αυτό το παράδειγμα δημιουργεί ένα χρονόμετρο t το οποίο αυξάνεται συνεχώς με την πάροδο του χρόνου λόγω του κανόνα της παραγώγου του $t' = 1$. Με την πάροδο ενός δευτερολέπτου, το χρονόμετρο επαναφέρεται στο μηδέν και ενημερώνονται τα δύο στοιχεία της λίστας x. Το πρώτο στοιχείο αυξάνεται κατά 1 ενώ το δεύτερο μειώνεται κατά 1. Το παράδειγμα όμως παράγει το ακόλουθο σφάλμα στην κονσόλα:

```
Index(Var(Name(x,0)), List(Lit(GInt(0))))
```

Το σφάλμα υποδηλώνει ότι υπάρχει εσφαλμένη συμπεριφορά κατά την αντιμετώπιση της δομής Index. Αυτές οι δομές αποτελούνται από δύο μέρη, το όνομα της μεταβλητής και την απαιτούμενη θέση στη λίστα. Σε αυτό το παράδειγμα το πρώτο μέρος *Var(Name(x,0))*, περιγράφει το όνομα της μεταβλητής και το βαθμό της παραγώγου της. Εδώ, μια τιμή μηδέν σημαίνει ότι δεν υπάρχει παράγωγος. Το δεύτερο μέρος *Lit(GInt(0))*, δείχνει τη θέση της λίστας που γίνεται η πρόσβαση. Αυτό το μέρος συμπληρώνεται από θέσεις σε όλες τις διαθέσιμες διαστάσεις κατά τη διάρκεια της λειτουργίας αναζήτησης.

Τα αποτελέσματα της έρευνας έδειξαν ότι το σφάλμα δημιουργείται κατά το βήμα της εξειδίκευσης, η οποία εξηγήθηκε στην Ενότητα 3.2.1. Μία συνάρτηση disODEs είναι υπεύθυνη για την επίλυση DAE σαν να είναι κανονικές εξισώσεις. Κατά τη διάρκεια αυτής της διαδικασίας επιλέγονται οι υπονοούμενες ODE και μέσω της εφαρμογής συμβολικής Gaussian Elimination - Απαλοιφή Gauss γίνεται προσπάθεια μετατροπής τους σε ODE ρητής μορφής. Ωστόσο, κατά την προσπάθεια διάκρισης μεταξύ υπονοούμενης και

ρητής μορφής ODE, δεν λαμβάνεται υπόψη η περίπτωση της δομής **Index** να υπάρχει στην αριστερή πλευρά μιας εξίσωσης, η οποία οδηγεί πάντα σε εσφαλμένη αναγνώριση ως υπονοούμενης μορφής. Επιλύσαμε αυτό το ζήτημα τροποποιώντας τη συλλογή των υπονοούμενων και ρητών μορφών έτσι ώστε να επιτρέπεται ο χειρισμός των δομών **Index**. Η εφαρμογή αυτών των αλλαγών κώδικα επιτρέπει την προσομοίωση του παραδείγματος στην αρχή αυτής της ενότητας που τώρα επιστρέφει αναμενόμενα σχέδια, όπως απεικονίζονται στο Σχήμα 7 παρακάτω:



Σχήμα 7: Σχεδιαγράμματα μετά την επιδιόρθωση του περάσματος BTA

Αξίζει να σημειωθεί ότι μετά την εφαρμογή του περάσματος BTA ορισμένα παραδείγματα που εμπεριέχονται στη διανομή του Acumen σταμάτησαν να λειτουργούν. Ένα σημαντικό αποτέλεσμα των αλλαγών κώδικα που υλοποιήσαμε και παρουσιάσαμε στην προηγούμενη παράγραφο, είναι το γεγονός ότι κάποια από αυτά τα παραδείγματα παρέχουν τώρα επιτυχημένες προσομοιώσεις ξανά. Αυτά τα παραδείγματα δημιουργήθηκαν από τον Emannuel Brelle[1] και αφορούν στην προσομοίωση των δικτύων perceptron και της πολυωνυμική παλινδρόμησης ως δυναμικών μοντέλων στο Acumen. Το γεγονός

[1] Ο φάκελος 03_Projects/07_Emmanuel περιέχει τα αναφερόμενα παραδείγματα

αυτό αποτελεί ένα αποτέλεσμα από τις αλλαγές κώδικα που αναφέρθηκαν παραπάνω και δεν σχετίζονται με το αντικείμενο της πτυχιακής. Παρακάτω δίνουμε την πλήρη λίστα των παραδειγμάτων που λειτουργούν ξανά μετά τις αλλαγές μας:

- Perceptron/perceptron.acm
- Perceptron/perceptronComments.acm
- PolynomilaReg-Versions/PolynomialRegression.acm

4.3.2 Εξάλειψη του Προβλήματος Desugar Pass

Η υποστήριξη για συνεχείς αναθέσεις σε συλλογές είναι πολύ περιορισμένη. Ενώ αναθέσεις στο σύνολο μιας λίστας (π.χ. *x=(5,pi)*) υποστηρίζονται, οποιαδήποτε άλλη περίπτωση αποτυγχάνει όπως απεικονίζεται στο Πίνακα 3. Θα εξετάσουμε το ακόλουθο μοντέλο:

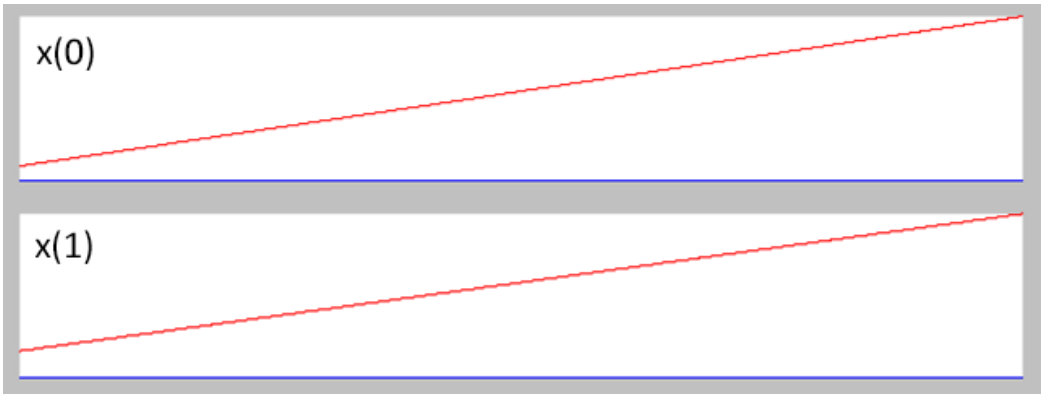
```
model Main(simulator) =
initially
  x=(1,2), x'=(0,0)
always
  foreach i in 0:(length(x)-1) do
    x'(i)=1
```

Σε αυτό το παράδειγμα, δημιουργείται μια απλή λίστα x παράλληλα με την πρώτη της παράγωγο. Στη συνέχεια, χρησιμοποιώντας τη λέξη-κλειδί της επανάληψης *foreach*, διασχίζουμε τη λίστα παραγώγων x' και αναθέτουμε την τιμή 1 σε κάθε στοιχείο. Ωστόσο, η προσπάθεια προσομοίωσης αυτού του μοντέλου, οδηγεί σε σφάλμα:

```
The left-hand side of an assignment must be of the form 'e.x'.
x'(i)=1
```

Το σφάλμα υποδηλώνει παρερμηνεία κατά την διάρκεια μίας συνεχής ανάθεσης. Εδώ, η λειτουργία αναζήτησης χρησιμοποιείται για την εύρεση και αξιολόγηση του αριστερού μέρους της ανάθεσης, όπου τελικά οδηγεί σε σφάλμα. Όπως εξηγείται στην ενότητα 3.2.2, πρώτα αξιολογείται η προϋπόθεση για το εαν η μεταβλητή που βρίσκεται στο αριστερό μέρος μιας εξίσωσης έχει αντιστοιχηθεί σε κάποια παρουσία μοντέλου. Περιπτώσεις που δεν ταιριάζουν σε αυτή η συνθήκη οδηγούν σε παρόμοια σφάλματα.

Καταφέραμε να επιλύσουμε τέτοια ζητήματα προσθέτοντας την περίπτωση της δομής Index να υπάρχει ως έκφραση στην αριστερή πλευρά και ακολουθώντας τη σωστή αναγνώριση ενεργειών που εμπεριέχονται στις κατασκευές Index, όπως αυτές αναφέρθηκαν στην Ενότητα 3.2.2[1]. Μετά την αντιμετώπιση του προβλήματος, η προσομοίωση του παραπάνω μοντέλου εκτελείται με επιτυχία και το πρόβλημα εξαλείφεται. Οι φιγούρες που παρουσιάζονται στο Σχήμα 8 δικαιολογούν τα αποτελέσματα:



Σχήμα 8: Σχεδιαγράμματα μετά την επιδιόρθωση του περάσματος Desugarer

4.4 Προσθήκες στις λειτουργίες των λιστών

Στις προηγούμενες ενότητες εξαλείψαμε διάφορα σημαντικά ζητήματα που παρεμπόδιζαν τη σωστή συμπεριφορά των αναθέσεων σε συλλογές δεδομένων. Ενώ ήδη επιτύχαμε να έχουμε ένα τεράστιο πλεονέκτημα στη διαχείριση τέτοιων συλλογών, υπάρχει ακόμα έλλειψη λειτουργιών σε αυτές. Κάποιες κοινές λειτουργίες όπως η εύρεση των μέγιστων και ελάχιστων τιμών μιας λίστας ή η επιστροφή ενός συγκεκριμένου αριθμού θέσης σε αυτή, είναι εξαιρετικά δύσκολο να εκφραστούν στο Acumen. Ας εξετάσουμε μια περίπτωση χρήσης όπου είναι απαραίτητο να υπολογιστεί και να επιστραφεί η μέγιστη τιμή μιας απλής λίστας όπως η $x = (1, 1, 2, 3)$.

```
foreach i in 0:(length(x)-1) do
  if x(i) > max then
    max+ = x(i)
  noelse
```

[1]Πρέπει επίσης να σημειώσουμε ότι μια πρώτη προσπάθεια για την επίλυση αυτών των ζητημάτων έγινε από τον Yingfu Zeng στις 05.2015 ενώ αργότερα η ίδια προσπάθεια ακυρώθηκε από τον Adam Duraz στις 11.2015.

Οι επαναληπτικές δηλώσεις, όπως αυτή που χρησιμοποιείται σε αυτό το παράδειγμα, αποτυγχάνουν καθώς δεν επιτρέπεται η επαναλαμβανόμενη ανάθεση στην ίδια μεταβλητή. Η αποφυγή αυτών των ζητημάτων με τη χρήση της σημειογραφίας Acumen αποδείχθηκε εξαιρετικά περίπλοκη διαδικασία, αν όχι αδύνατη, να επιτευχθεί. Η ανάγκη για λειτουργίες που εφαρμόζονται σε συλλογές δεδομένων είναι αναγκαιότητα για τη γλώσσα και για αυτό το λόγο εισήγαμε μια ποικιλία κοινών λειτουργιών όπως αυτές παρουσιάζονται στον ακόλουθη λίστα:

- **max(x)**, επιστρέφει τη μέγιστη τιμή μιας λίστας
- **min(x)**, επιστρέφει τη ελάχιστη τιμή μιας λίστας
- **argmax(x)**, επιστρέφει τη θέση μέγιστης τιμής σε μια λίστα
- **argmin(x)**, επιστρέφει τη θέση ελάχιστης τιμής σε μια λίστα
- **argrand(x)**, επιστρέφει μια τυχαία θέση μιας λίστας

5 Περίπτωση Μελέτης Πράκτορα Επίλυσης Λαβυρίνθου

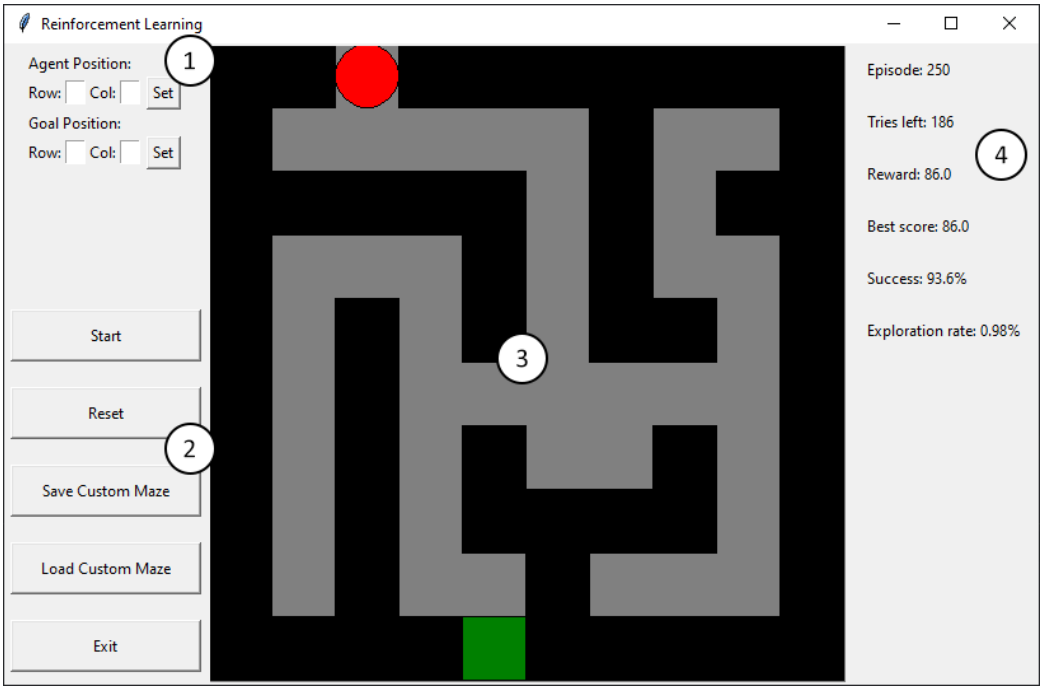
Σε αυτήν την ενότητα παρουσιάζουμε μια μικρή εφαρμογή Python [30], που υλοποιεί τον αλγόριθμο Q-learning, όπως αυτός παρουσιάστηκε στην ενότητα 4.2. Η εκτέλεση της εφαρμογής δείχνει πώς η RL μπορεί να χρησιμοποιηθεί για να εκπαιδεύσει έναν πράκτορα λογισμικού να μάθει να πλοηγείται σε ένα λαβύρινθο δύο διαστάσεων. Στη συνέχεια διερευνούμε μια προσπάθεια υλοποίησης του ίδιου αλγορίθμου στο περιβάλλον Acumen, χρησιμοποιώντας μια έκδοση που ενσωματώνει τις τροποποιήσεις σε κώδικα που αναφέρονται στις υπό-ενότητες 4.3.1 και 4.3.2. Στο υπόλοιπο κείμενο της ενότητας παρουσιάζουμε σκέψεις και συγκρίσεις μεταξύ των δύο υλοποιήσεων.

5.1 Εφαρμογή Python

Η εφαρμογή[1] στη γλώσσα της Python περιέχει ένα Graphical User Interface - Γραφικό Περιβάλλον Χρήστη υλοποιημένο με τη βιβλιοθήκη TkInter [22] και απεικονίζεται στο Σχήμα 9. Η διάταξη της εφαρμογής χωρίζεται σε 3 κάθετα μέρη. Στο αριστερό μέρος υπάρχουν επιλογές για τη ρύθμιση της αρχικής θέσης του πράκτορα (κόκκινος κύκλος), καθώς και της θέσης του στόχου (πράσινο τετράγωνο) στο λαβύρινθο (1). Επιπλέον, υπάρχουν επιλογές με την μορφή κουμπιών για την έναρξη της προσομοίωσης και την επαναφορά του λαβυρίνθου, με την εκκαθάριση όλων των ορίων (μαύρα τετράγωνα). Τα όρια αντιπροσωπεύουν καταστάσεις όπου ο πράκτορας δεν επιτρέπεται να μετακινηθεί. Η εφαρμογή παρέχει επίσης υποστήριξη για αποθήκευση και φόρτωση προσαρμοσμένων λαβυρίνθων ως αρχείων με την κατάληξη xml (2).

Το μεσαίο τμήμα δείχνει το στιγμιότυπο της τρέχουσας κατάστασης του λαβυρίνθου (3). Το στιγμιότυπο αυτό είναι διαδραστικό με διάφορους τρόπους. Με το πάτημα του αριστερού πλήκτρου του ποντικιού σε οποιοδήποτε πλακίδιο (tile), γίνεται εισαγωγή ενός ορίου ενώ με το δεξί πλήκτρο εκτελείται αφαίρεση. Σε οποιοδήποτε πλακίδιο, η χρήση του μεσαίου πλήκτρου στο ποντίκι εκτυπώνει τις τέσσερις τιμές Q στην κονσόλα, οι οποίες δείχνουν την ποιότητα των διαφορετικών ενεργειών στη συγκεκριμένη κατάσταση. Στο δεξί τμήμα εμφανίζονται πληροφορίες σχετικά με την τρέχουσα κατάσταση του αλγορίθμου (4), όπως λεπτομέρειες σχετικά με τον αριθμό του ενεργού επεισοδίου, την τρέχουσα συσσωρευτική ανταμοιβή, την καλύτερη ανταμοιβή που έχει βρεθεί, καθώς και τα ποσοστά της εξερεύνησης και της επιτυχίας του πράκτορα.

[1] Διεύθυνση: `https://github.com/sotostzam/artificial-intelligence`, κάτω από το φάκελο machine-learning/q-learning.



Σχήμα 9: Πράκτορας επίλυσης λαβυρίνθου στην Python

Η εφαρμογή περιλαμβάνει επίσης την πολιτική ϵ-greedy. Η μεταβλητή αυτής της πολιτικής, αρχικοποιείται στην τιμή 1 και μειώνεται στην αρχή κάθε επεισοδίου με την ακόλουθη έκφραση: $\epsilon = \epsilon * 97/100$. Κάθε φορά που ο πράκτορας εκτελεί μια ενέργεια, μία συνάρτηση ανατροφοδότησης παρέχει μία ανταμοιβή. Οι διάφορες ανταμοιβές εμπεριέχονται σε ένα πίνακα που έχει το ίδιο μέγεθος με το χώρο καταστάσεων, και κάθε στοιχείο του περιέχει την ανταμοιβή που θα επιστραφεί εάν ο πράκτορας επιλέξει αυτήν την κατάσταση. Ο πράκτορας βελτιστοποιεί τη συμπεριφορά του με σκοπό την μεγιστοποίηση αυτής της ανταμοιβής. Οι τιμές των διάφορων ανταμοιβών απεικονίζονται στον ακόλουθο πίνακα:

Επιλεγμένη Ενέργεια	**Ανταμοιβή**
Κίνηση σε διαθέσιμο πλακίδιο	-1
Κίνηση σε όριο	-10
Εύρεση στόχου	100

Πίνακας 4: Ανταμοιβές με βάση τις ενέργειες

Η MDP που περιγράφει το συγκεκριμένο λαβύρινθο περιέχει τρεις τερματικές καταστάσεις που αξιολογούνται κατά τη διάρκεια κάθε επεισοδίου. Η πρώτη κατάσταση ενεργοποιείται όταν ο αριθμός του τρέχοντος επεισοδίου υπερβεί την επιτρεπόμενο μέγιστο αριθμό επεισοδίων. Η δεύτερη κατάσταση ενεργοποιείται ο πράκτορας βρεθεί στην θέση του στόχου. Η τρίτη και τελευταία κατάσταση γίνεται ενεργή όταν ο πράκτορας έχει ξεπεράσει τις διαθέσιμες κινήσεις του για το κάθε επεισόδιο[1]. Εάν βρεθεί κάποια από τις τερματικές καταστάσεις, εκτελείται εκκίνηση ενός νέου επεισοδίου και ο πράκτορας επαναφέρεται στην αρχική του κατάσταση (θέση).

Q-Table Positions	Actions			
	Left (0)	Up (1)	Right (2)	Down (3)
(0, 2)	0	0	0	0
(5, 4)	0	0	0	0
(8,4)	0	0	0	0

↓ Training

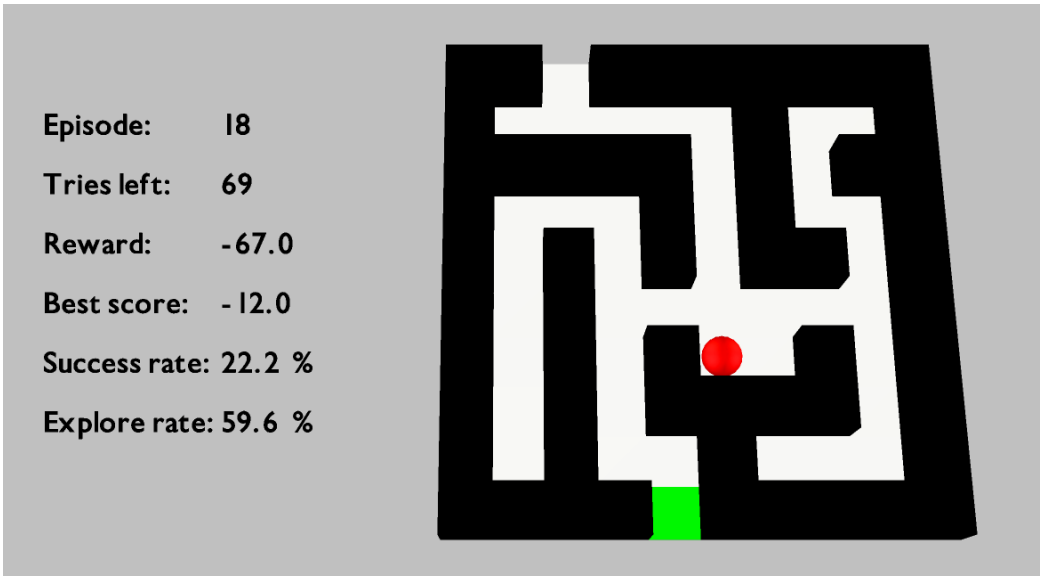
Q-Table Positions	Actions			
	Left (0)	Up (1)	Right (2)	Down (3)
(0, 2)	-11.29555251	-12.24345596	-12.57515642	1.23808058
(5, 4)	71.27999294	1.35525552	-7.06488183	-6.7356234
(8,4)	5.0052398	19.44037061	30.76566736	99.99938719

Σχήμα 10: Τιμές πίνακα Q πριν και μετά την μάθηση

Τα αποτελέσματα από την εφαρμογή της συνάρτησης Q-learning στον Πίνακα Q, τόσο κατά την αρχικοποίηση του όσο και μετά την εκμάθηση για 250 επεισόδια, απεικονίζονται στο Σχήμα 10 παραπάνω. Είναι επίσης χρήσιμο να σημειωθεί ότι δεν εκτελείται ποτέ η ενημέρωση των τιμών Q που αντιπροσωπεύουν τερματικές καταστάσεις.

[1]Αξίζει να αναφέρουμε ότι γίνεται αρχικοποίηση τόσο της τιμής του μέγιστου αριθμού επεισοδίων όσο και των διαθέσιμων κινήσεων του πράκτορα ανά επεισόδιο σε κάποια αφηρημένη τιμή. Στη δική μας εφαρμογή οι τιμές αυτές αρχικοποιούνται σε 250 επεισόδεια και 200 κινήσεις αντίστοιχα.

5.2 Εφαρμογή Acumen



Σχήμα 11: Πράκτορας Επίλυσης Λαβυρίνθου στο Acumen

Στόχος μας σε αυτή την υλοποίηση ήταν να παραμείνουμε όσο το δυνατόν πιο πιστοί στην υλοποίηση της Python, όπως παρουσιάστηκε στην προηγούμενη ενότητα. Μια σημαντική πρώτη διαφορά υπάρχει κατά την οπτικοποίηση του λαβυρίνθου στις δύο γλώσσες. Στο Acumen η κινούμενη εικόνα εμφανίζεται σε μια σκηνή 3D, η οποία υποστηρίζει μόνο ενέργειες χειρισμού σκηνής, όπως μεγέθυνση και μετακίνηση της κάμερας. Επομένως, δεν είναι δυνατή η δυναμική τροποποίηση του περιβάλλοντος. Για αυτό το λόγο, χρησιμοποιήσαμε μια προεπιλεγμένη διάταξη λαβυρίνθου που είναι όμοια και στις δύο υλοποιήσεις.

Για να υλοποιήσουμε τον τρισδιάστατο λαβύρινθο, δημιουργούμε ένα μοντέλο με το όνομα **Environment**. Αυτό το μοντέλο είναι υπεύθυνο για τη δημιουργία όλων των απαραίτητων τρισδιάστατων αντικειμένων για τη σωστή απεικόνιση της διάταξης του λαβυρίνθου. Το μοντέλο αυτό περιέχει ένα πίνακα επιβραβεύσεων (rewards). Μέσω της δήλωσης match του Acumen ορίζουμε διαφορετικά βήματα για την αρχικοποίηση αυτού του πίνακα σύμφωνα με τις τιμές από το Πίνακα 4. Τα πρώτα βήματα τροποποιούν τις τιμές των στοιχείων που αντιπροσωπεύουν τα όρια (-10) καθώς και τη θέση στόχου (+100). Ένα τελικό βήμα είναι υπεύθυνο για την δημιουργία των αντικειμένων σύμφωνα με την τιμή της ανταμοιβής τους: μία τιμή -10 δημιουργεί ένα μαύρο κουτί ενώ μία τιμή +100 δημιουργεί ένα λεπτό πράσινο τετράγωνο. Η προκύπτουσα σκηνή φαίνεται στο Σχήμα 11 παραπάνω.

Στη συνέχεια παρουσιάζουμε το μοντέλου του πράκτορα (**Agent**). Αυτό το μοντέλο δημιουργεί όλες τις απαραίτητες παραμέτρους που σχετίζονται με τη συμπεριφορά του πράκτορα, όπως την τρέχουσα θέση και την θέση επαναφοράς, τις διαθέσιμες κινήσεις του και την συσσωρευτική ανταμοιβή. Απαραίτητη είναι και η δημιουργία ενός τρισδιάστατου αντικείμενου (κόκκινη σφαίρα) για την απεικόνιση τον πράκτορα. Μια μεταβλητή *action* αντιπροσωπεύει την τρέχουσα κατάσταση του πράκτορα και λαμβάνει τις εξής τιμές:

- **left**, μειώνει τη θέση x του πράκτορα κατά 1
- **up**, αυξάνει τη θέση y του πράκτορα κατά 1
- **right**, αυξάνει τη θέση x του πράκτορα κατά 1
- **down**, μειώνει τη θέση y του πράκτορα κατά 1
- **reset**, επαναφέρει τη θέση, ανταμοιβή και κινήσεις του πράκτορα
- **standby**, κατάσταση αναμονής για αλλαγές

Κάθε φορά που ο πράκτορας πρέπει να εκτελέσει μια ενέργεια, η μεταβλητή action τροποποιείται έτσι ώστε να αντιπροσωπεύει την αντίστοιχη κίνηση. Μετά την εκτέλεση των οδηγιών της τρέχουσας κατάστασης, η τιμή της επόμενης κατάστασης είναι πάντα η **standby**. Αυτή η κατάσταση προσομοιώνει μια επίμονη (persist) συμπεριφορά, όπου ο πράκτορας αναμένει νέες οδηγίες.

Η τελευταία και υποχρεωτική διαμόρφωση είναι αυτή του μοντέλου **Main** που εφαρμόζει τον αλγόριθμο Q-learning. Το συγκεκριμένο μοντέλο εκτελεί την αρχικοποίηση των ακόλουθων μεταβλητών: *ποσοστό μάθησης*, *συντελεστής έκπτωσης*, παράμετρος *έψιλον* και τον πίνακα Q. Ωστόσο, αντίθετα με την εφαρμογή σε Python, αυτό το μοντέλο δημιουργεί δύο επί πλέον μεταβλητές, την περίοδο *period* και τη φάση *phase*. Η πρώτη ελέγχει τη χρονική τιμή μεταξύ κάθε ενέργειας και αποτελεί προαιρετική επιλογή[1]. Η δεύτερη μεταβλητή, αντιπροσωπεύει τρία βήματα:

- **init**, ορίζει την διάρκεια προσομοίωσης στα 5 λεπτά
- **scout**, υπολογισμός επόμενης κίνησης (εξερεύνηση ή εκμετάλλευση)
- **action**, αξιολόγηση τερματικών καταστάσεων και μετακίνηση πράκτορα

[1]Η εξάλειψη της μεταβλητής έχει ως αποτέλεσμα ένα στάσιμο χρόνο προσομοίωσης που λαμβάνει μέρος σε ένα και μόνο χρονικό βήμα (timestep). Αυτό οφείλεται στο γεγονός ότι το μοντέλο Main εκφράζεται μόνο από μια συλλογή διακριτών γεγονότων.

Η τιμή της μεταβλητής *phase* αξιολογείται σε κάθε χρονικό βήμα (timestep), ενώ η προσομοίωση ξεκινά με ένα βήμα **init** και στη συνέχεια εκτελεί τα βήματα **scout** και **action** εναλλάξ έως ότου επιτευχθεί ο μέγιστος αριθμός επεισοδίων. Η τιμή της στρατηγικής ϵ-greedy υπολογίζεται σε κάθε βήμα **scout**, και δείχνει εάν ο πράκτορας θα εκτελέσει μία κίνηση εξερεύνησης ή μία κίνηση εκμετάλλευσης. Στη συνέχεια πραγματοποιείται ένα βήμα **action**. Αυτό το βήμα είναι υπεύθυνο για την αξιολόγηση των συνθηκών επαναφοράς και τερματισμού. Συγκεκριμένα, υπάρχουν δύο συνθήκες επαναφοράς που αξιολογούνται:

1. Ο πράκτορας βρίσκεται στην κατάσταση του στόχου
2. Ο πράκτορας εξάντλησε όλες τις διαθέσιμες κίνησεις του

Και στις δύο περιπτώσεις εκτελείται επαναφορά του πράκτορα στην αρχική του κατάσταση, ενημερώνεται η τιμή της παραμέτρου *epsilon* και εάν χρειάζεται η τιμή της *καλύτερης ανταμοιβής*, και αξιολογείται η κατάσταση τερματισμού. Αυτή η κατάσταση τερματίζει την προσομοίωση όταν ο τρέχων αριθμός επεισοδίων υπερβεί τον προκαθορισμένο μέγιστο αριθμό, διαφορετικά ένα νέο επεισόδιο ξεκινά.

Εάν δεν ισχύει καμία από τις συνθήκες επαναφοράς, ο πράκτορας εκτελεί μια ενέργεια (φάση action). Ανάλογα με την επιλεγμένη κίνηση, ελέγχεται η εγκυρότητα της κίνησης. Θεωρούμε έγκυρη κίνηση ως τη μετακίνηση σε μια νέα κατάσταση που δεν αποτελεί όριο και δεν υπερβαίνει το μέγεθος του λαβυρίνθου. Εφόσον η κίνηση θεωρηθεί ως έγκυρη, ενημερώνεται η τιμή Q της τρέχουσας κατάστασης του πράκτορα σύμφωνα με την Εξίσωση 1. Ακολούθως, η ανταμοιβή για την μετάβαση στη νέα κατάσταση προστίθεται στην συσσωρευτική ανταμοιβή του πράκτορα και ο πράκτορας μεταφέρεται στη νέα κατάσταση. Αντιθέτως, εάν η νέα κατάσταση δεν είναι έγκυρη τότε ο πράκτορας παραμένει στην ίδια κατάσταση και επιστρέφεται μία ανταμοιβή με τιμή -10. Τέλος, ενημερώνονται οι τιμές των αντικειμένων **Text** όπως αυτά εμφανίζονται στην αριστερή πλευρά του Σχήματος 11.

5.3 Σύγκριση Κώδικα

Στην Python χρησιμοποιείται μία δημοφιλής βιβλιοθήκη όπως η NumPy [21] για την υποστήριξη ισχυρής δημιουργίας και χειρισμού ν-διαστάσεων πινάκων. Μετά τις προσθήκες σε λειτουργίες συλλογών, όπως αυτές παρουσιάστηκαν στην Ενότητα 4.4, μπορούμε εύκολα να κάνουμε συγκρίσεις μεταξύ της Python και του Acumen. Η μόνη διαφορά που χρειάζεται προσοχή είναι η μεταβλητή *explore_factor*, της οποίας η τιμή έχει βρεθεί πριν την αξιολόγηση

της συνθήκης. Ο Πίνακας 5 δείχνει τη σύνταξη κώδικα κατά την αξιολόγηση της επόμενης ενέργειας του πράκτορα:

Python	**Acumen**
if random.uniform(0, 1) <epsilon	if explore_factor <epsilon then
action = random.randint(0, 3)	action +=
	argrand(qtable(agent.pos(0), agent.pos(1)))
else:	else
action =	action +=
np.argmax(	argmax(
q_table[agent_pos[0], agent_pos[1]])	qtable(agent.pos(0), agent.pos(1))),
	explore_factor += rand(),

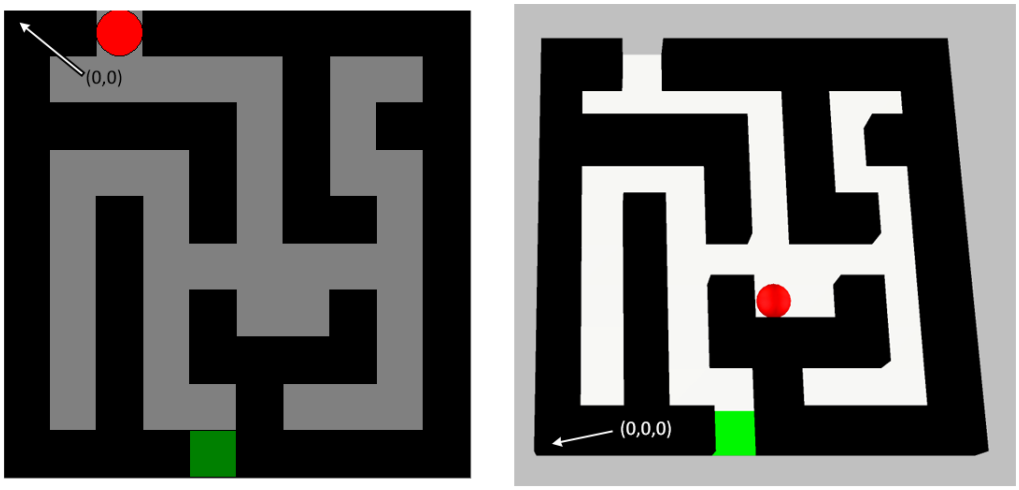
Πίνακας 5: Αξιολόγηση επόμενης ενέργειας με βάση την πολιτική ϵ-greedy

Οι δηλώσεις υπό συνθήκες είναι επίσης αρκετά όμοιες σε φύση και στις δύο υλοποιήσεις, με ελάχιστες διαφορές στη σύνταξη τους. Για παράδειγμα, η σύνταξη μιας δήλωσης if που αξιολογεί την εγκυρότητα μιας ενέργειας υλοποιείται ως εξής:

Python	**Acumen**
if agent_pos[1] >0 and	if agent.pos(0) >0 &&
rewards[agent_pos[0], agent_pos[1] - 1]	env.rewards(agent.pos(0)-1, agent.pos(1))
!= -10:	<>-10 then
# Εκτέλεση κίνησης	\\Εκτέλεση κίνησης

Πίνακας 6: Αξιολόγηση εγκυρότητας κατά την αριστερή κίνηση

Μία ακόμη θεμελιώδης διαφορά υπάρχει στον τρόπο διάσχισης του πίνακα Q. Στη βιβλιοθήκη TkInter της Python η αρχική θέση (0,0) βρίσκεται στην επάνω αριστερή γωνία του παραθύρου, οπότε η μετακίνηση προς τα δεξιά σημαίνει διάσχιση στις στήλες pixel, ενώ η κίνηση προς τα κάτω σημαίνει διάσχιση στις σειρές pixel. Σε αντίθεση, η τρισδιάστατη βιβλιοθήκη του Acumen jPCT ορίζει ως αρχική θέση το κέντρο της σκηνής. Είναι πολύ εύκολο να μετακινήσουμε αντικείμενα στη σκηνή τροποποιώντας τις θέσεις στους άξονες (x, y, z). Αυτό απεικονίζεται καλύτερα στο Σχήμα 12.



Σχήμα 12: Αρχικές θέσεις σε TkInter (αριστερά) και jPCT (δεξιά)

Στο Acumen όλες οι αναθέσεις εκτελούνται ταυτόχρονα. Η ενημέρωση και η χρήση μεταβλητών στο ίδιο πεδίο δεν είναι δυνατή και απαιτεί ένα πρόσθετο χρονικό βήμα. Αντίθετα στην Python υπάρχει η ικανότητα τροποποίησης και άμεσης χρήσης μεταβλητών, καθώς ακολουθείται σειριακή εκτέλεση. Στην εφαρμογή Acumen αποφασίσαμε να αξιοποιήσουμε κάποια επί πλέον χρονικά βήματα που κάνουν την υλοποίηση του αλγορίθμου πιο ευανάγνωστη.

current_state = tuple(np.copy(agent_pos))
new_state = q_table[agent_pos[0], agent_pos[1]]
reward = rewards[agent_pos[0], agent_pos[1]]
max_future_q = np.max(new_state)
current_q = q_table[current_state[0], current_state[1]][action]
q_table[current_state[0]][current_state[1]][action] = new_q

Πίνακας 7: Τοπικές μεταβλητές της συνάρτησης Q στην Python

Ο Πίνακας 7 παρουσιάζει τις διάφορες παραμέτρους στην Python που αξιολογούνται πριν τη συνάρτηση Q. Η τελευταία παράμετρος αποτελεί εξαίρεση καθώς χρησιμοποιείται μετά τη συνάρτηση. Στην υλοποίηση σε Acumen η φάση **action** είναι εκείνη που αξιολογεί τις ίδιες μεταβλητές στο ίδιο χρονικό βήμα χωρίς να τις αναθέτει σε καινούρια ονόματα μεταβλητών.

Στον Πίνακα 8 παρακάτω παρουσιάζεται η σύνταξη της ενημέρωσης ενός πεδίου του πίνακα Q. Παρατηρήστε πως στην Python μπορούμε να χρησιμοποιήσουμε άμεσα τις μεταβλητές που παρουσιάζονται στον Πίνακα 8, ενώ στη σύνταξη Acumen επιλέξαμε να τις αξιολογήσουμε κατά τη διάρκεια της εκτέλεσης. Ενώ είναι εφικτό να δημιουργήσουμε καινούριες μεταβλητές με την ίδια ονομασία στο Acumen, δεν ακολουθήσαμε αυτή την ιδέα επειδή θα ήταν αναγκαία η χρήση ενός πρόσθετου χρονικού βήματος για τον υπολογισμό τους, αυξάνοντας την πολυπλοκότητα του αλγορίθμου.

Python	**Acumen**
new_q =	qtable(agent.pos(0), agent.pos(1), action) +=
current_q +	qtable(agent.pos(0), agent.pos(1), action) +
LEARNING_RATE *	LEARNING_RATE *
(reward +	(env.rewards(agent.pos(0)-1, agent.pos(1)) +
DISCOUNT *	DISCOUNT *
max_future_q -	max(qtable(agent.pos(0)-1, agent.pos(1))) -
current_q)	qtable(agent.pos(0), agent.pos(1), action)),

Πίνακας 8: Κώδικας συνάρτησης Q για Python και Acumen

Ως αποτέλεσμα των συγκρίσεων, πρέπει να αναφέρουμε ότι η απεικόνιση και οι δημιουργία κινούμενων εικόνων είναι πιο εύκολο να επιτευχθεί στο Acumen. Στην Python πρέπει να δημιουργήσουμε ειδικές συναρτήσεις για τον ορισμό κάθε δισδιάστατου αντικείμενου καθώς και να κάνουμε μετατροπή των μονάδων κίνησης σε μονάδες pixel. Από την άλλη, οι μεταβλητές και το εύρος ορισμού τους χρειάζονται ιδιαίτερη προσοχή στο Acumen και απαιτούν ιδιαίτερη προσοχή για την επίτευξη της ίδιας λειτουργικότητας σε σχέση με ένα πρόγραμμα γραμμένο στη γλώσσα Python.

6 Συμπεράσματα

Αυτή η πτυχιακή περιγράφει τις δυνατότητες και τους περιορισμούς της γλώσσας Acumen όσο αναφορά την ικανότητα μοντελοποίησης Κυβερνο-Φυσικών Συστημάτων χρησιμοποιώντας μια προσέγγιση RL. Παρουσιάσαμε ποιες γλωσσικές δυνατότητες χρειάζονται για την προσομοίωση πρακτόρων λογισμικού που λαμβάνουν διακριτές ενέργειες για πλοήγηση σε ένα περιβάλλον, παρουσιάζοντας έξυπνες συμπεριφορές. Επίσης δημιουργήσαμε μια μελέτη περίπτωσης που υλοποιεί αυτή τη νέα προσέγγιση.

6.1 Περίληψη των συνεισφορών

Σε αυτή τη πτυχιακή εντοπίσαμε κάποιους γλωσσικούς περιορισμούς όσον αναφορά στην λειτουργικότητα των συλλογών. Ο σκοπός ήταν να ενσωματώσουμε αλλαγές κώδικα στην εφαρμογή για να επιτευχθεί σωστή μοντελοποίηση μεθόδων RL σε διακριτό χώρο. Στο Πίνακα 2 και στο Πίνακα 3 δείχνουμε την τωρινή υποστήριξη του Acumen για αναθέσεις σε λίστες, ενώ μετά την εφαρμογή των αλλαγών τον υπάρχων κώδικα του Acumen υπάρχει βελτιωμένη υποστήριξη για τέτοιες λειτουργίες όπως φαίνεται στο Πίνακα 9 παρακάτω. Η χρήση του όρου ¨Σ-Λίστα¨ αντιπροσωπεύει συνεχής αναθέσεις και ο όρος ¨Δ-Λίστα¨ διακριτές αναθέσεις.

Αναθέσεις	**Δ-Λίστα**	**Δ-Πίνακας**	**Σ-Λίστα**	**Σ-Πίνακας**
Ολόκληρη συλλογή	Ναι	Ναι	Ναι	Ναι
Όλα τα στοιχεία με επανάληψη	Ναι	Ναι	Ναι	Ναι
Μεμονωμένα στοιχεία	Ναι	Ναι	Ναι	Ναι

Πίνακας 9: Βελτιωμένη υποστήριξη για αναθέσεις σε δομές συλλογών

Αυτή η πτυχιακή συμβάλει επίσης στη βελτίωση της λειτουργικότητας των συλλογών δεδομένων στη γλώσσα Acumen, καθώς παρουσιάζει νέες επιλογές όσον αφορά την αναζήτηση και εύρεση τιμών σε τέτοιες συλλογές. Επιπρόσθετα παρουσιάζουμε μια μελέτη περίπτωσης δημιουργημένη τόσο σε σύνταξη Python όσο και σε Acumen, προβάλλοντας διαφορές ανάμεσα στον κώδικα και απεικονίζοντας τη χρήση μεθόδων RL μέσα από ένα εργαλείο μοντελοποίησης για CPS.

6.2 Μαθήματα

Το πιο σημαντικό μάθημα που αποκομίσαμε κατά τη διάρκεια αυτής της πτυχιακής είναι ότι τα CPS μπορούν να επωφεληθούν σε μεγάλο βαθμό με την κατασκευή έξυπνης συμπεριφοράς, ενισχύοντας τον τρόπο με τον οποίο αντιδρούν σε διαφορετικά ερεθίσματα από ένα περιβάλλον. Αυτό παρουσιάζεται μέσω της προοπτικής του Acumen, μιας γλώσσας μοντελοποίησης που χρησιμοποιείται για τη μοντελοποίηση και την προσομοίωση υβριδικών συστημάτων. Τα αποτελέσματα των συγκρίσεων μας δείχνουν πως το Acumen μπορεί να χρησιμοποιηθεί για την εύκολη μοντελοποίηση και προσομοίωση τόσο διακριτών όσο και συνεχή μοντέλων. Το γεγονός αυτό συνδέεται άμεσα με τις χαμηλές απαιτήσεις στη σύνταξη της γλώσσας η οποία μπορεί εύκολα να συγκριθεί με κοινές γλώσσες προγραμματισμού, όπως η Python.

6.3 Μελλοντική Δουλειά

Η προσέγγιση που μελετήσαμε δεν είχε σκοπό να εξετάσει διεξοδικά την ποικιλία των διαφορετικών αλγορίθμων RL ή ML με οποιονδήποτε τρόπο. Βασίστηκε σε ένα πολύ συγκεκριμένο παράδειγμα περιβάλλοντος με διακριτό χώρο καταστάσεων που μπορεί να περιγραφεί με μία FMDP. Αυτή η μελέτη είχε σκοπό να δείξει εάν και πώς το Acumen μπορεί να υποστηρίξει τεχνικές μηχανικής μάθησης, στην περίπτωσή μας έναν αλγόριθμο RL που ονομάζεται Q-learning.

Σε μελλοντικές εργασίες θα θέλαμε να διερευνήσουμε και να αναπτύξουμε γενικές τεχνικές για την ενσωμάτωση τεχνικών ML στο Acumen. Για παράδειγμα, θα θέλαμε να διερευνήσουμε την ίδια προσέγγιση αλλά σε ένα συνεχές περιβάλλον. Αυτό μπορεί να περιγραφεί από ένα μοντέλο cart-pole στο οποίο ένας πράκτορας λογισμικού μαθαίνει να ελέγχει ένα αμαξίδιο στο γνωστό inverted pendulum problem[1], προκειμένου να ισορροπήσει ένα εκκρεμές πάνω του.

Επιπλέον, θα πρέπει να διεξαχθεί περαιτέρω έρευνα στην εφαρμογή του περάσματος Binding Time Analysis - Ανάλυση Χρόνου Δέσμευσης. Παρά τα πολλά πλεονεκτήματα που παρείχε στη γλώσσα Acumen, εισήγαγε και ορισμένα προβλήματα. Υπάρχουν δύο εντοπισμένα ζητήματα που προκαλούν αποτυχία προσομοιώσεων παραδειγμάτων της διανομής Acumen. Αυτά τα ζητήματα αφορούν τη συνάρτηση τυχαίου αριθμού *rand()* και τη χρήση της λέξης-κλειδί *sum*.

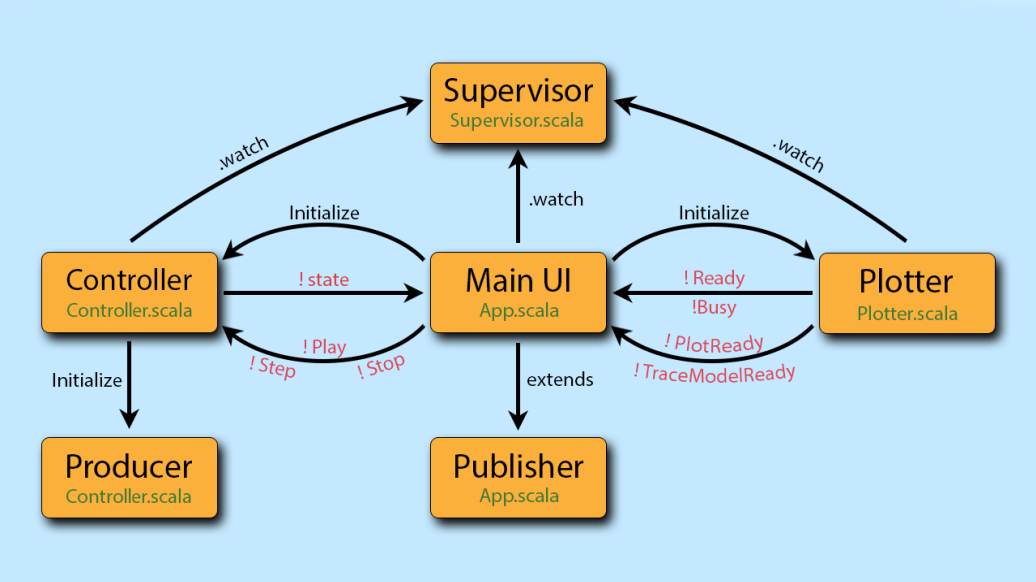
[1] Διεύθυνση: `https://en.wikipedia.org/wiki/Inverted_pendulum`

Παραρτήματα

Α΄ Περιβάλλον Acumen

Σε αυτό το παράρτημα Α παρέχουμε πληροφορίες σχετικά με τον τρόπο που το GUI του Acumen επιτυγχάνει αποκριτικότητα (responsiveness) και παραθέτουμε μία λεπτομερή εξήγηση σχετικά με την υποδομή δοκιμών. Επιπλέον, παρουσιάζουμε μία νέα διεπαφή χρήστη που έχει ως βάση τα προγράμματα περιήγησης για το Acumen, στα πλάισια της προσπάθειας ανανέωσης του παλαιότερου και ξεπερασμένου περιβάλλοντος διεπαφής. Τα αποτελέσματα που παρουσιάζονται εδώ ήταν στα πλαίσια της πρακτικής μου και πιστεύω ότι θα είναι χρήσιμη στη διαδικασία αναβάθμισης ολόκληρης της διανομής Acumen με ενημερωμένες τεχνολογίες και βιβλιοθήκες.

Α΄.1 Το Μοντέλο του ¨Ηθοποιού¨



Σχήμα 13: Στιγμιότυπα και μηνύματα ηθοποιών

Το GUI του Acumen χρησιμοποιεί ένα τρόπο υλοποίησης ταυτόχρονων υπολογισμών που είναι γνωστό ως το μοντέλο του ηθοποιού (actor model) [15]. Η αρχική υλοποίηση του μοντέλου στο Acumen δημιουργήθηκε από τον Paul Brauner [25] και ήταν δυνατή χρησιμοποιώντας τη βιβλιοθήκη Scala Actors [14]. Το μοντέλο είναι ικανό να δημιουργεί υπολογιστικές οντότητες, με

την ονομασία *ηθοποιός* (actor), που μπορούν να χρησιμοποιηθούν για τον χειρισμό διαφορετικών στοιχείων του GUI. Αυτό διασφαλίζει την σωστή ανταπόκριση μεταξύ κάθε στοιχείου του GUI καθώς και τη συμπεριφορά τους σε περίπτωση σφάλματος.

Οι διαφορετικοί actors που χρησιμοποιούνται στο Acumen καθώς και τα διάφορα μηνύματα που ανταλλάσσουν μεταξύ τους εμφανίζονται στο Σχήμα 13. Τα στιγμιότυπα των actors απεικονίζονται με στρογγυλεμένα κουτιά, τα βέλη δείχνουν τη κατεύθυνση των εκτελέσεων, οι λέξεις υποδεικνύουν λειτουργίες είτε για την αρχικοποίηση, την επέκταση ή την επίβλεψη άλλων παραγόντων ενώ λέξεις που ξεκινούν με θαυμαστικό δείχνουν τα διάφορα μηνύματα. Σε γενικές γραμμές μετά δημιουργία του, ένας actor είναι ικανός είναι ικανός να απαντά ταυτόχρονα σε ένα μήνυμα με διάφορους τρόπους:

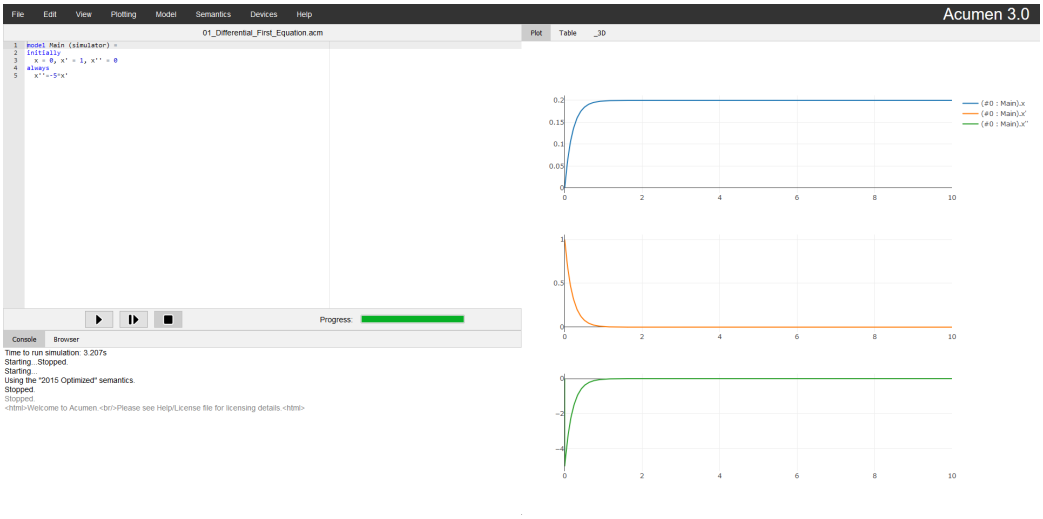
- Να στείλει μηνύματα σε άλλους ηθοποιούς.
- Να δημιουργήσει νέους ηθοποιούς.
- Να λάβει τοπικές αποφάσεις.
- Να προσδιορίσει τον τρόπο απάντησης στο επόμενο μήνυμα.

Υπάρχουν τέσσερις σημαντικοί actors που δραστηριοποιούνται κατά τη διάρκεια του χρόνου εκτέλεσης, οι ακόλουθοι: **Supervisor**, **Controller**, **Main UI** και **Plotter**. Όπως απεικονίζεται στο Σχήμα 13, ο **Supervisor** είναι υπεύθυνος για τη σωστή εκτέλεση των υπολοίπων ηθοποιών. Συχνά, οι actors είναι προγραμματισμένοι ή τείνουν να αποτυγχάνουν και ο **Supervisor** διασφαλίζει την σωστή επανεκκίνηση τους έτσι ώστε να συνεχίσουν κανονικά τον κύκλο ζωής τους.

Ο **Main UI** είναι υπεύθυνος για τον έλεγχο και την απάντηση σε ενέργειες του GUI. Αυτές οι ενέργειες περιλαμβάνουν επιλογές από το μενού και τις καρτέλες, αλλαγή μεγέθους παραθύρου και πατήματα πλήκτρων. Αναφέρει επίσης τις διάφορες καταστάσεις της εφαρμογής στην κονσόλα. Οι ενέργειες προσομοίωσης (*Play, Pause, Step, Stop*) αποστέλλονται με τη μορφή μηνυμάτων στον **Controller**. Αυτός ο **Controller** παρακολουθεί τις ενέργειες και είναι υπεύθυνος για την εκτέλεση της προσομοίωσης, ενώ ενημερώνει άλλους actors για την τρέχουσα κατάσταση της προσομοίωσης και ενημερώνοντας τη γραμμή προόδου. Για να πραγματοποιήσει την προσομοίωση, δημιουργεί έναν επιπλέον actor που ονομάζεται **Producer**, με παραμέτρους προσομοίωσης από την επιλεγμένη σημασιολογία. Ο **Producer**, ανάλογα με την κατάσταση της προσομοίωσης, αναλύει το μοντέλο από τον επεξεργαστή κειμένου και δημιουργεί την προσομοίωση. Μετά την ολοκλήρωση οποιασδήποτε προσομοίωσης αυτός ο actor τερματίζεται.

Ο τρίτος actor που είναι μέρος του μοντέλου ηθοποιού στο Acumen ονομάζεται **Plotter** και είναι υπεύθυνος για τον έλεγχο και την ενημέρωση των καρτελών *Plot* και *Table*. Αντιδρά σε μηνύματα που καθοδηγούν τη σχεδίαση, τον επανασχεδιασμό, την ανανέωση και την εκκαθάριση της καρτέλας Plot και μπορεί να αναφέρει την κατάσταση της. Τέλος, γίνεται η δημιουργία ενός actor που ελέγχει την καρτέλα *_3D*. Αυτός ονομάζεται **receiver** και αποκρίνεται σε μηνύματα χειραγώγησης σκηνής. Επιπλέον, ο **receiver** δημιουργεί έναν ακόμη actor με όνομα **timer3d**, ο οποίος μέσω της ανταλλαγής μηνυμάτων με τον **receiver** μπορεί να συντονίσει την προσομοίωση στην καρτέλα κινουμένων σχεδίων.

Α΄.2 Διεπαφή Χρήστη σε Προγράμματα Περιήγησης



Σχήμα 14: Διεπαφή χρήστη σε προγράμματα περιήγησης για το Acumen

Μέρος του σχεδίου αναβάθμισης για το Acumen ήταν η δημιουργία μιας διεπαφής που βασίζεται σε πρόγραμμα περιήγησης, η οποία είναι παρόμοια σε αισθητική με την παλαιότερη. Η διεπαφή χρήστη απεικονίζεται στο Σχήμα 14. Αποφασίσαμε να χρησιμοποιήσουμε τη λειτουργικότητα της υποδοχής sockets για να επιτύχουμε την επικοινωνία μεταξύ του Acumen και της διεπαφής. Ενώ η Scala παρέχει υποστήριξη για sockets, μια διεπαφή που βασίζεται σε πρόγραμμα περιήγησης απαιτούσε τη χρήση WebSockets για τη δημιουργία μιας αμφίδρομη συνεδρίας επικοινωνίας. Για αυτόν τον λόγο χρησιμοποιήσαμε το περιβάλλον εκτέλεσης Node.js [5] που υποστηρίζει και τις δύο τεχνολογίες ως μεσολαβητή. Δημιουργούμε δύο διακομιστές, ένα

διακομιστή socket στον οποίο συνδέεται η εφαρμογή Acumen και έναν διακομιστή WebSocket ο οποίος συνδέεται με το πρόγραμμα περιήγησης. Αυτό το μοντέλο επικοινωνίας μας επιτρέπει να στέλνουμε πλαίσια δεδομένων μέσω της υποδομής sockets του Acumen, να τα λαμβάνουμε μέσω του Node.js και να τα στέλνουμε μέσω ενός καναλιού WebSocket στη διεπαφή του προγράμματος περιήγησης.

Η διεπαφή του προγράμματος περιήγησης είναι παρόμοια με την παλιά. Χρησιμοποιήσαμε νέες βιβλιοθήκες που βασίζονται στη γλώσσα JavaScript για να μετακινήσουμε τη λειτουργικότητα όλων των διαφορετικών στοιχείων. Η βιβλιοθήκη *Ace Editor* [1] υλοποιεί τον επεξεργαστή κώδικα και υποστηρίζει την επισήμανση σύνταξης και την ολοκλήρωση κώδικα. Μια βιβλιοθήκη με το όνομα *Plotly* [6] προσθέτει εντυπωσιακές δυνατότητες σχεδίασης, με υποστήριξη τόσο για κανονικές όσο και για σχεδιάσεις με περίφραξη. Τέλος, η Sundas Munir η οποία ήταν συνάδελφος κατά τη διάρκεια της πρακτικής μου, εφάρμοσε την υποστήριξη απόδοσης 3D με τη χρήση της βιβλιοθήκης *Babylon.js* [3], με δυνατότητα οπτικοποίησης όλων των διαθέσιμων αντικειμένων σε σκηνές 3D και δημιουργίας ομαλών κινούμενων εικόνων.

Α΄.3 Υποδομή Δοκιμών

Η ύπαρξη υποδομής δοκιμών αποτελεί θεμελιώδες μέρος κατά τη διάρκεια των σταδίων ανάπτυξης οποιασδήποτε εφαρμογής. Ως ένα εργαλείο που πρέπει να είναι όσο το δυνατόν πιο αυστηρό και σωστό, το Acumen υιοθέτησε μία πολύ αυστηρή προσέγγιση όσον αφορά τις δοκιμές. Αυτή η υποδομή μπορεί να υποστηρίξει δύο διαφορετικούς τύπους δοκιμών. Πρώτον τις δοκιμές μονάδας, όπου αποδεικνύονται χρήσιμες για τον έλεγχο της συμπεριφοράς των συναρτήσεων, εντοπίζοντας πιθανά ζητήματα καθώς και τυχόν τυχαίες αλλαγές στη σημασιολογία. Αυτού του είδους οι δοκιμές πραγματοποιούνται μέσω της χρήσης της βιβλιοθήκης ScalaTest [9] και κατηγοριοποιούνται σε δοκιμαστικές σουίτες. Δεύτερον, οι δοκιμές βάσει ιδιοτήτων [26] είναι επίσης διαθέσιμες, όπου οι δοκιμές χρησιμοποιούνται για επικύρωση βασικών ιδιοτήτων της αριθμητικής. Η υλοποίηση των δοκιμών βάσει ιδιοτήτων κατέστη δυνατή μέσω της βιβλιοθήκης ScalaCheck [8]. Όλα τα είδη δοκιμών μπορούν είτε να εκτελεστούν μεμονωμένα, όπου επαληθεύουν μόνο πολύ συγκεκριμένα τμήματα του κώδικα ή ως δοκιμαστικές σουίτες που ελέγχουν τη συμπεριφορά διαφορετικών εσωτερικών λειτουργιών.

Κατά την ανάπτυξη των αλλαγών κώδικα σε αυτή την εργασία, η εκτέλεση της δοκιμαστικής υποδομής ήταν ένα απαραίτητο βήμα για να επαληθεύσουμε τη σωστή λειτουργία του Acumen. Ωστόσο, υπήρχαν ορισμένα προβλήματα κατά την προσπάθεια πρόσβασης σε τοπικά αρχεία χρησιμοποιώντας διαδρομές. Κάθε λειτουργικό σύστημα χρησιμοποιεί διαφορετικό

χαρακτήρα για να διαχωρίσει αρχεία και φακέλους. Για την επίλυση αυτού του ζητήματος, τροποποιήσαμε όλες τις δοκιμαστικές σουίτες ώστε να χρησιμοποιούν ένα δυναμικό διαχωριστικό, μια λειτουργία που προσδιορίζει το είδος του λειτουργικού συστήματος που χρησιμοποιείται και εφαρμόζει έναν σωστό διαχωριστικό χαρακτήρα.

Β΄ Επανεκκίνηση Και Αρχική Κυκλοφορία

Καθ' όλη την ανάπτυξη του Acumen, υπήρξαν πολλές δραστηριότητες που συνέβαλαν στη διανομή [25]. Κατά τη διάρκεια της μελέτης μας υπήρξε η επιθυμία για ένα ¨σταθερό¨ σημείο εκκίνησης, καθώς η ανάπτυξη της γλώσσας Acumen είχε ένα σύντομο διάλειμμα από το 2017 έως πρόσφατα και ως εκ τούτου περιέχει πλέον ξεπερασμένες βιβλιοθήκες. Για αυτόν ακριβώς τον λόγο πραγματοποιήσαμε μία έρευνα σχετικά με την ιστορία της γλώσσας προκειμένου να εντοπίσουμε ένα ¨καλό¨ σημείο εκκίνησης. Τα κύρια σημεία ενδιαφέροντος που ακολουθήσαμε είναι η λειτουργικότητα συλλογών, οι περιπτώσεις χρήσης (π.χ. διδασκαλία, βιβλία, κ.α.) και η ταχύτητα προσομοίωσης. Αυτά τα σημεία μείωσαν την αναζήτηση σε δύο επιλογές εκδόσεων: Μάρτιος 2016 και Αύγουστος 2016.

Αναθέσεις	**Κυκλοφορία Μαρτίου**	**Κυκλοφορία Αυγούστου**
Ολόκληρη συλλογή (**Σ**)	✓	✓
Ολόκληρη συλλογή (**Δ**)	✓	✓
Όλα τα στοιχεία με επανάληψη (**Σ**)	✗	✗
Όλα τα στοιχεία με επανάληψη (**Δ**)	✓	✗
Όλα τα στοιχεία με επανάληψη και length (**Σ**)	✗	✗
Όλα τα στοιχεία με επανάληψη και length (**Δ**)	✓	✓
Μεμονωμένα στοιχεία (**Σ**)	✗	✗
Μεμονωμένα στοιχεία (**Δ**)	✓	✗

Πίνακας 10: Ανάθεσεις συλλογών στις κυκλοφορίες Μαρτίου και Αυγούστου

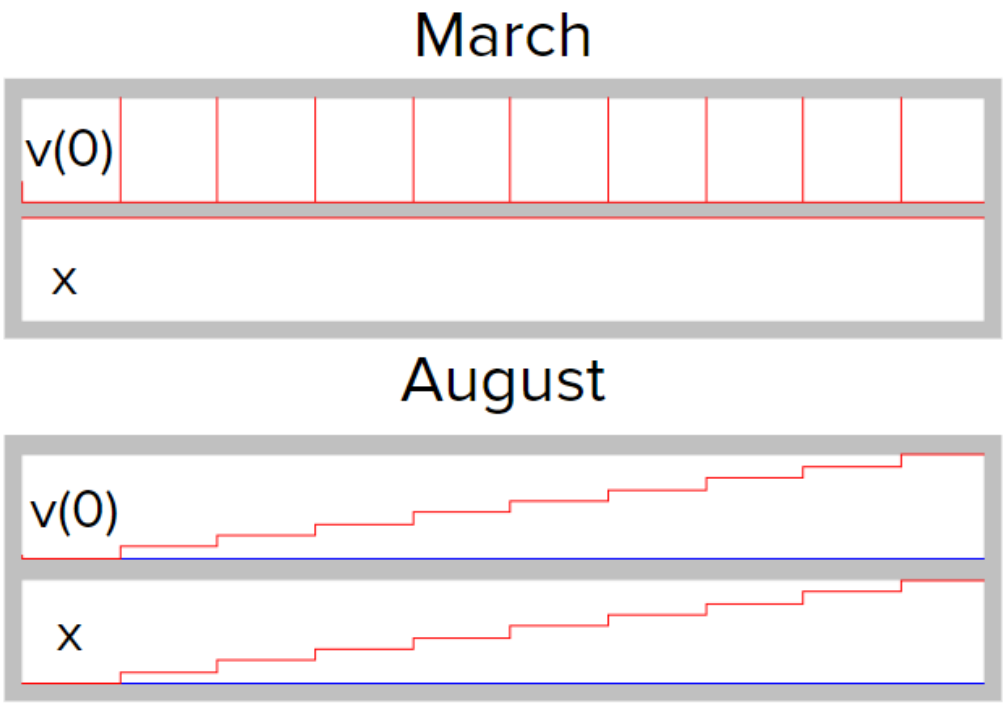
Και στις δύο εκδόσεις οι αρχές με τις οποίες δουλεύουν οι αναθέσεις συλλογών, διαφέρουν με διάφορους τρόπους. Ο Πίνακας 10 απεικονίζει τις διαφορές μεταξύ των δύο επιλεγμένων κυκλοφοριών όσον αφορά τις αναθέσεις λίστας. Κάνουμε χρήση των γραμμάτων ¨**Δ**¨ και ¨**Σ**¨ για αναφορά σε διακριτές

και συνεχείς αναθέσεις αντίστοιχα. Οι αναθέσεις σε ολόκληρες συλλογές δεδομένων σημαίνουν την τροποποίηση ολόκληρης της συλλογής ταυτόχρονα, για παράδειγμα x=(0,1), ενώ οι αναθέσεις σε όλα τα στοιχεία γίνονται με δηλώσεις επανάληψης χρησιμοποιώντας τη λέξη-κλειδί *foreach* και ένα εύρος. Είναι χρήσιμο να παρατηρήσουμε εδώ ότι υπάρχουν επίσης πρόσθετες επιλογές στις δηλώσεις επανάληψης χρησιμοποιώντας τη λέξη-κλειδί *length* για την εύρεση του μήκους μιας συλλογής δεδομένων.

Στην Ενότητα 3.2.1 έχουμε ήδη εξηγήσει πώς η προσθήκη του περάσματος BTA εισήγαγε ζητήματα κατά την προσπάθεια είτε διακριτών είτε συνεχων αναθέσεων σε μια συλλογή. Υπάρχει όμως μία μοναδική αρχή στην κυκλοφορία του Αυγούστου που τράβηξε την προσοχή μας, με τη χρήση διακριτών αναθέσεων σε όλα τα στοιχεία και χρησιμοποιώντας επανάληψη με τη λέξη-κλειδί *length*. Ο Πίνακα 10 επιβεβαιώνει αυτήν την ανακάλυψη και δείχνει τις τροποποιημένες αρχές συμπεριφοράς σε αναθέσεις λιστών. Αν και αυτή η συμπεριφορά αποδείχθηκε εξαιρετικά περιορισμένη στην κυκλοφορία του Αυγούστου 2015, ένας εναλλακτικός τρόπος αντιμετώπισης τέτοιων λιστών είναι με τη χρήση προκαθορισμένων μεταβλητών και την εκχώρησή τους ως στοιχεία των συλλογών. Εξετάστε το ακόλουθο μοντέλο σε Acumen:

```
model Main(simulator) =
initially
  v=(1,2), x=0, y=0,
  t=0, t'=0
always
  t'=1, v=(x,y),
  if t>1 then
    v(0)+=v(0)+5,
    t+=0
  noelse
```

Σε αυτό το παράδειγμα πραγματοποιούμε συνεχή ανάθεση των μεταβλητών x και y στη λίστα v. Αυτό επιτρέπει να παρακάμψουμε τους περιορισμούς από την κυκλοφορία του Αυγούστου και να κάνουμε διακριτές αναθέσεις σε τέτοιες λίστες. Ωστόσο η εκτέλεση αυτού του μοντέλου με τις δύο εκδόσεις αποφέρει πολύ διαφορετικά αποτελέσματα όπως φαίνεται στο Σχήμα 15. Η κυκλοφορία του Αυγούστου 2015 φαίνεται να διαχειρίζεται καλύτερα τις αναθέσεις σε αυτήν τη λίστα. Από την άλλη πλευρά στην έκδοση Μαρτίου 2015 τέτοιες μεταβλητές εκχωρούνται στιγμιαία για ένα και μόνο χρονικό βήμα της προσομοίωσης. Επιπρόσθετα, το πέρασμα BTA εισήγαγε επίσης τις λέξεις-κλειδιά *ones* και *zeros* στη γλώσσα για τη δημιουργία ν-διαστάσεων συλλογών δεδομένων. Η επιλογή της έκδοσης Αυγούστου 2015 θεωρήθηκε ως πιο κατάλληλη για τέτοιες λειτουργίες.



Σχήμα 15: Διακριτές αναθέσεις στις εκδόσεις Μαρτίου και Αυγούστου (2015)

Η κυκλοφορία του Αυγούστου 2015 είναι η πιο χρησιμοποιημένη διανομή του Acumen μέχρι στιγμής και είναι πολύ καλά δοκιμασμένη. Έχει χρησιμοποιηθεί για την διδασκαλία CPS [24, 23, 27] και εμπεριέχεται επίσης σε ένα υπό έκδοση βιβλίο για τα CPS [31]. Συνοψίζοντας, επιλέξαμε την κυκλοφορία του Αυγούστου 2015 ως την έκδοση στην οποία θα εφαρμόσουμε όλες τις τροποποιήσεις στο κώδικα του Acumen.

Επιπρόσθετα, καταβλήθηκε σημαντική προσπάθεια για τη μετακίνηση της επιλεγμένης αρχικής έκδοσης σε ένα κεντρικό αποθετήριο git. Ως σελίδα φιλοξενίας επιλέξαμε να μετακινηθούμε από το Bitbucket της Atlassian στο Github. Κάναμε τον κύριο κλάδο (branch) να δείχνει την έκδοση του Αυγούστου και μετακινήσαμε οποιαδήποτε εργασία μετά από αυτήν την ημερομηνία σε ένα νέο κλαδί με το όνομα master-until-2017 ως αντίγραφο ασφαλείας. Η μελέτη, οι αλλαγές σε κώδικα και το παράδειγμα της RL που παρουσιάστηκε στην υποενότητα 5.2 βρίσκονται κάτω από τον κλάδο vectors. Τα προβλήματα διαδρομής έχουν επίσης εξαλειφθεί καθώς εμπόδιζαν τους χρήστες Windows OS να δουλέψουν στο αποθετήριο.

Αναφορές

[1] Ace javascript editor. https://ace.c9.io/. Accessed: 2020-08-27.

[2] Acumen modeling language. http://acumen-language.org. Accessed: 2020-07-29.

[3] Babylon web rendering engine. https://www.babylonjs.com/. Accessed: 2020-08-27.

[4] jpct 3d engine. http://www.jpct.net//. Accessed: 2020-08-07.

[5] Node.js runtime. https://nodejs.org/en/. Accessed: 2020-08-27.

[6] Plotly javascript graphing library. https://plotly.com/javascript/. Accessed: 2020-08-27.

[7] Rsyntaxtextarea. http://bobbylight.github.io/RSyntaxTextArea. Accessed: 2020-07-29.

[8] Scalacheck property-based testing tool. https://www.scalacheck.org/. Accessed: 2020-07-29.

[9] Scalatest testing tool. https://www.scalatest.org/. Accessed: 2020-07-29.

[10] Ken Arnold, James Gosling, and David Holmes. *The Java programming language*. Addison Wesley Professional, 2005.

[11] Hong Chen. Applications of cyber-physical system: A literature review. *Journal of Industrial Integration and Management*, 02:1750012, 10 2017.

[12] Adam Duracz. *Rigorous simulation: its theory and applications*. PhD thesis, Halmstad University Press, 2016.

[13] Eyal Even-Dar and Yishay Mansour. Learning rates for q-learning. *Journal of machine learning Research*, 5(Dec):1–25, 2003.

[14] Philipp Haller and Martin Odersky. Scala actors: Unifying thread-based and event-based programming. *Theoretical Computer Science*, 410(2-3):202–220, 2009.

[15] Carl Hewitt, Peter Bishop, and Richard Steiger. Session 8 formalisms for artificial intelligence a universal modular actor formalism for artificial intelligence. In *Advance Papers of the Conference*, volume 3, page 235. Stanford Research Institute, 1973.

[16] Leslie Pack Kaelbling, Michael L Littman, and Andrew W Moore. Reinforcement learning: A survey. *Journal of artificial intelligence research*, 4:237–285, 1996.

[17] Tambet Matiisen. Demystifying deep reinforcement learning. *Computational Neuroscience LAB*, 19, 2015.

[18] Francisco S Melo. Convergence of q-learning: A simple proof. *Institute Of Systems and Robotics, Tech. Rep*, pages 1–4, 2001.

[19] Adriaan Moors, Frank Piessens, and Martin Odersky. Parser combinators in scala. *CW Reports*, 2008.

[20] Martin Odersky, Lex Spoon, and Bill Venners. *Programming in scala*. Artima Inc, 2008.

[21] Travis E Oliphant. *A guide to NumPy*, volume 1. Trelgol Publishing USA, 2006.

[22] John W Shipman. Tkinter 8.4 reference: a gui for python. *New Mexico Tech Computer Center*, 2013.

[23] Walid Taha, Robert Cartwright, Roland Philippsen, and Yingfu Zeng. Experiences with a first course on cyber-physical systems. In *2013 Workshop on Embedded and Cyber-Physical Systems Education (WESE), Montreal, Canada*, 2013.

[24] Walid Taha, Robert Cartwright, Roland Philippsen, and Yingfu Zeng. A first course on cyber physical systems. In *Workshop on Cyber-Physical Systems Education (CPS-Ed)*, 2013.

[25] Walid Taha, Adam Duracz, Yingfu Zeng, Kevin Atkinson, Ferenc A Bartha, Paul Brauner, Jan Duracz, Fei Xu, Robert Cartwright, Michal Konečnỳ, et al. Acumen: An open-source testbed for cyber-physical systems research. In *International Internet of Things Summit*, pages 118–130. Springer, 2015.

[26] Walid Taha, Veronica Gaspes, and Rex Page. Accurate programming: Thinking about programs in terms of properties. *arXiv preprint arXiv:1109.0786*, 2011.

[27] Walid Taha, Yingfu Zeng, Adam Duracz, Xu Fei, Kevin Atkinson, Paul Brauner, Robert Cartwright, and Roland Philippsen. Developing a first course on cyber-physical systems. *ACM SIGBED Review*, 14(1):44–52, 2017.

[28] Michel Tokic and Günther Palm. Value-difference based exploration: adaptive control between epsilon-greedy and softmax. In *Annual Conference on Artificial Intelligence*, pages 335–346. Springer, 2011.

[29] Martijn Van Otterlo and Marco Wiering. Reinforcement learning and markov decision processes. In *Reinforcement Learning*, pages 3–42. Springer, 2012.

[30] Guido Van Rossum and Fred L. Drake. *Python 3 Reference Manual*. CreateSpace, Scotts Valley, CA, 2009.

[31] Johan Thunberg Walid Taha, Abd-Elhamid M. Taha. *Cyber-Physical Systems: A Model-Based Approach*. Springer International Publishing, 2021.

[32] Christopher John Cornish Hellaby Watkins. Learning from delayed rewards. 1989.

[33] Fei Xu. Lightweight immersion techniques for acumen, 2015.

[34] Yingfu Zeng. Lightweight three-dimensional visualization for hybrid systems simulation. *Master's thesis, Halmstad University, Halmstad*, 2012.

[35] Yingfu Zeng. *Making Hybrid Systems Easier to Model, Simulate, and Visualize*. PhD thesis, 2019.

[36] Yun Zhu, Edwin Westbrook, Jun Inoue, Alexandre Chapoutot, Cherif Salama, Marisa Peralta, Travis Martin, Walid Taha, Marcia O'Malley, Robert Cartwright, et al. Mathematical equations as executable models of mechanical systems. In *Proceedings of the 1st ACM/IEEE International Conference on Cyber-Physical Systems*, pages 1–11, 2010.