ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Ανάπτυξη και Σχεδίαση Παιχνιδιών Ανοιχτού Κώδικα με την Χρήση του Pygame

Εμμανουήλ Σπανός

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

Αντώνιος Δαδαλιάρης
Επίκουρος Καθηγητής

Λαμία, 2024

ΠΑΝΕΠΙΣΤΗΜΙΟ
ΘΕΣΣΑΛΙΑΣ

ΣΧΟΛΗ ΘΕΤΙΚΩΝ ΕΠΙΣΤΗΜΩΝ

ΤΜΗΜΑ ΠΛΗΡΟΦΟΡΙΚΗΣ ΚΑΙ ΤΗΛΕΠΙΚΟΙΝΩΝΙΩΝ

Ανάπτυξη και Σχεδίαση Παιχνιδιών Ανοιχτού Κώδικα με την Χρήση του Pygame

Εμμανουήλ Σπανός

ΠΤΥΧΙΑΚΗ ΕΡΓΑΣΙΑ

ΥΠΕΥΘΥΝΟΣ

Αντώνιος Δαδαλιάρης
Επίκουρος Καθηγητής

Λαμία, 2024

UNIVERSITY OF
THESSALY

SCHOOL OF SCIENCE

DEPARTMENT OF COMPUTER SCIENCE & TELECOMMUNICATIONS

Developing and Designing Open Source Games Using Pygame

Emmanouil Spanos

FINAL THESIS

ADVISOR

Antonios Dadaliaris
Assistant Professor

Lamia, 2024

«Με ατομική μου ευθύνη και γνωρίζοντας τις κυρώσεις [(1)], που προβλέπονται από της διατάξεις της παρ. 6 του άρθρου 22 του Ν. 1599/1986, δηλώνω ότι:

*1. Δεν παραθέτω κομμάτια βιβλίων ή άρθρων ή εργασιών άλλων αυτολεξεί **χωρίς να τα περικλείω σε εισαγωγικά** και χωρίς να αναφέρω το συγγραφέα, τη χρονολογία, τη σελίδα. Η αυτολεξεί παράθεση χωρίς εισαγωγικά χωρίς αναφορά στην πηγή, είναι λογοκλοπή. Πέραν της αυτολεξεί παράθεσης, λογοκλοπή θεωρείται και η παράφραση εδαφίων από έργα άλλων, συμπεριλαμβανομένων και έργων συμφοιτητών μου, καθώς και η παράθεση στοιχείων που άλλοι συνέλεξαν ή επεξεργάσθηκαν, χωρίς αναφορά στην πηγή. Αναφέρω πάντοτε με πληρότητα την πηγή κάτω από τον πίνακα ή σχέδιο, όπως στα παραθέματα.*

*2. Δέχομαι ότι η αυτολεξεί **παράθεση χωρίς εισαγωγικά**, ακόμα κι αν συνοδεύεται από αναφορά στην πηγή σε κάποιο άλλο σημείο του κειμένου ή στο τέλος του, είναι αντιγραφή. Η αναφορά στην πηγή στο τέλος π.χ. μιας παραγράφου ή μιας σελίδας, δεν δικαιολογεί συρραφή εδαφίων έργου άλλου συγγραφέα, έστω και παραφρασμένων, και παρουσίασή τους ως δική μου εργασία.*

3. Δέχομαι ότι υπάρχει επίσης περιορισμός στο μέγεθος και στη συχνότητα των παραθεμάτων που μπορώ να εντάξω στην εργασία μου εντός εισαγωγικών. Κάθε μεγάλο παράθεμα (π.χ. σε πίνακα ή πλαίσιο, κλπ), προϋποθέτει ειδικές ρυθμίσεις, και όταν δημοσιεύεται προϋποθέτει την άδεια του συγγραφέα ή του εκδότη. Το ίδιο και οι πίνακες και τα σχέδια

4. Δέχομαι όλες τις συνέπειες σε περίπτωση λογοκλοπής ή αντιγραφής.

Ημερομηνία: ……/…../20……

Ο – Η Δηλ.

(1) «Όποιος εν γνώσει του δηλώνει ψευδή γεγονότα ή αρνείται ή αποκρύπτει τα αληθινά με έγγραφη υπεύθυνη δήλωση
του άρθρου 8 παρ. 4 Ν. 1599/1986 τιμωρείται με φυλάκιση τουλάχιστον τριών μηνών. Εάν ο υπαίτιος αυτών των πράξεων
σκόπευε να προσπορίσει στον εαυτόν του ή σε άλλον περιουσιακό όφελος βλάπτοντας τρίτον ή σκόπευε να βλάψει άλλον, τιμωρείται με κάθειρξη μέχρι 10 ετών.»

ΠΕΡΙΛΗΨΗ

Η παρούσα πτυχιακή αποσκοπεί στην παρουσίαση της ροής υλοποίησης ενός βιντεοπαιχνιδιού με τη χρήση της γλώσσας προγραμματισμού Python και την βιβλιοθήκη Pygame. Αρχικά η εργασία κάνει μια αναφορά στον χώρο των βιντεοπαιχνιδιών, από την απαρχή τους μέχρι και το σήμερα. Παραθέτει οικονομικά στοιχεία που αιτιολογούν την σημαντικότητας της για την παγκόσμια αγορά, ενώ σκιαγραφούνται και οι τρόποι που τα βιντεοπαιχνίδια αλληλεπιδρούν με άλλους επιστημονικούς χώρους. Στη συνέχεια παρουσιάζεται η γλώσσα Python και η βιβλιοθήκη Pygame, ενώ ακολουθεί ο σχεδιασμός των παιχνιδιών και η υλοποίησή τους. Ολοκληρώνοντας την ανάλυση της σχεδίασης και υλοποίησης των παιχνιδιών, παρουσιάζονται τα συμπεράσματα που αντλήθηκαν από την ολοκλήρωση της εκπόνησής της.

ABSTRACT

This bachelor thesis aims to present the implementation flow of a video game using the Python programming language and the Pygame library. Initially the thesis makes a reference to the field of video games, from their inception to the present day. It provides economic data that justifies its importance to the global market and outlines the ways in which video games interact with other disciplines. The Python language and the Pygame library are then introduced, followed by the design of the games and their implementation. Having completed the analysis of the game design and implementation, the conclusions drawn from the completion of the project are presented.

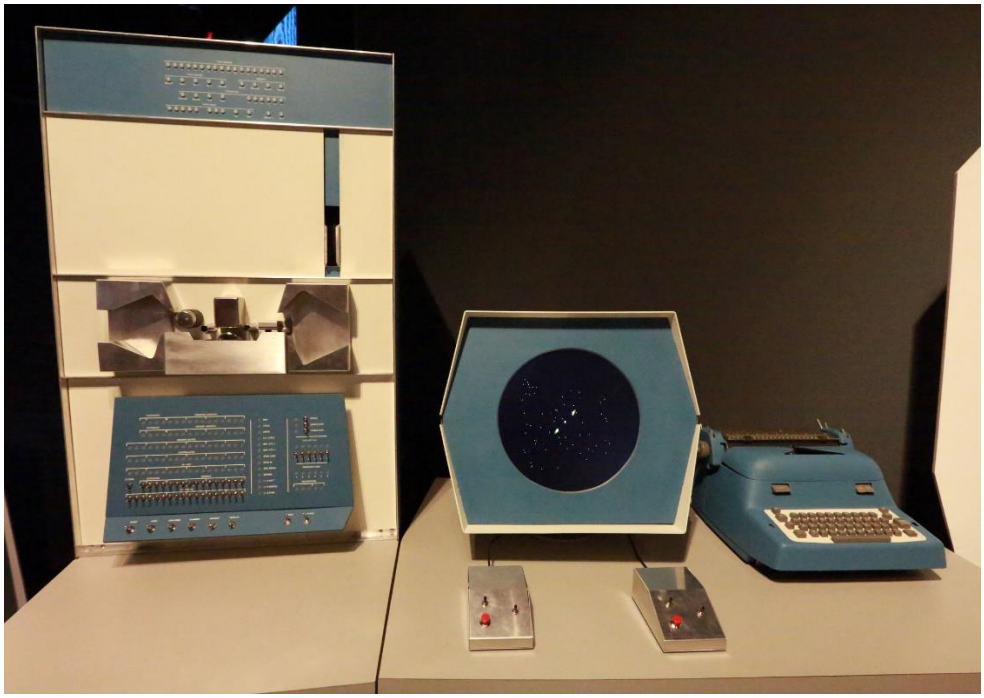
Table of Contents

ΚΕΦΑΛΑΙΟ 1 Εισαγωγή

1.1 Ο χώρος του gaming

1.1.a Η Ιστορία του Gaming

Το πρώτο βιντεοπαιχνίδι (video game) ονόματι «Spacewar!» (βλ. Εικόνα 1) υλοποιήθηκε από την ομάδα του Steve Russel στο MIT το 1962, και αναγνωρίζεται ευρέως ως ένα από τα πρώτα ψηφιακά ηλεκτρονικά παιχνίδια [1]. Το παιχνίδι εκτελούταν στο υπολογιστικό σύστημα PDP-1 (βλ. Εικόνα 1). Προχωρώντας στην δεκαετία του ’70, τα λεγόμενα «arcade machines» έκαναν τις πρώτες εμφανίσεις τους στα γνωστά «Arcades» της εποχής με το «Computer Space» των Nolan Bushnell και Ted Dabney να κάνει την εμφάνισή του ως το πρώτο «arcade cabinet» το 1971. Εμπνευσμένο από το προαναφερθέν «Spacewar!», στο «Computer Space» ο παίκτης παίρνει τον έλεγχο ενός πυραύλου και πολεμά με UFO, με τελικό στόχο να κερδίσει τους περισσότερους πόντους. Η συλλογή πόντων εκτελούταν με την εξολόθρευση των UFO, ενώ παράλληλα αυτά ελέγχονται από την μηχανή και έχουν στόχο να εξολοθρεύσουν, με τη σειρά τους, τον παίκτη.



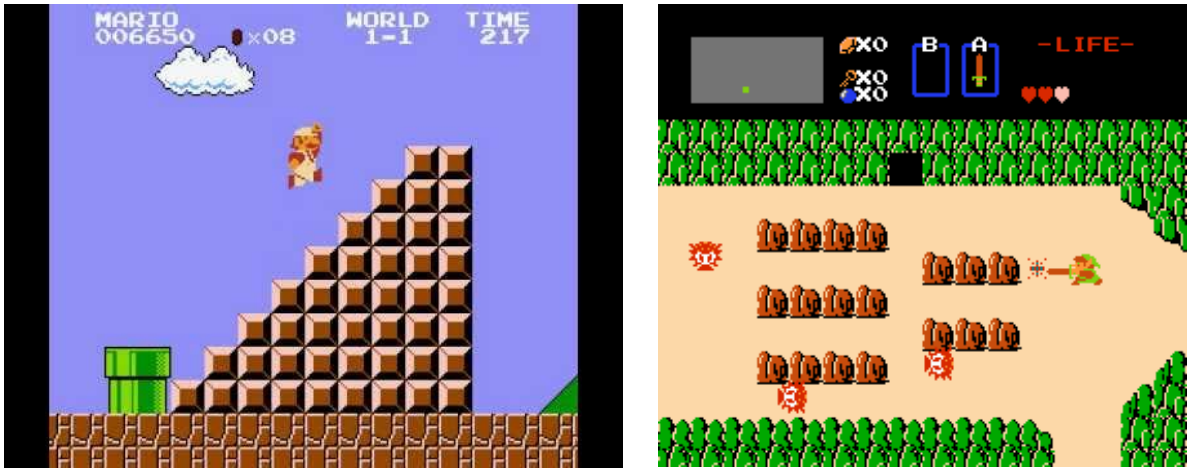
Εικόνα 1: Φωτογραφία του παιχνιδιού "Spacewar!". Πηγή: https://www.nytimes.com/2012/12/15/arts/video-games/spacewar-video-games-at-museum-of-the-moving-image.html

Η εκκίνηση της βιομηχανίας των video games ή gaming industry έγινε το 1972 με την ίδρυση της εταιρίας Atari [2] από τους Bushnell και Dabney, και την κυκλοφορία του δημοφιλούς παιχνιδιού «Pong». Το «Pong» αποτελούσε ένα απλοποιημένο παιχνίδι

επιτραπέζιου τένις (ping-pong). Τα video games όμως δεν θα είχαν φτάσει στο μέγεθος βιομηχανίας που είναι σήμερα χωρίς το όραμα του Ralph H. Baer και της ομάδας του, που τον Σεπτέμβριο του 1972 κυκλοφόρησαν το «Magnavox Odyssey» (βλ. Εικόνα 2), την πρώτη οικιακή κονσόλα παιχνιδιών, με τίτλους όπως το «Table Tennis», «Ski», «Soccer» και «Dogfight» μεταξύ άλλων [3].

Εικόνα 2: Η κονσόλα Odyssey με το χαρακτηριστικό αναλογικό χειριστήριο. Πηγή: https://el.wikipedia.org/wiki/Magnavox_Odyssey.

Το 1980 οι πρώτοι οικιακοί υπολογιστές (Apple II, Commodore 64, IBM PCs, κ.α.) αποτέλεσαν πλατφόρμες για την ανάπτυξη πιο σύνθετων και πολύπλοκων παιχνιδιών. Μερικοί από τους τίτλους που κυκλοφόρησαν για το Apple II είναι τα «Pinball Construction Set» (1983), «Ultima 1: The First Age of Darkness» (1981), και «Prince of Persia» (1989), ενώ για το Commodore 64 είναι τα «International Soccer» (1983), «Impossible Mission» (1984) και «Ghostbusters» 1984. Από το 1983 μέχρι το 1985 η βιομηχανία υπέστη κρίση λόγο κορεσμού της αγοράς παιχνιδιών χαμηλής ποιότητας. Την κρίση αυτή έσπασε η Nintendo [4] το 1985 με την κυκλοφορία του Nintendo Entertainment System (NES) και τους εμβληματικούς τίτλους των «Super Mario Bros.» (1985, Εικόνα 3), και «The Legend of Zelda» (1986, Εικόνα 3).



Εικόνα 3: Στιγμιότυπο gameplay από το παιχνίδια Super Mario Bros. (1985) αριστερά και Legend of Zelda (1986) δεξιά. Πηγή: https://en.wikipedia.org/wiki/The_Legend_of_Zelda_%28video_game%29.

Μετά την επανάσταση που ξεκίνησε η Nintendo, έγιναν προφανή τα οικονομικά πλεονεκτήματα της διαδραστικής διασκέδασης σε ψηφιακά περιβάλλοντα, συνεπώς ήταν θέμα χρόνου οι κατασκευαστές κονσόλων και οι σχεδιαστές παιχνιδιών να ξεκινήσουν τον αγώνα για καινοτομία, με το επόμενο μεγαλύτερο άλμα να είναι η χρήση τρισδιάστατων γραφικών (3D Graphics). Έτσι, την δεκαετία του ΄90, οι πρόοδοι στα υπολογιστικά συστήματα, όπως η αύξηση της υπολογιστικής ισχύος, του αποθηκευτικού χώρου και του μεγέθους της κύριας μνήμης επεξεργασίας επέτρεψαν την υλοποίηση πιο εντυπωσιακών και ελκυστικών παιχνιδιών [5]. Η αντικατάσταση των μαγνητικών δίσκων (δισκέτες) από τα CDs και τα DVDs αποτέλεσε επίσης μεγάλο άλμα, τόσο στην κατασκευαστική διαδικασία των παιχνιδιών, όσο και στο μέγεθος των δεδομένων που μπορούσαν να αποθηκευτούν. Η εξέλιξη αυτή οδήγησε στην εμφάνιση κονσόλων που έχουν αφήσει το αποτύπωμά τους στην ιστορία, όπως το PlayStation της SONY [6] και το Xbox της Microsoft [7] (βλ. Εικόνα 4).

Εικόνα 4: Κονσόλες Sony PlayStation (αριστερά) και Microsoft Xbox (δεξιά).

Σήμερα, ο χώρος του gaming έχει αποκτήσει τεράστιες διαστάσεις, με υπέρογκες επενδύσεις για ανάπτυξη και σχεδίαση καινοτόμων τεχνολογιών και εμπειριών. Ο χώρος της εικονικής και επαυξημένης πραγματικότητας (VR/AR) επέτρεψε την υλοποίηση νέων περιφερειακών συσκευών (π.χ. Meta Quest [8], PlayStation VR, κ.α.) οι οποίες έφεραν επανάσταση στον τρόπο με τον οποίο οι παίκτες απολάμβαναν τα παιχνίδια (βλ. Εικόνα 5). Επιπλέον, η εξέλιξη των κινητών συσκευών οδήγησε στην ανάπτυξη παιχνιδιών κινητών συσκευών (mobile games), τα οποία σε συνδυασμό με τις τεχνολογίες κατανεμημένων συστημάτων (π.χ. cloud), καταλαμβάνουν ένα σημαντικό ποσοστό του gaming [9].

Εικόνα 5: Στιγμιότυπο gameplay από το παιχνίδι Horizon Call of the Mountain στο PlayStation VR2.

1.1.β Οικονομικά στοιχεία

Το 2023, η Entertainment Software Association (ESA) [10] ανέθεσε στην TEConomy Partners, LLC [11] τη διεξαγωγή μιας επικαιροποιημένης και ολοκληρωμένης αξιολόγησης των οικονομικών επιπτώσεων της βιομηχανίας βιντεοπαιχνιδιών στις Ηνωμένες Πολιτείες. Η ανάλυση αυτή κατηγοριοποιεί τη βιομηχανία βιντεοπαιχνιδιών των ΗΠΑ σε πέντε τομείς, που περιλαμβάνουν πάνω από 104.000 εργαζόμενους. Ο εργασιακός τομέας που βασίζεται κυρίως από τη δημιουργία λογισμικού βιντεοπαιχνιδιών απασχολεί περισσότερα από 75.000 άτομα [12].

Η βιομηχανία βιντεοπαιχνιδιών στις ΗΠΑ έχει σημαντική οικονομική επιρροή, καθώς έχει δημιουργήσει και διατηρήσει πάνω από 350.000 θέσεις εργασίας σε ολόκληρη την οικονομία, ενώ συνεισφέρει επίσης πάνω από 101 δισεκατομμύρια δολάρια σε συνολικές οικονομικές επιπτώσεις. Από αυτά, ο κλάδος αντιπροσώπευε σχεδόν 66 δισεκατομμύρια δολάρια του ΑΕΠ των ΗΠΑ το 2023 [12].

Το συνολικό εισόδημα που δημιουργήθηκε για τους εργαζόμενους στις ΗΠΑ σε όλους τους τομείς της οικονομίας ανήλθε σε 35,2 δισεκατομμύρια δολάρια, το οποίο περιλαμβάνει 17,6 δισεκατομμύρια δολάρια σε άμεσο εισόδημα από την εργασία ειδικά για τους εργαζόμενους στη βιομηχανία βιντεοπαιχνιδιών. Αυτό μεταφράζεται σε μια μέση συνολική αποζημίωση ύψους 168.627 δολαρίων ανά εργαζόμενο του κλάδου. Επιπλέον, ο κλάδος συνεισέφερε 14,4 δισεκατομμύρια δολάρια σε φόρους, με 9,5 δισεκατομμύρια δολάρια να πηγαίνουν στην ομοσπονδιακή κυβέρνηση, 3,2 δισεκατομμύρια δολάρια στις πολιτειακές κυβερνήσεις και 1,6 δισεκατομμύρια δολάρια στις τοπικές και νομαρχιακές κυβερνήσεις [12].

Επιπλέον από την έκθεση των Hong et al. [9], παρατηρήθηκε πως αν και το mobile gaming αποτελεί την πιο προσιτή λύση για το ευρύτερο κοινό, εμφανίζει τα μεγαλύτερα ποσοστά εξόδων ως προς τα συνολικά έξοδο, αγγίζοντας σχεδόν το 25%, ενώ εμφανίζει και το μεγαλύτερο ρίσκο για αποτυχία, με πιθανότητα αποτυχίας μεταξύ 55-70%. Αντιθέτως, τα παιχνίδια που υλοποιούνται ψηφιακά, χωρίς φυσική υπόσταση, και διανέμονται από κάποια πλατφόρμα (π.χ. PS Store [13], Valve Steam [14], Epic Games [15], κ.α.), εμφανίζουν πιθανότητα αποτυχίας μεταξύ του 10% και 20%. Συνολικά, η βιομηχανία των παιχνιδιών με τη μορφή λογισμικού εμφανίζει την μικρότερη πιθανότητα αποτυχίας. Με βάση την έκθεση, η βιομηχανία αναμένεται να αγγίξει το συνολικό ποσό των 216 δισεκατομμυρίων δολαρίων σε έσοδα παγκοσμίως μέχρι το τέλος του 2025. Ήδη τα παγκόσμια έσοδα της βιομηχανίας

παιχνιδιών αγγίζουν τα 204 δισεκατομμύρια, ενώ με πρόβλεψη της αγοράς, αναμένονται έσοδα σε ύψος 257 δισεκατομμυρίων δολαρίων παγκοσμίως μέχρι το 2028.

1.2 Εργαλεία Ανάπτυξης

Μαζί με την πορεία εξέλιξης των παιχνιδιών, παράλληλα έπρεπε να αλλάζουν και οι τρόποι υλοποίησής τους. Ανάλογα με τις απαιτήσεις του εκάστοτε παιχνιδιού, πρέπει να υπάρχει και ανάλογη χρήση τεχνολογιών, κατάλληλης πολυπλοκότητας για την επίτευξη της επιθυμητής λειτουργίας. Τα πρώτα παιχνίδια όπως το «Spacewar!» και τα παιχνίδια της κονσόλας Atari 2600 [2] (βλ. Εικόνα 6) ήταν τυπικά γραμμένα σε συμβολική γλώσσα Assembly. Η Assembly είχε την ικανότητα να βελτιστοποιεί τη χρήση των περιορισμένων πόρων υλικού των πρώτων υπολογιστικών συστημάτων και κονσόλων. Η γλώσσα BASIC ήταν επίσης δημοφιλής την δεκαετία του ’70 και του ’80, καθώς ήταν προσιτή και πιο εύκολη στην εκμάθηση για χομπίστες και αρχάριους προγραμματιστές.

Εικόνα 6: Κονσόλα παιχνιδιών Atari 2600. Πηγή: https://en.wikipedia.org/wiki/Atari_2600.

Με την πρόοδο της τεχνολογίας, η γλώσσα προγραμματισμού C προτιμήθηκε περισσότερο, ειδικότερα στις δεκαετίας ’80 και ’90, λόγω της ισορροπίας μεταξύ απόδοσης και υψηλότερων επιδόσεων αφαίρεσης. Η C χρησιμοποιήθηκε για παιχνίδια σε πλατφόρμες όπως το NES, το SNES και οι πρώιμοι προσωπικοί υπολογιστές. Με την σειρά της η C++ εξελίχθηκε σε μια από τις ευρέως χρησιμοποιούμενες γλώσσες στην ανάπτυξη παιχνιδιών, παρέχοντας μία καλή ισορροπία απόδοσης και αντικειμενοστραφών χαρακτηριστικών. Η πιο γνωστή χρήση της γλώσσας είναι η λειτουργία της ως βάση για τις μηχανές ανάπτυξης παιχνιδιών (game engines) Unreal (βλ. Εικόνα 7) και Unity [16].

Εικόνα 7: Στιγμιότυπο χρήσης της μηχανής παιχνιδιών Unreal.

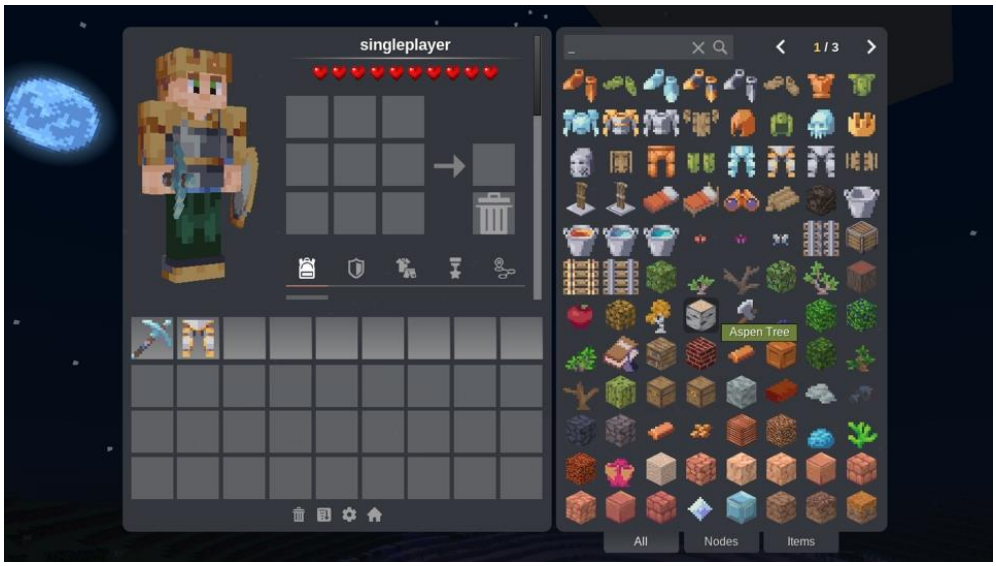
Η Unity πέρα της C++ χρησιμοποιεί κυρίως τη γλώσσα C# για την υλοποίηση σεναρίων (scripting), καθιστώντας τη μία κοινή γλώσσα για την ανάπτυξη indie και mobile παιχνιδιών. Η Java [17] χρησιμοποιείται επίσης ως γλώσσα ανάπτυξης παιχνιδιών, και ιδιαίτερα για παιχνίδια που εκτελούνται στο λειτουργικό σύστημα Android [18], και υπήρξε καθοριστική για την ανάπτυξη παιχνιδιών σε κινητά κατά την εποχή του Java ME.

Η Python [19] χρησιμοποιείται συχνά για τη δημιουργία σεναρίων και την πρωτοτυποποίηση λόγω της απλότητας και της αναγνωσιμότητάς της. Η JavaScript έγινε επίσης πολύ σημαντική για τα διαδικτυακά παιχνίδια (web-based games) που βρίσκονταν ενσωματωμένα σε ιστοσελίδες. Τα παιχνίδια αυτά αποτελούσαν τα γνωστά παιχνίδια περιηγητή (browser games), και χρησιμοποιούσαν πακέτα όπως το Phaser [20] και το Babylon.js [21] ώστε να διευκολύνουν την ανάπτυξή τους. Στο παρόν οι αναδυόμενες γλώσσες προγραμματισμού όπως η Rust [22] και η Go [23] κερδίζουν έδαφος λόγω των χαρακτηριστικών απόδοσης και ασφάλειας, με την Rust να υιοθετείται από ορισμένες σύγχρονες μηχανές παιχνιδιών.

Πέρα τον γνωστών γλωσσών προγραμματισμού γενικού σκοπού υπάρχουν ειδικές γλώσσες όπως η HLSL και η GLSL που χρησιμοποιούνται για τη συγγραφή των shaders, ενώ η Lua [24] χρησιμοποιείται για τη συγγραφή σεναρίων μέσα σε μηχανές ανάπτυξης παιχνιδιών λόγω της ταχύτητας και της χαμηλού υπολογιστικού φόρτου. Η επιλογή γλώσσας προγραμματισμού στην ανάπτυξη παιχνιδιών εξαρτάται από τις συγκεκριμένες απαιτήσεις του παιχνιδιού, την πλατφόρμα και την εξοικείωση της ομάδας ανάπτυξης με αυτή.

1.3 Ανοικτός Κώδικας

Τα προγράμματα ανοικτού κώδικα είναι προγράμματα όπου ο πηγαίος κώδικας είναι διαθέσιμος στο ευρύ κοινό για μετατροπή, ανάγνωση και βελτίωση. Σε αντίθεση με τα παραδοσιακά προγράμματα όπου ο πηγαίος κώδικας δεν είναι ορατός και προσιτός στον απλό χρήστη, τα προγράμματα ανοικτού κώδικα προσφέρουν στον προγραμματιστή την ευχέρεια να τα τροποποιήσει όπως εκείνος θέλει, να προσθέσει λειτουργίες αλλά και να τα αλλάξει εντελώς αν εκείνος το επιθυμεί. Ένα παράδειγμα προγράμματος ανοικτού κώδικα είναι το LibreOffice [25], ένα εργαλείο παραγωγικότητας (productivity tool) που ξεκίνησε ως ένα fork του Openoffice.org [26] το 2010. Στον κόσμο των βιντεοπαιχνιδιών τα παιχνίδια ανοικτού κώδικα ακολουθούν ίδια ιδεολογία, δηλαδή την ανεξαρτητοποίηση από την παγκόσμια αγορά και τους mainstream δημιουργούς και εταιρίες και δίνουν την δυνατότητα σε προγραμματιστές παγκοσμίως να συμμετέχουν σε ένα κοινό έργο. Ένα παράδειγμα παιχνιδιού ανοικτού κώδικα είναι το Minetest [27], το sandbox παιχνίδι εμπνευσμένο από το Minecraft [28] το οποίο δημιουργήθηκε το 2010 από τον Perttu Ahola γνωστός ως Celeron55, ο οποίος ήθελε να φτιάξει και να προσφέρει ένα πιο ελαφρύ και προσβάσιμο Minecraft και με την βοήθεια προγραμματιστών και χρηστών από όλο τον κόσμο το Minetest συνεχίζει να αναπτύσσεται μέχρι και σήμερα.



Εικόνα 8: Οθόνη εξατομίκευσης παίκτη στο Minetest με χρήση του i3 mod. Πηγή: [27]

Εικόνα 9: Steampunk Blimp mod του Minetest επιτρέπει στους παίκτες να χρησιμοποιήσουν αερόστατα για μετακίνηση μέσα στο παιχνίδι. Πηγή: [27]

1.3 Διεπιστημονική συμβολή

Πέρα από την χρήση τους ως ένα μέσο ψυχαγωγίας, τα βιντεοπαιχνίδια μπορούν να χρησιμοποιηθούν και ως μέσα περίθαλψης, διάγνωσης και εκπαίδευσης. Λόγω των ισχυρών αισθητήριων ερεθισμάτων που δημιουργούν οι γρήγορες εναλλαγές γραφικών και η διαδραστικότητα που προσφέρεται, τα βιντεοπαιχνίδια προτείνονται ως λύση για περιπτώσεις ηλικιωμένων και παιδιών, στους οποίους απαιτείται η συγκέντρωση και προσοχή. Στην παρούσα υποενότητα θα συζητηθούν διάφορες χρήσεις των βιντεοπαιχνιδιών σε άλλους τομείς, με βάση τη βιβλιογραφία.

1.3.α Υγειονομικός τομέας

Στην ερευνητική τους εργασία οι Dell'Osso et al. [29] μελέτησαν τις επιπτώσεις του gaming στις γνωστικές λειτουργίες ηλικιωμένων. Οι συγγραφείς τονίζουν πως η ήπια γνωστική εξασθένηση επηρεάζει ένα σημαντικό τμήμα του ηλικιωμένου πληθυσμού και συχνά εξελίσσεται σε άνοια μέσα σε λίγα χρόνια. Σε αυτό το στάδιο, τα άτομα μπορεί να ωφεληθούν από μη φαρμακευτικές θεραπείες που μπορούν να καθυστερήσουν ή να σταματήσουν την εξέλιξη της ήπιας γνωστικής εξασθένησης σε άνοια και είναι ζωτικής σημασίας για τη βελτίωση της ποιότητας ζωής του υποκειμένου, ενώ παράλληλα είναι εύκολα προσβάσιμα και ασφαλή για χρήση. Έρευνες έχουν δείξει ότι μια ποικιλία ασκήσεων, συμπεριλαμβανομένης της γνωστικής εκπαίδευσης, έχουν τη δυνατότητα να ενισχύσουν ή να βελτιστοποιήσουν τη γνωστική λειτουργία και τη γενική ευημερία. Ως εκ τούτου, η

προτάσεις για την χρήση βιντεοπαιχνιδιών ως ασκήσεις για την γνωστικές λειτουργίες πληθαίνουν. Τα παιχνίδια παρέχουν στους ηλικιωμένους κίνητρα ενώ τα ψηφιακά περιβάλλοντα διατηρούν την συγκέντρωση μέσω των οπτικών τους ερεθισμάτων.

Από την συγκέντρωση της διαθέσιμης βιβλιογραφίας οι Dell'Osso et al. συμπέραναν πως ηλικιωμένοι που εκπαιδεύτηκαν με τη χρήση βιντεοπαιχνιδιών εμφάνισαν βελτίωση στις γνωστικές τους ικανότητες, μείωση καταθλιπτικών συμπτωμάτων, βελτιωμένη ποιότητα ύπνου και μείωση στο άγχος. Επιπλέον, παρατηρήθηκε βελτίωση στην μνήμη των συμμετεχόντων στις έρευνες, καθώς και αύξηση της προσοχής τους και συνολικά ευνοϊκές επιδράσεις στις καθημερινές τους δραστηριότητες.

Σε ανάλογη εργασία, οι Wu et al. [30] μελέτησαν τα πιθανά οφέλη της ενασχόλησης με βιντεοπαιχνίδια στη γνωστική λειτουργία και ο πιθανός ρόλος του στη μείωση του κινδύνου της νόσου του Αλτσχάιμερ (AD – Alzheimer's Disease). Ωστόσο, οι μελέτες σε αυτό το θέμα έχουν αξιοσημείωτους περιορισμούς. Για να αντιμετωπιστεί αυτό, οι συγγραφείς εφάρμοσαν Μενδελιανή τυχαιοποίηση (MR – Mendelian Randomization) και αξιοποίησαν δεδομένα παρμένα μέσω της μελέτης GWAS (Genome-Wide Association Study) για να διερευνήσουμε τη συσχέτιση μεταξύ της ενασχόλησης με παιχνίδια, της γνωστικής λειτουργίας, του κινδύνου AD και των επιπέδων του νευροτροφικού παράγοντα (BDNF – Brain-derived Neurotrophic Factor) που προέρχεται από τον εγκέφαλο. Οι συγγραφείς συνέλεξαν σύνολα δεδομένων για την ενασχόληση με τα βιντεοπαιχνίδια, τη γνωστική λειτουργία, τον κίνδυνο της AD και τα επίπεδα του νευροτροφικού παράγοντα που προέρχεται από τον εγκέφαλο (BDNF) από το έργο IEU Open GWAS [31]. Για να αναλύσουν τις αιτιώδεις σχέσεις μεταξύ αυτών των μεταβλητών, χρησιμοποίησαν μια ποικιλία μεθόδων MR. Για την εξασφάλιση της ακρίβειας των αποτελεσμάτων χρησιμοποιήθηκαν αναλύσεις ευαισθησίας.

Τα αποτελέσματα της μελέτης αποκάλυψαν σημαντικές συσχετίσεις μεταξύ της ενασχόλησης με τα βιντεοπαιχνίδια και της γνωστικής λειτουργίας, υποδηλώνοντας ότι η ενασχόληση με τέτοιες δραστηριότητες μπορεί να έχει ευεργετικό αντίκτυπο στις γνωστικές ικανότητες. Η ανάλυση έδειξε ότι η αυξημένη συμμετοχή στα παιχνίδια υπολογιστών συνδέθηκε με μειωμένο κίνδυνο για τη νόσο του Αλτσχάιμερ (AD). Επιπλέον, η μελέτη διαπίστωσε ότι τα υψηλότερα επίπεδα του νευροτροφικού παράγοντα που προέρχεται από τον εγκέφαλο (BDNF) συσχετίστηκαν τόσο με το παιχνίδι στον υπολογιστή όσο και με τη βελτίωση της γνωστικής λειτουργίας.

Οι αναλύσεις MR υποστήριξαν περαιτέρω την ιδέα ότι οι παρατηρούμενες σχέσεις ήταν πιθανώς αιτιώδεις παρά συγχέονται από άλλους παράγοντες. Οι αναλύσεις ευαισθησίας επιβεβαίωσαν την αξιοπιστία αυτών των ευρημάτων, υποδεικνύοντας ότι οι εκτιμήσεις ήταν σταθερές και δεν επηρεάστηκαν σημαντικά από παραβιάσεις των υποθέσεων της MR. Συνολικά, τα αποτελέσματα παρέχουν πειστικές αποδείξεις για τα πιθανά γνωστικά οφέλη από το παιχνίδι στον υπολογιστή και τον ρόλο του στον μετριασμό του κινδύνου της ΝΑ, υπογραμμίζοντας τη σημασία της περαιτέρω έρευνας σε αυτόν τον τομέα.

1.3.β Εκπαίδευση

Λόγω της εξοικείωσης των παιδιών του σήμερα με τα βιντεοπαιχνίδια έχουν προταθεί κατά καιρούς μεθοδολογίες διδασκαλίας και διάγνωσης/αξιολόγησης μαθησιακών δυσκολιών με τη χρήση βιντεοπαιχνιδιών. Μέσω των παιχνιδιών αυτών, τα παιδιά εκφράζονται και εισέρχονται σε μια διαφορετική νοητική κατάσταση. Η αύξηση της συγκέντρωσης, η ελκυστικότητα του παιχνιδιού και η άμεση επιβράβευση είναι μερικά από τα χαρακτηριστικά που καθιστούν τα βιντεοπαιχνίδια από ένα μέσο διασκέδασης σε οριακά «εθιστικές ουσίες». Το μέσο έδαφος είναι η χρήση μιας πλατφόρμας παιχνιδιού η οποία να κεντρίζει το ενδιαφέρον των παιδιών και να μεταδίδει την απαραίτητη πληροφορία στους/στις μαθητές/τρίες.

Στην ερευνητική της εργασία, η Shokirovna [32] ασχολήθηκε με την μελέτη της σχέσης μεταξύ εκπαίδευσης και βιντεοπαιχνιδιών. Συγκεκριμένα στο άρθρο της εξέτασε τον ρόλο των παιχνιδιών στην εκπαίδευση. Τα βιντεοπαιχνίδια με βάση τη συγγραφέα παρέχουν ψυχαγωγία, χαλάρωση και ψυχική τόνωση και μπορεί επίσης να αποτελέσουν πολύτιμή πηγή εκπαίδευσης. Η σημασία των παιχνιδιών στην εκπαίδευση έχει αποκτήσει αυξημένη προσοχή τα τελευταία χρόνια, με πολλούς εκπαιδευτικούς να χρησιμοποιούν παιχνίδια για να ερμηνεύσουν και να παρακινήσουν τους/τις μαθητές/τριες σε διάφορα μαθήματα. Η συγγραφέας παραθέτει την ανάγκη για κριτική σκέψη, συνεργασία και αφοσίωση ως σημαντικά χαρακτηριστικά που εμφανίζονται στα άτομα που ασχολούνται σε υγιή βαθμό με το gaming. Η επιμονή και η ανάπτυξη δεξιοτήτων, όπως ο συγχρονισμός ματιού-χεριού (hand-eye coordination) αποτελούν θετικά αποτελέσματα της ενασχόλησης με τα βιντεοπαιχνίδια. Αυτό που τονίζεται στην έρευνα είναι πως τα βιντεοπαιχνίδια δεν πρέπει

για κανένα λόγο να αντικαθιστούν τις παραδοσιακές μεθόδους διδασκαλίας εξ' ολοκλήρου, αλλά θα πρέπει να λαμβάνουν υποστηρικτικό ρόλο.

Τα παιχνίδια διαδραματίζουν σημαντικό ρόλο και κατά την διάγνωση μαθησιακών δυσκολιών. Στις ερευνητικές τους εργασίες οι Zygouris et al. [33] [34], υλοποίησαν διαδικτυακά παιχνίδια με την χρήση τεχνολογιών διαδικτύου για την αξιολόγηση δυσαριθμησίας και δυσλεξίας αντίστοιχα. Κάθε παιχνίδι αποτελούταν από προκαθορισμένο αριθμό ειδικά σχεδιασμένων δοκιμασιών, που τα παιδιά έπρεπε να φέρουν εις πέρας. Κατά την πειραματική αξιολόγηση, οι συγγραφείς συνέλεξαν στατιστικά δεδομένα και αξιολόγησαν την επίδοση των παιδιών που ήταν διαγνωσμένα με κάποια μαθησιακή δυσκολία, έναντι παιδιών δίχως κάποια μαθησιακή δυσκολία. Στόχος ήταν η μελέτη των συμπτωμάτων που εμφανίζονται και η σημαντικότητα τους κατά την αξιολόγηση μιας μαθησιακής δυσκολίας. Και στις δύο περιπτώσεις παρατηρήθηκαν στατιστικά σημαντικές διαφορές μεταξύ των δύο ομάδων.

ΚΕΦΑΛΑΙΟ 2 Ανάπτυξη Παιχνιδιών με τη Γλώσσα Python

2.1 Βασικά χαρακτηριστικά

Η γλώσσα προγραμματισμού Python [19] χαρακτηρίζεται από την εύκολη εκμάθηση, παρέχοντας παράλληλα υψηλές δυνατότητες. Η γλώσσα είναι σχεδιασμένη έτσι ώστε να δίνει έμφαση στην αναγνωσιμότητα, ενώ έχει την δυνατότητα να κλιμακώνεται ανάλογα με την ικανότητα του προγραμματιστή. Η python στα χέρια ενός αρχάριου προγραμματιστή αποτελεί απλά ένα εργαλείο συγγραφής σεναρίων, ενώ ένας έμπειρος προγραμματιστής μπορεί να την χρησιμοποιήσει για πληθώρα εφαρμογών όπως τεχνητή νοημοσύνη, ανάπτυξη διαδικτυακών εφαρμογών, προγραμματισμό ρομποτικών συστημάτων, προσομοιώσεις και υλοποίηση παιχνιδιών. Η καταλληλότερη χρήση της γλώσσας Python είναι η ταχεία προτυποποίηση, που επιτρέπει την γρήγορη κατασκευή σεναρίων προσομοίωσης και δοκιμών. Πέρα από αυτά, η Python μπορεί να χρησιμοποιηθεί για την ανάπτυξη πλήρως ανεπτυγμένων εφαρμογών.

Η Python είναι σχεδιασμένη για να μεταφέρει κατευθείαν την προγραμματιστική λογική (αλγόριθμος) σε αναπαράσταση κωδικοποίησης (κώδικας). Αν η υλοποίηση είναι δύσκολο να εξηγηθεί, τότε είναι κακός ο σχεδιασμός του αλγόριθμου. Από την άλλη, αν είναι εύκολο να εξηγηθεί, τότε μάλλον ο τρόπος είναι σωστός. Από την πλευρά του χρόνο για την ανάπτυξη ενός λογισμικού έχουμε δύο βασικούς παράγοντες:

- Το χρόνο για τον προγραμματισμό του, το οποίο περιγράφεται σε ανθρωποώρες.
- Το χρόνο που το πρόγραμμα κάνει να εκτελεστεί, το οποίο μετριέται σε χρόνο μηχανής.

Κάθε γλώσσα έχει σχεδιαστεί με μια αντιστάθμιση αυτών των δύο παραγόντων. Η Python απαιτεί λιγότερες ανθρωποώρες και εμφανίζει καλούς χρόνους εκτέλεσης για τις περισσότερες εφαρμογές. Η γλώσσα δεν ενδείκνυται για υλοποίηση εφαρμογών όπως λειτουργικά συστήματα και μεταγλωττιστές, τα οποία απαιτούν άμεση επαφή με το υλικό του υπολογιστή και πολύ υψηλές ταχύτητες επεξεργασίας. Η Python εντάσσεται στις γλώσσες υψηλού επιπέδου, καθώς περιλαμβάνει ενσωματωμένες δομές δεδομένων υψηλού επιπέδου και είναι αντικειμενοστραφής. Ως γλώσσα είναι διερμηνευόμενη, το οποίο σημαίνει πως:

- Τα προγράμματα δεν μεταφράζονται σε γλώσσα μηχανής πριν την εκτέλεση τους.

- Πλαίσιο εξαιρέσεων με backtracing λαθών, ενώ δεν υπάρχουν segmentation faults.
- Αυτόματη διαχείριση μνήμης.
- Δυναμική ανάθεση τύπων μεταβλητών και δυναμική δεικτοδότηση μεταβλητών στη μνήμη.

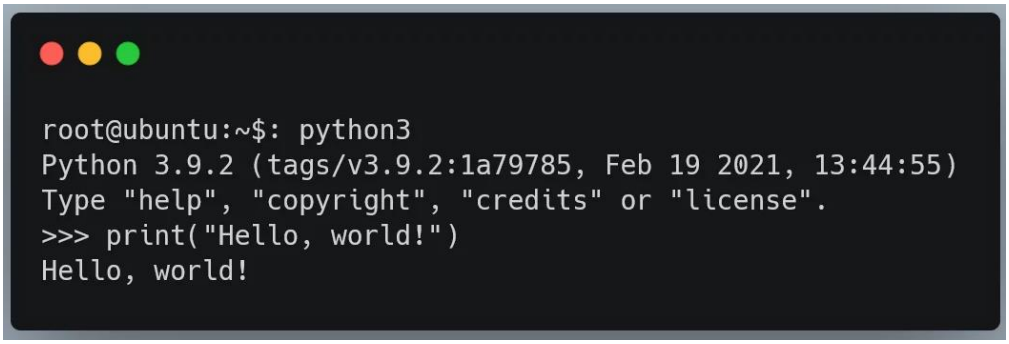
Στα περαιτέρω πλεονεκτήματα της Python συγκαταλέγονται η τεράστια ενσωματωμένη βιβλιοθήκη με όλων των ειδών λειτουργικότητα και το σημαντικότερο όλων είναι πως η Python εκτελείται σε όλα τα λειτουργικά συστήματα, είναι δωρεάν και ανοικτού κώδικα. Τέλος, η Python είναι μοναδική και ως προς το ότι διαθέτει:

- Επεκτασιμότητα, τόσο εγγενή, όσο και δυνατότητα συνδυασμού με άλλες γλώσσες.
- Ένα πολύ ενεργό οικοσύστημα ανάπτυξης λογισμικού από τρίτους.
- Ευρεία αποδοχή για την ανάπτυξη εφαρμογών διαδικτύου.
- Πολλές βιβλιοθήκες για επιστημονικό προγραμματισμό.
- Πολλές επιλογές για μεγαλύτερη αποδοτικότητα.

2.2 Προγραμματισμός με Python

2.2.α Διερμηνευτής της Python

Ο προγραμματισμός με την γλώσσα Python μπορεί να γίνει διάφορους τρόπους. Για αρχή έχουμε προγραμματισμό μέσω του διερμηνευτή της Python (βλ. Εικόνα 10).



Εικόνα 10: Στιγμιότυπο χρήσης διερμηνευτή της Python.

Ο διερμηνευτής της Python αποτελεί πρόγραμμα παρόμοιο με τα προγράμματα κελύφους των Linux. Επιτρέπει την εκτέλεση κώδικα Python και εμφανίζει τα αποτελέσματα κατευθείαν στην οθόνη μέσω ενός ατέρμονος βρόχου, ο οποίος δέχεται ως είσοδο της εντολές και παράγει τα κατάλληλα αποτελέσματα. Ως συστατικό της Python, ο διερμηνευτής

κατέχει κεντρικό ρόλο καθώς είναι το πρόγραμμα που εκτελεί έναν κώδικα γραμμή προς γραμμή. Κάθε γραμμή μεταφράζεται σε ενδιάμεσο κώδικα (bytecode), και εκτελείται άμεσα. Παράλληλα με την μετάφραση, ο διερμηνευτής αναθέτει τους τύπους δεδομένων στις μεταβλητές που αναγνωρίζει. Ανάλογα με την τιμή που ανατίθεται στην μεταβλητή, επιλέγεται και ο κατάλληλος τύπος δεδομένων. Πρακτικά, ο διερμηνευτής λειτουργεί σαν ένα επίπεδο αφαίρεσης, δηλαδή σαν ένα μικροσκοπικό λειτουργικό σύστημα πάνω από το πραγματικό λειτουργικό σύστημα.

Η κύρια χρήση του διερμηνευτή είναι διαδραστικά, δηλαδή ο προγραμματιστής να γράφει και να εκτελεί τις εντολές σε πραγματικό χρόνο. Η χρήση αυτή είναι αποτελεσματική όταν εκτελούμε σύντομες προτυποποιήσεις ή αριθμητικούς υπολογισμούς. Σε καμία περίπτωση όμως η χρήση της γλώσσας δεν γίνεται ως μια απλή αριθμομηχανή. Ως εκ τούτου, ο διερμηνευτής μπορεί να εκτελέσει τα λεγόμενα σενάρια Python (Python scripts). Τα σενάρια Python είναι αρχεία με την κατάληξη .py και περιλαμβάνουν εντολές που συγκροτούν ένα πρόγραμμα. Σε αντίθεση με τον κλασικό προγραμματισμό, όπου απαιτείται η ύπαρξη μιας συνάρτησης main, τα σενάρια Python δεν απαιτούν την σύνταξη αυτή.

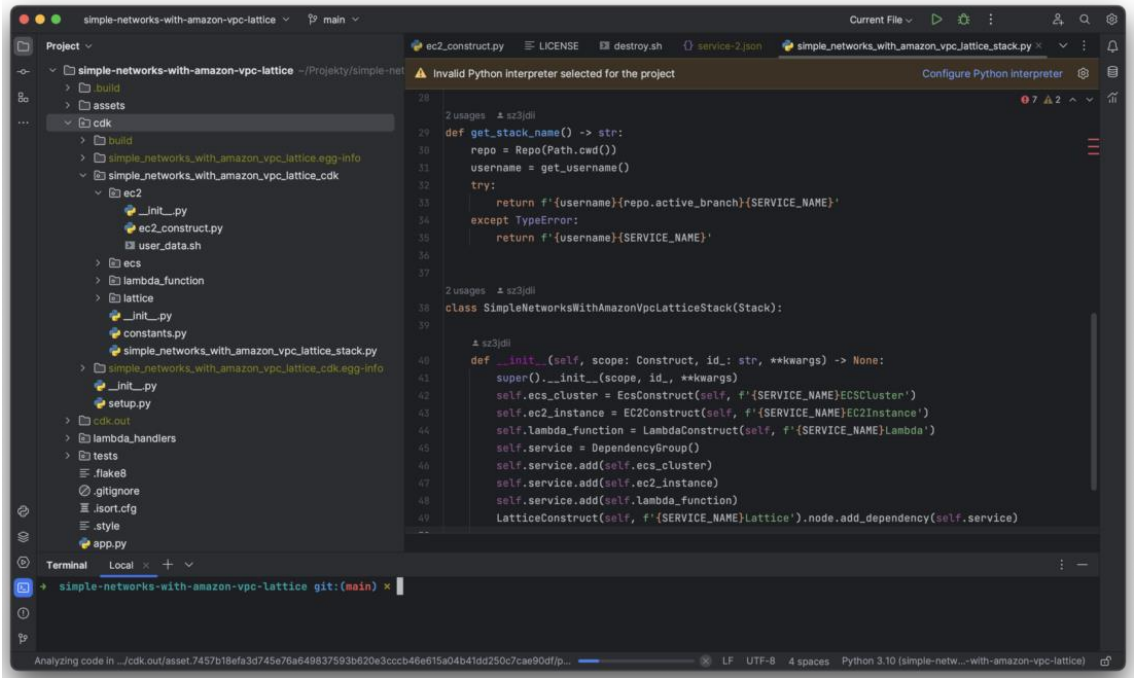
Ο διερμηνευτής Python ακολουθείται από μια προκαθορισμένη βιβλιοθήκη που περιλαμβάνει πληθώρα μεθόδων και αντικειμένων. Οι βασικές λειτουργίες παρέχονται με την μορφή ενοτήτων (modules) και αφορούν λειτουργίες όπως είσοδος/έξοδος, διαχείριση αρχείων και κλάσεις δομών δεδομένων όπως λίστες και λεξικά, που με την σειρά τους παρέχουν μεθόδους για διαχείριση των αντικειμένων που δημιουργούνται.

Η δυνατότητα εκτέλεσης της γλώσσας Python με μόνη απαίτηση την ύπαρξη του διερμηνευτή της επιτρέπει την φορητότητα της γλώσσας. Ένας κώδικας γραμμένος σε γλώσσα Python μπορεί να εκτελεστεί σε οποιαδήποτε πλατφόρμα για την οποία διατίθεται ο διερμηνευτής.

2.2.β Ολοκληρωμένα Προγραμματιστικά Περιβάλλοντα

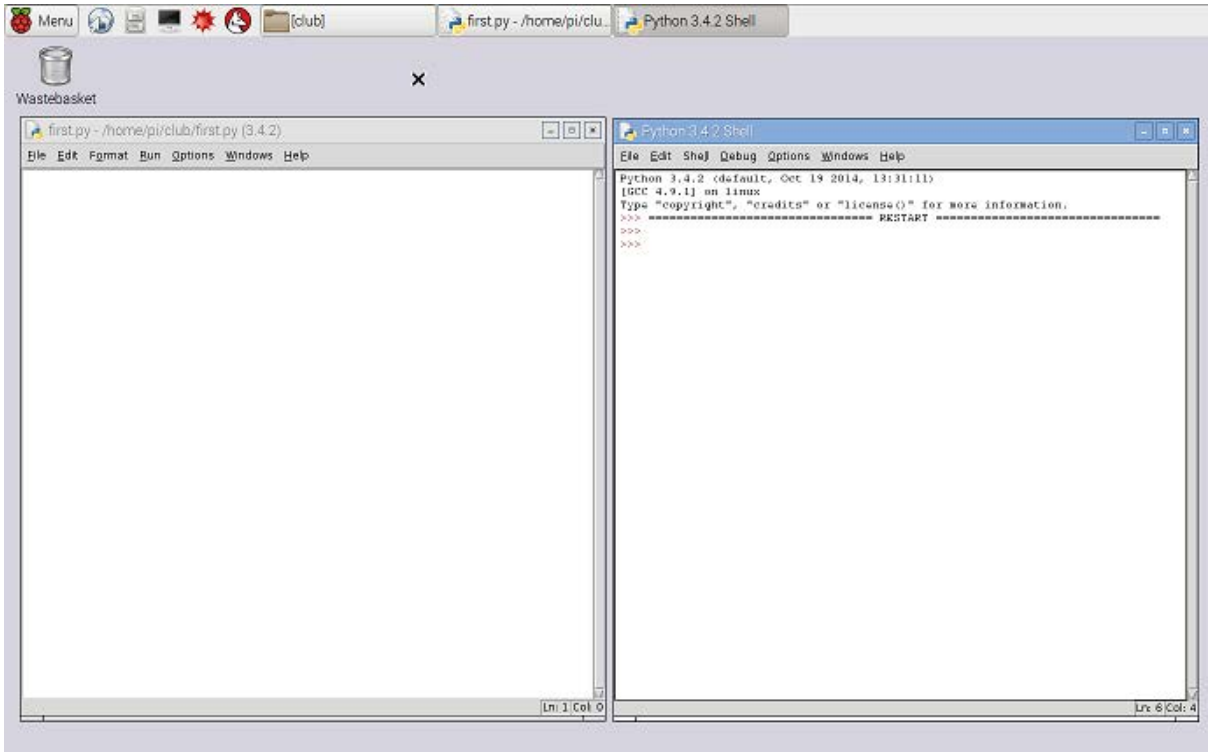
Για την ανάπτυξη πολύπλοκων λογισμικών συχνά δεν προτιμάται η χρήση ενός απλού κειμενογράφου, όπως το Visual Studio Code [35] ή το Notepad++ [36]. Αντιθέτως, υπάρχουν ολοκληρωμένα προγραμματιστικά περιβάλλοντα ή ολοκληρωμένα περιβάλλοντα ανάπτυξης (IDE – Integrated Development Environments) τα οποία επιτρέπουν την δημιουργία και διαχείριση μεγάλων project λογισμικού, τα οποία περιέχουν πολλαπλά αρχεία, ενότητες και βιβλιοθήκες. Τα περιβάλλοντα αυτά παρέχουν, πέρα των εργαλείων debug και εκτέλεσης

κώδικα, διασυνδέσεις με λογισμικά όπως το Git για έλεγχο των εκδόσεων του project. Ένα από τα πιο γνωστά και πρακτικά IDE, ειδικά υλοποιημένο για Python, αποτελεί το JetBrains PyCharm [37]. Ως λογισμικό το PyCharm παρέχει ανάλυση και βοήθεια κατά τη συγγραφή κώδικα, αυτόματη συμπλήρωση, έλεγχο σύνταξης κατά τη συγγραφή, επισήμανση λαθών και παραλήψεων όπως και διαχείριση project και κώδικα μέσω εξειδικευμένων views. Επιπλέον παρέχει υποστήριξη για Docker, ενώ επιτρέπει την σύνδεση με πλατφόρμες Git όπως είναι το GitHub ή το GitLab.



Εικόνα 11: Στιγμιότυπο χρήσης IDE JetBrains PyCharm. Πηγή: https://cloudybarz.com/fixing-highlights-for-aws-cdk-in-jetbrains-pycharm/

Από την εγκατάστασή της, η Python παρέχει τον χρήστη με το IDLE (Integrated Development and Learning Environment). Στόχος του λογισμικού είναι να αποτελεί ένα απλοποιημένο IDE το οποίο είναι κατάλληλο για αρχάριους. Χρησιμοποιείται ιδιαίτερα σε εκπαιδευτικά περιβάλλοντα καθώς απαιτεί μηδενικό χρόνο εγκατάστασης και προετοιμασίας. Το IDLE χρησιμοποιεί δύο παράθυρα. Το βασικό παράθυρο περιέχει τον διερμηνευτή της Python, το οποίο ονομάζεται Shell. Το δεύτερο παράθυρο αποτελεί τον συντάκτη σεναρίων, όπου επιτρέπεται η συγγραφή σεναρίων για εκτέλεσή τους στο Shell. Το IDLE εμφανίζει την μεγαλύτερη συμβατότητα με τους διερμηνευτές της Python καθώς υλοποιείται ώστε να τους συνοδεύει ενώ έχει και πολύ μικρές απαιτήσεις μνήμης.



Εικόνα 12: Στιγμιότυπο χρήσης Python IDLE. Παράθυρο κώδικα (αριστερά), παράθυρο Shell (δεξιά). Πηγή: https://python-with-science.readthedocs.io/en/latest/program/program.html

2.3 Στοιχεία της Python

Όπως κάθε γλώσσα προγραμματισμού έτσι και η Python αποτελείται από κάποια συγκεκριμένα στοιχεία. Τα στοιχεία αυτά είναι:

- Λεξιλόγιο: μεταβλητές και δεσμευμένες λέξεις.
- Δομή πρότασης: έγκυρα πρότυπα σύνταξης (συντακτικοί κανόνες).

Οι προτάσεις αποτελούν εκφράσεις που συντίθενται από μεταβλητές, τελεστές, σταθερές ή δεσμευμένες λέξεις. Παραδείγματα εντολών αποτελούν οι παρακάτω εντολές:

- X = 2
- X = X + 2
- Print(x)

Η πρώτη πρόταση περιγράφει μια εντολή εκχώρησης, η δεύτερη μια εντολή εκχώρησης με έκφραση ενώ η τρίτη μια εντολή εκτύπωσης στην οθόνη.

Ως κομμάτι του λεξιλογίου οι σταθερές αποτελούν θέσεις μνήμης που η τιμή τους δεν αλλάζει κατά την εκτέλεση του κώδικα. Οι σταθερές αριθμητικών τύπων όπως integer ή double ορίζονται όπως τις γνωρίζουμε από άλλα μαθήματα. Η διαφορά είναι πως αν

θέλουμε να δηλώσουμε μια αλφαριθμητική σταθερά χρησιμοποιούμε μονά εισαγωγικά (‘) αντί για διπλά εισαγωγικά (“).

Η χρήση των μεταβλητών είναι παρόμοια με όλες τις άλλες γλώσσες προγραμματισμού, μόνο που στην Python δεν απαιτείται η δήλωση του τύπου δεδομένων των μεταβλητών πριν την χρήση τους. Πρακτικά, στην Python η δήλωση και η εκχώρηση τιμής σε μια μεταβλητή γίνεται με μια εντολή εκχώρησης. Οι περιορισμοί ονοματοδοσίας των μεταβλητών είναι παρόμοιοι με όλες τις γλώσσες προγραμματισμού. Κάθε όνομα μεταβλητής πρέπει να ξεκινά με ένα γράμμα ή με κάτω παύλα (_). Επιπλέον θα πρέπει να αποτελούνται μόνο από γράμματα, αριθμούς και κάτω παύλες. Σημαντικό είναι να αναφερθεί πως η Python δεν αδιαφορεί για τα κεφαλαία και τα μικρά, συνεπώς μεταβλητές που έχουν το ίδιο όνομα αλλά η μια είναι με μικρά και η άλλη είναι με κεφαλαία γράμματα, τότε αφορούν δύο διαφορετικές διευθύνσεις μνήμης. Πέρα από τα ονόματα μεταβλητών, ορίζεται και ένα σύνολο δεσμευμένων λέξεων, οι οποίες απαγορεύεται να χρησιμοποιούνται από τους προγραμματιστές. Τέτοιες λέξεις είναι οι: def, del, for, is, raise, assert, elif, from, lambda, return, break, else, global, not, try, class, except, if, or, while, continue, exec, import, pass, yield, finally, in, print, as with.

Το σημαντικότερο χαρακτηριστικό της Python είναι πως δεν διαθέτει κάποιον ειδικό χαρακτήρα που να ορίζει κάποια δομή, όπου αλλάζει η εμβέλεια, όπως π.χ. κάποια δομή επιλογής ή επανάληψης. Αντί κάποιου ειδικού χαρακτήρα, η Python χρησιμοποιεί εσοχές (tabs) για να δηλώσει πως μια συγκεκριμένη μεταβλητή ανήκει σε μια συγκεκριμένη εμβέλεια.

2.4 Modules (Ενότητες)

Για να χρησιμοποιηθεί ένα module πρέπει πρώτα να εκτελεστεί η πρόταση import module. Το import module καλεί την Python να εκτελέσει κάθε εντολή του module. Συνήθως μέσα σε ένα module υπάρχουν δηλώσεις συναρτήσεων που εκτελούνται με πολύ μεγάλη ταχύτητα. Για την κλήση μιας συνάρτηση ή την χρήση μιας μεταβλητής που υπάρχει μέσα σε ένα module χρησιμοποιείται ο τελεστής τελεία για πρόσβαση στα δεδομένα του module. Για παράδειγμα η εντολή module_name.function_name() σημαίνει πως καλείται η συνάρτηση function_name(), που είναι ορισμένη μέσα στο module module_name.

Με τα modules γίνεται εφικτή η δημιουργία βιβλιοθηκών κώδικα και η ανάπτυξη εργαλείων για καινοτόμες εφαρμογές. Τα modules είναι αναπόσπαστο κομμάτι του προγραμματισμού με Python, καθώς πάντα υπάρχει κάποιο module για μια διαδικασία ή υπολογισμό που απαιτείται για την επίλυση ενός προβλήματος. Ακόμα και αν δεν λύνει ακριβώς το πρόβλημα, τα modules είναι ανοικτού κώδικα, πράγμα που σημαίνει πως είναι εφικτή η τροποποίηση από τον προγραμματιστή. Μερικά από τα σημαντικότερα modules για έναν προγραμματιστή είναι τα: copy, csv, json, pandas, math, os, random, re, threading κ.α. Το module copy επιτρέπει την δημιουργία αντιγράφων με ασφαλή τρόπο, ενώ τα modules csv και json παρέχουν parsers για την διαχείριση αρχείων του αντίστοιχου τύπου. Το module pandas χρησιμοποιείται πολύ στην ανάλυση δεδομένων καθώς επιτρέπει την δημιουργία πακέτων δεδομένων (Dataframes) για την αποθήκευση, επεξεργασία και μετασχηματισμό των δεδομένων. Τα modules math, os, random και re παρέχουν μαθηματικές πράξεις, υποστήριξη λειτουργικού συστήματος (διαχείριση αρχείων), παραγωγή τυχαίων αριθμό και διαχείριση κανονικών εκφράσεων αντίστοιχα.

2.5 Αντικειμενοστρέφεια

Ως γλώσσα υψηλού επιπέδου η Python παρέχει την δυνατότητα κατασκευής αντικειμένων μέσω δημιουργίας κλάσεων. Μια κλάση αποτελεί ένα «καλούπι» για την δημιουργία ενός αντικειμένου, το οποίο καλείται και στιγμιότυπο της κλάσης. Κάθε κλάση περιγράφει ένα αντικείμενο μέσω των πεδίων του, τα οποία αποτελούν στοιχεία δεδομένων, για παράδειγμα το όνομα ενός ανθρώπου. Επιπλέον των πεδίων μιας κλάσης υπάρχουν οι μέθοδοι που περιγράφουν τις ενέργειες που μπορεί να εκτελέσει ένα αντικείμενο της κλάσης. Για παράδειγμα ένας σκύλος έχει την μέθοδο «γάβγισμα», όπου παράγει ένταση ήχου ανάλογα με την ράτσα του.

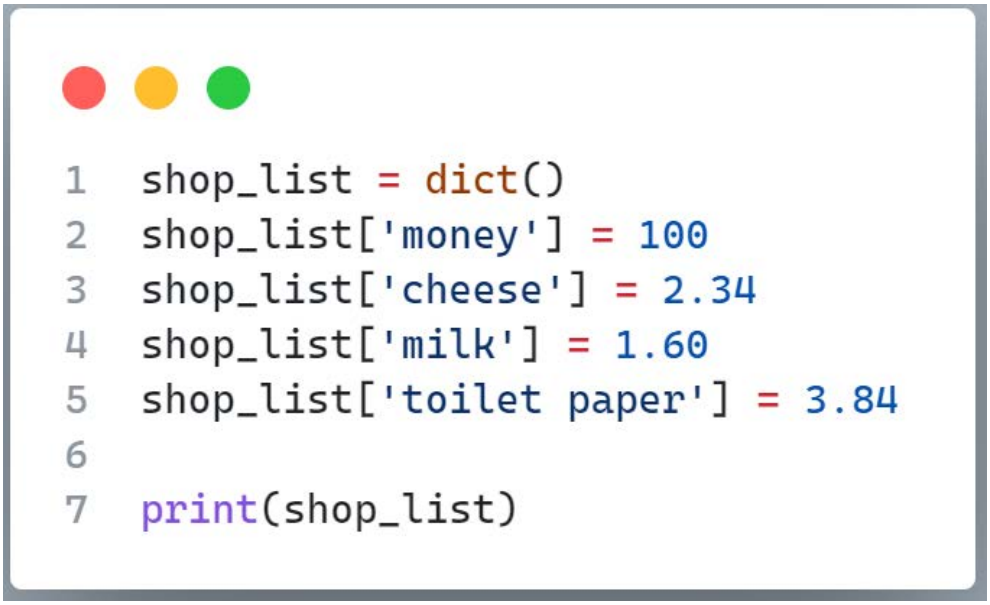
Μια κλάση στην Python δημιουργείται χρησιμοποιώντας την δεσμευμένη λέξη class και δίπλα το όνομα της κλάσης, π.χ. **class Human**. Ένα αντικείμενο δημιουργείται όταν κληθεί ο constructor του αντικειμένου, π.χ. **h1 = Human(“Manos”, 26, “Student”)**. Στο παράδειγμα η κλάση **Human** περιλαμβάνει ως πεδία το όνομα, την ηλικία και την ιδιότητα ενός ανθρώπου. Μια μέθοδος για την κλάση **Human** θα μπορούσε να είναι η **study(int hours)**, η οποία αντιστοιχεί το διάβασμα που κάνει ένας άνθρωπος για μια συγκεκριμένη διάρκεια. Η μέθοδος θα μπορούσε να καλεί την συνάρτηση **sleep()** για τον αριθμό ωρών που έδωσε ο

χρήστης, καθιστώντας τον άνθρωπο που περιγράφεται από το αντικείμενο **h1** απασχολημένο για **hours** ώρες.

2.6 Συλλογές

Μια συλλογή επιτρέπει την αποθήκευση πολλαπλών τιμών κάτω από ένα όνομα μεταβλητής. Η πιο απλή μορφή συλλογής είναι η λίστα, η οποία είναι μεταβλητή επιτρέποντας την αλλαγή σε μέλη της λίστας με τη χρήση ενός δείκτη, όμοια με τους πίνακες στον κλασικό προγραμματισμό. Το βασικότερο πλεονέκτημα των λιστών σε σχέση με τους παραδοσιακούς πίνακες της C ή της Java είναι πως οι λίστες είναι δυναμικές. Με χρήση μεθόδων των αντικειμένων τύπου λίστας, όπως η **append** είναι δυνατή η συνεχής προσθήκη δεδομένων μέσα σε μια λίστα. Αν τώρα είναι επιθυμητό να μην αλλάζουν τα περιεχόμενα μιας λίστας, τότε αυτή μπορεί να μετατραπεί σε **πλειάδα (tuple)**. Η μετατροπή αυτή βοηθά αν υπάρχει η πιθανότητα να συμβούν απροσεξίες κατά την ανάθεση μεταβλητών.

Οι λίστες και οι πλειάδες δεν είναι οι μόνες συλλογές που υπάρχουν στην Python. Η πιο ισχυρή συλλογή της Python είναι τα **Λεξικά (Dictionaries)**. Τα λεξικά επιτρέπουν την υλοποίηση γρήγορων ενεργειών, παρόμοιες με αυτές στις βάσεις δεδομένων. Τα λεξικά της Python είναι ανάλογα με τα HashMaps της Java και τα Associative Arrays της PHP. Σε αντίθεση με τις λίστες που στοιχειοθετούν τις εγγραφές βάσει της σειράς τους μέσα σε αυτή, τα λεξικά είναι σαν καλάθι σούπερ μάρκετ, δεν υπάρχει καμία σειρά. Τα στοιχεία μέσα σε ένα λεξικό τοποθετούνται με τη χρήση μιας μοναδικής ετικέτας αναζήτησης. Σημαντικό είναι πως τα λεξικά δεν είναι ταξινομημένα, ενώ τα κλειδιά αναζήτησης είναι αμετάβλητα, όπως στις πλειάδες, αλλά οι τιμές είναι μεταβλητές όπως στις λίστες. Αν έχει δημιουργηθεί ένα λεξικό είναι δυνατή η προσθήκη ενός ζεύγους κλειδιών-τιμών με ανάθεση μιας τιμής σε ένα κλειδί, **dictionary_name[key] = value** (βλ. Εικόνα 13).



```
1  shop_list = dict()
2  shop_list['money'] = 100
3  shop_list['cheese'] = 2.34
4  shop_list['milk'] = 1.60
5  shop_list['toilet paper'] = 3.84
6
7  print(shop_list)
```

Εικόνα 13: Στιγμιότυπο δημιουργίας δομής δεδομένων Λεξικού.

2.7 Pygame

Η βιβλιοθήκη Pygame [38] αποτελεί ένα περιτύλιγμα της βιβλιοθήκης SDL (Simple DirectMedia Layer). Η βιβλιοθήκη SDL είναι φορητή και επιτρέπει την πρόσβαση σε υλικό που ασχολείται με την επεξεργασία πολυμέσων, όπως ο ήχος, το βίντεο, το ποντίκι το πληκτρολόγιο και τα χειριστήρια (joysticks). Η εγκατάσταση της βιβλιοθήκης γίνεται εκτελώντας την εντολή

```
pip install pygame
```

Η βιβλιοθήκη αποτελείται από διάφορα modules τα οποία παρέχουν αφηρημένη πρόσβαση στο υλικό του συστήματος. Για παράδειγμα, το **display** επιτρέπει την πρόσβαση στην οθόνη, ενώ το **joystick** παρέχει μια αφαίρεση για τον έλεγχο ενός χειριστηρίου.

Επιπλέον των modules που παρέχονται, η βιβλιοθήκη περιέχει και αρκετές κλάσεις, οι οποίες ενθυλακώνουν λειτουργίες που δεν είναι άμεσα συνδεδεμένες με το υλικό. Για παράδειγμα η κλάση **Surface** ορίζει μια ορθογώνια περιοχή που μέσα μπορεί να σχεδιαστεί κάτι. Τα αντικείμενα της κλάσης **Surface** χρησιμοποιούνται εκτενώς σε υλοποιήσεις που χρησιμοποιείται το Pygame.

Ο βασικός σχεδιασμός ενός παιχνιδιού ακολουθεί συγκεκριμένα στάδια. Αρχικά πρέπει να κάνουμε import το Pygame και στη συνέχεια να αρχικοποιήσουμε τα απαραίτητα στοιχεία του. Συχνά αυτή η αρχικοποίηση αφορά το mapping των πλήκτρων κίνησης σε συγκεκριμένες σταθερές. Το Pygame ξεκινά να εκτελείται με την κλήση της μεθόδου **init()** ως **Pygame.init()**.

Το αμέσως επόμενο βήμα είναι η κατασκευή της οθόνης. Ορίζεται αρχικά το μέγεθος του παραθύρου παιχνιδιού και στη συνέχεια εκτελείται η μέθοδος **Pygame.display.set_mode()** για την σχεδίαση του παραθύρου.

```
1  import pygame
2
3  from pygame.locals import(
4      K_UP,
5      K_DOWN,
6      K_LEFT,
7      K_RIGHT,
8      K_ESCAPE,
9      KEYDOWN,
10     QUIT,
11 )
12
13 pygame.init()
14
15 SCREEN_WIDTH = 800
16 SCREEN_HEIGHT = 600
17
18 screen = pygame.display.set_mode((SCREEN_WIDTH, SCREEN_HEIGHT))
19
```

Εικόνα 14: Στιγμιότυπο κώδικα αρχικοποίησης και σχεδιασμού αρχικού παραθύρου.

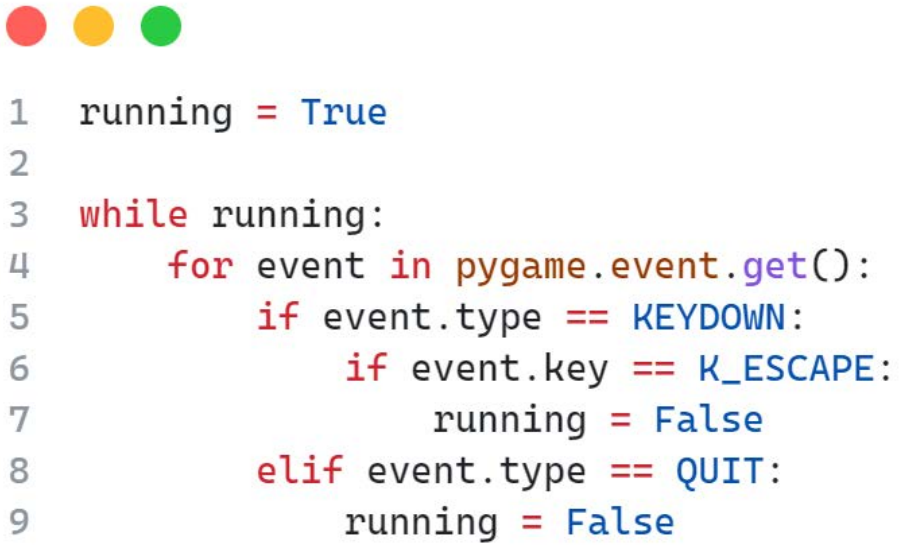
Με την εκτέλεση του παραπάνω κώδικα (Εικόνα 14), δημιουργείται το παράθυρο, αλλά εξαφανίζεται αμέσως. Αυτό συμβαίνει γιατί δεν έχει δημιουργηθεί ακόμα το **λεγόμενο game loop**. Κάθε παιχνίδι, από το Pong μέχρι και το God of War χρησιμοποιεί αυτό που ονομάζεται game loop για τον έλεγχο του **gameplay**. Το game loop κάνει τέσσερα πολύ βασικά πράγματα:

1. Λαμβάνει την είσοδο χρήστη.
2. Ενημερώνει την κατάσταση κάθε αντικειμένου στο παιχνίδι.
3. Ενημερώνει την οθόνη και τους ήχους του παιχνιδιού.
4. Διατηρεί την ταχύτητα του παιχνιδιού.

Κάθε κύκλος του game loop ονομάζεται **frame** (πλαίσιο). Όσο γρηγορότερα γίνονται οι κινήσεις σε ένα frame, τόσο πιο γρήγορα θα τρέχει το παιχνίδι. Το **game loop** εκτελείται μέχρις ότου να επαληθευτεί μια συνθήκη τερματισμού. Οι πιο συχνές συνθήκες τερματισμού είναι η ήττα του παίκτη και η διακοπή του παιχνιδιού με κλείσιμο του παραθύρου.

Η πρώτη εντολή που εκτελεί ένα game loop είναι η λήψη της εισόδου χρήστη, η οποία είσοδος τις περισσότερες φορές είναι μια κίνηση. Αυτό από μόνο του δημιουργεί αρκετά

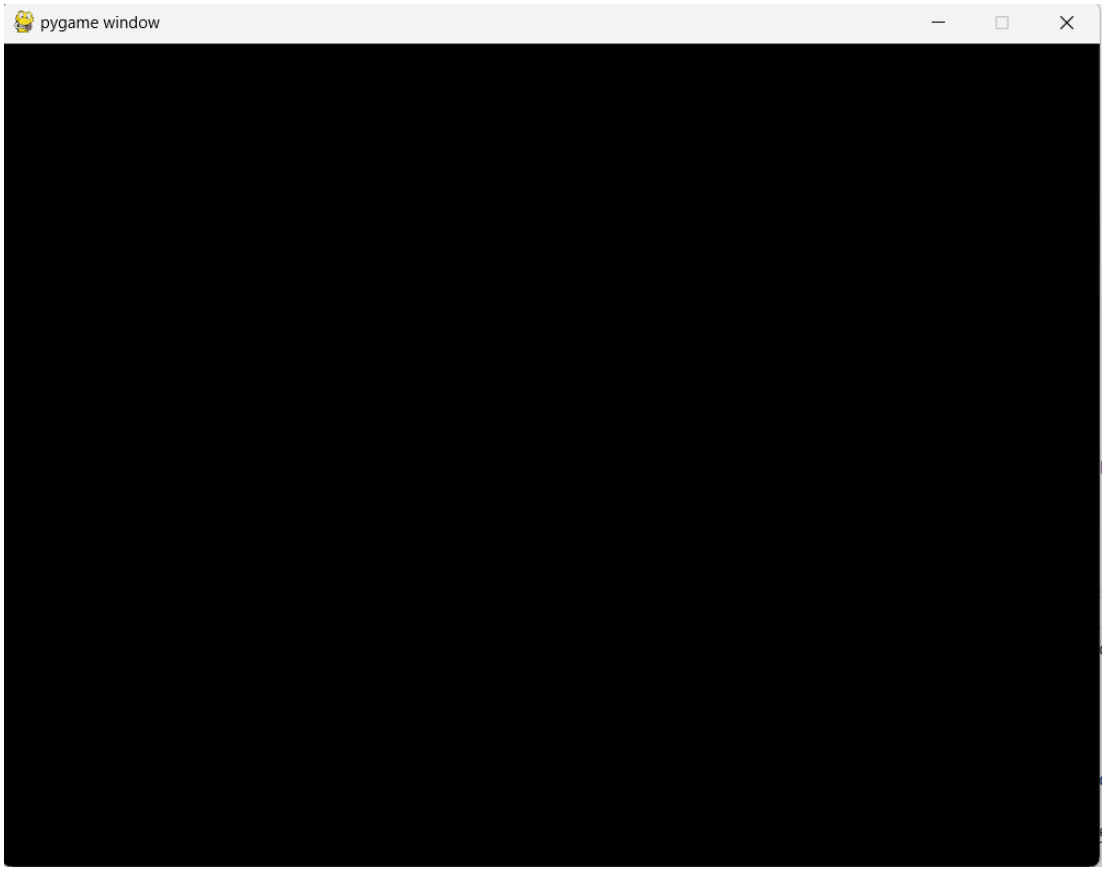
προβλήματα καθώς πρέπει να χρησιμοποιηθούν και να γίνεται επεξεργασία πολλαπλών εισόδων. Η λύση στην διαχείριση πολλαπλών εισόδων είναι η χρήση γεγονότων (**events**) μέσω του συστήματος γεγονότων του Pygame. Για την λήψη ενός γεγονότος, είτε αυτό είναι το πάτημα ενός πλήκτρου στο πληκτρολόγιο, είτε είναι η κίνηση του ποντικιού, πρέπει να χρησιμοποιηθεί ένας **event handler**. Όλα τα γεγονότα τοποθετούνται σε μια ουρά γεγονότων, καθώς είναι ασύγχρονα, και εξυπηρετούνται με την σειρά. Κάθε γεγονός έχει τον δικό του τύπο, και κάθε διαφορετικός τύπος συσχετίζεται με διαφορετικές τιμές δεδομένων.



```
running = True

while running:
    for event in pygame.event.get():
        if event.type == KEYDOWN:
            if event.key == K_ESCAPE:
                running = False
        elif event.type == QUIT:
            running = False
```

Εικόνα 15: Δημιουργία του game loop.

Με την προσθήκη των εντολών της Εικόνα 15 δημιουργείται το game loop και ξεκινά η διαχείριση των γεγονότων. Το παράθυρο που στην προηγούμενη εκτέλεση δεν εμφανιζόταν ποτέ, τώρα παραμένει μέχρι να τερματιστεί με το ποντίκι ή με το πλήκτρο ESCAPE.



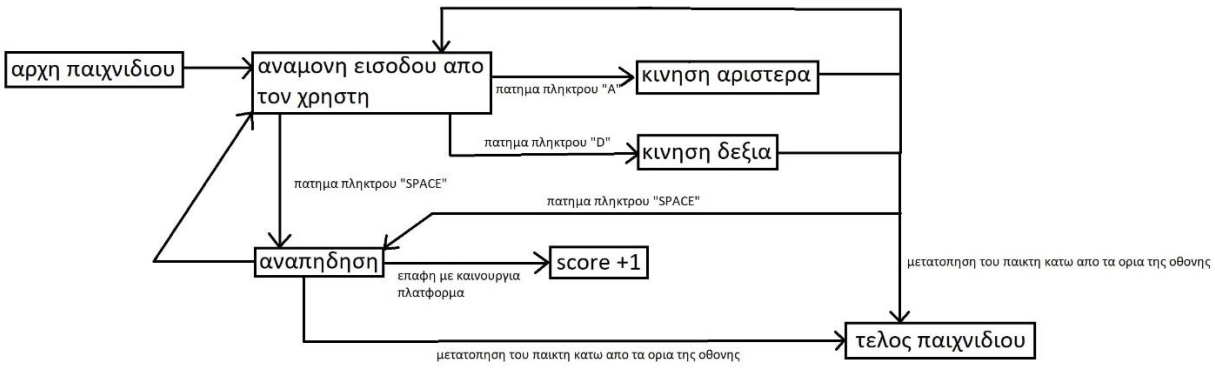
Εικόνα 16: Παράθυρο Pygame μετά από εκτέλεση του κώδικα δημιουργίας game loop.

Το επόμενο βήμα είναι η προσθήκη της λειτουργικότητας του παιχνιδιού, ο σχεδιασμός του περιβάλλοντος και η υλοποίηση ενός συστήματος πόντων. Αυτές οι λειτουργίες θα παρουσιαστούν στην επόμενη ενότητα.

ΚΕΦΑΛΑΙΟ 3 Tower Ascend

3.1 Σχεδίαση παιχνιδιού (Gameplay)

Για το πρώτο παιχνίδι θα δημιουργηθεί ένα απλό παιχνίδι τύπου platformer. Θα παρουσιαστεί ο τρόπος δημιουργίας των κλάσεων και των μεθόδων που θα ελέγχουν τους διάφορους μηχανισμούς του παιχνιδιού, αλλά και οι κατασκευή των μηχανισμών αυτών. Τα δύο βασικά μέρη του παιχνιδιού είναι ο χαρακτήρας που θα ελέγχει ο παίκτης, τον οποίο θα αναφερόμαστε ως παίκτη, και οι πλατφόρμες. Ο παίκτης θα πρέπει να έχει την ιδιότητα να χοροπηδάει πάνω στις πλατφόρμες, να έχει ένα συγκεκριμένο μέγεθος, και να επηρεάζεται από τη βαρύτητα, η οποία θα μπορεί να καθοριστεί από τον προγραμματιστή. Οι πλατφόρμες θα πρέπει να εμφανίζονται τυχαία όσο ο παίκτης σκαρφαλώνει όλο και πιο ψηλά. Η τυχαία τοποθέτησή τους γίνεται για να μην χρειάζεται η χειροκίνητη τοποθέτησή τους, αλλά και για να δώσουν μια επιπλέον δυσκολία στο παιχνίδι, αφού κάθε φορά που ο παίκτης θα χάνει και θα ξεκινάει από την αρχή, καμία νέα προσπάθεια δεν θα είναι ίδια με την προηγούμενη. Επίσης, θα προστεθεί ένα σύστημα βαθμολόγησης για να βλέπει ο παίκτης την πρόοδό του σε κάθε προσπάθεια. Για την αφηρημένη περιγραφή του πρώτου παιχνιδιού κατασκευάστηκε ένα δομικό διάγραμμα των διαδικασιών που εκτελούνται (βλ. Εικόνα 17).



Εικόνα 17: Αφηρημένο διάγραμμα διαδικασιών Tower Ascend.

3.2 Αρχική υλοποίηση παιχνιδιού (set-up)

Ο κώδικας ξεκινά κάνοντας import τις βιβλιοθήκες του Pygame και όλα τα modules από το **Pygame.locals**.

```python
import pygame
import sys
from pygame.locals import *
import random
import time

pygame.init()
#A TWO DIMENTIONAL VECTOR
vector = pygame.math.Vector2


#SCREEN SETTINGS
WIDTH = 400
HEIGHT = 450

#MOVEMENT RELATED VARIABLES
ACC = 0.5
FRIC = -0.12
FPS = 80


#THE FRAME COUNTER
FramePerSec = pygame.time.Clock()

screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("Tower Ascend")
```

Εικόνα 18: Αρχικοποίηση και προετοιμασία παραθύρου παιχνιδιού.

Καλείται η **pygame.init()** για την αρχικοποίηση του Pygame και ρυθμίζονται οι διάφορες σταθερές για το ύψος και το πλάτος της οθόνης. Ρυθμίζονται επίσης ένα ρολόι που θα χρησιμοποιηθεί αργότερα για να ελέγχονται τα frames per second. Με τις μεταβλητές **HEIGHT** και **WIDTH** αρχικοποιούνται οι διαστάσεις της οθόνης, και με τις **ACC** και **FRIC** θα ρυθμιστούν αργότερα η βαρύτητα και η κίνηση του παίκτη πάνω στις πλατφόρμες. Η μεταβλητή **vector** είναι ένας δισδιάστατος πίνακας που επίσης θα χρησιμοποιηθεί αργότερα. Η εντολή **FramePerSec = pygame.time.Clock()** θα χρησιμοποιηθεί για να περιοριστεί το frame rate του παιχνιδιού στην τιμή 60 fps. Η εντολή **screen = pygame.display.set_mode((WIDTH, HEIGHT))** δημιουργεί το παράθυρο του παιχνιδιού στις διαστάσεις που του δόθηκαν ως ορίσματα, και η εντολή

pygame.display.set_caption("Tower Ascend") δίνει το όνομα Tower Ascend στο παράθυρο. Αν ο κώδικας εκτελεστεί στην παρούσα του μορφή, εκτελείται χωρίς σφάλματα, ανοίγει και κλείνει ακαριαία ένα 450x400 pixel παράθυρο με ένα σκέτο μαύρο φόντο.

Στη συνέχεια, θα γίνει η δήλωση των δύο κλάσεων του παιχνιδιού, μία για τον παίκτη και μία για τις πλατφόρμες. Είναι καλό και οι δύο οντότητες να δηλωθούν ως διαφορετικές κλάσεις για να ελέγχονται πιο εύκολα τα περιεχόμενά τους.

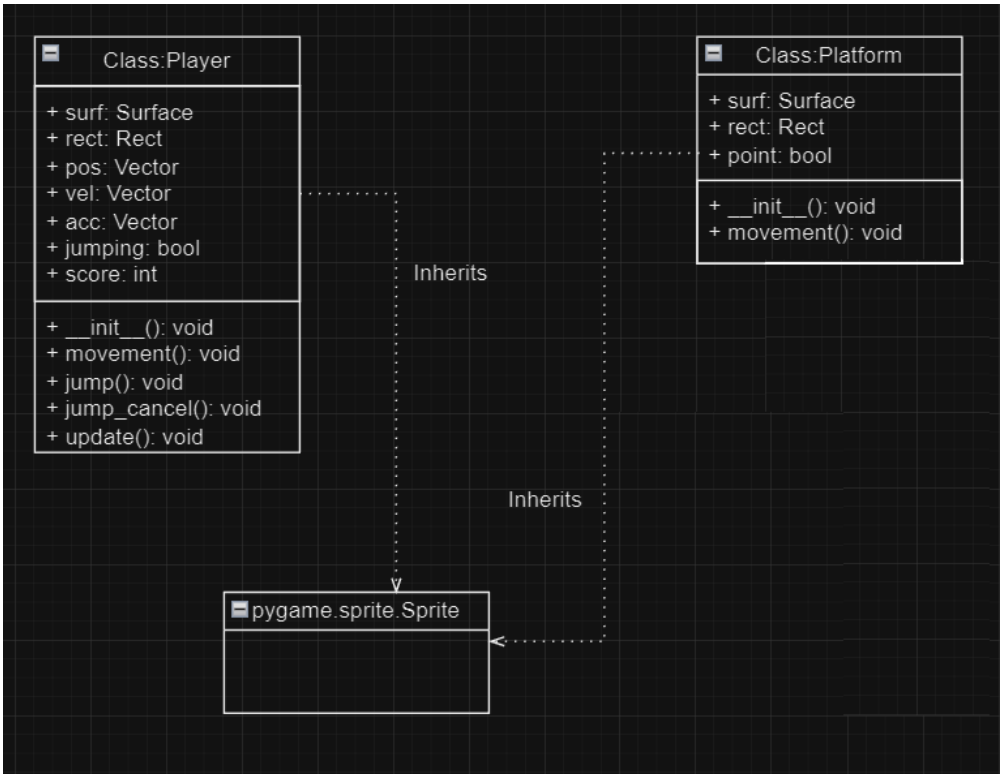
```python
class player(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        #SIZE OF PLAYER
        self.surf = pygame.Surface((30,30))
        self.surf.fill((128,255,40))
        #THE TOP LEFT CORNER OF OUR SCREEN IS THE 0,0 COORDS
        #ACCOUNTING FOR THE SIZE OF OUR PLAYER SQUARE THESE
        #COORDS PLACE IT ON THE BOTTOM LEFT CORNER FLAT ON
        #THE PLATFORM
        self.rect = self.surf.get_rect()

        self.pos = vector((15,430))
        self.vel = vector(0,0)
        self.acc = vector(0,0)
        self.jumping = False
        self.score = 0
```

Εικόνα 19: Κύρια δομή κλάσης παίκτη χωρίς τις μεθόδους.

```python
class platform(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.surf = pygame.Surface((random.randint(50,100), 12))
        self.surf.fill((255,50,0))
        self.rect = self.surf.get_rect(center = (random.randint(0, WIDTH-10),
                                                 random.randint(0, HEIGHT-30)))

        self.point = True
```

Εικόνα 20: Κύρια δομή κλάσης πλατφόρμας χωρίς τις μεθόδους.

Για την περιγραφή των κλάσεων σχεδιάστηκε το διάγραμμα κλάσεων του κώδικα. Παρατηρείται πως και οι δύο κλάσεις κληρονομούν την κλάση Sprite του Pygame.



Εικόνα 21: Διάγραμμα Κλάσεων (Class Diagram) του Tower Ascend.

Ξεκινώντας με τα βασικά χαρακτηριστικά και για τις δύο κλάσεις, δημιουργούνται οι επιφάνειες των αντικειμένων με την **pygame.Surface()** με μέγεθος 30x30 για το αντικείμενο της **Player** και **WIDTHx20** για την πρώτη πλατφόρμα. Δίνοντας το πλάτος της ίσο με **WIDTH**, η πλατφόρμα θα έχει πλάτος όσο είναι το πλάτος της οθόνης για να λειτουργήσει ως η αφετηρία του παιχνιδιού. Στη συνέχεια, με την **pygame.surf.fill()** χρωματίζονται οι επιφάνειες που δημιουργήθηκαν, και τέλος, με την **get_rect()** ανατίθεται ένα ορθογώνιο παραλληλόγραμμο στις επιφάνειές μας. Αυτό είναι πολύ σημαντικό, γιατί αργότερα αυτά τα ορθογώνια παραλληλόγραμμα θα χρησιμοποιηθούν για τον έλεγχο των συγκρούσεων μεταξύ των αντικειμένων.

```
PT1 = platform()
P1 = player()

all_sprites = pygame.sprite.Group()
all_sprites.add(PT1)
all_sprites.add(P1)

PT1.surf = pygame.Surface((WIDTH, 20))
PT1.surf.fill((255,0,0,))
PT1.rect = PT1.surf.get_rect(center = (WIDTH/2, HEIGHT-10))

platforms = pygame.sprite.Group()
platforms.add(PT1)

#PT1.moving = False
PT1.point = False

for x in range(random.randint(4,5)):
    C = True
    pl = platform()
    while C:
        pl = platform()
        C = clean_plats(pl, platforms)
    platforms.add(pl)
    all_sprites.add(pl)
```

Εικόνα 22: Τοποθέτηση αντικειμένων στο χώρο παιχνιδιού.

Με την ολοκλήρωση του σχεδιασμού των κλάσεων, κατασκευάζονται δύο αντικείμενα για κάθε κλάση που έχει δηλωθεί, ένα **PT1** για τις πλατφόρμες και ένα **P1** για τον παίκτη. Στη συνέχεια, δημιουργείται ένα sprite group μέσω της μεθόδου **pygame.sprite.Group()** και εκχωρείται στην μεταβλητή **all_sprites**. Έπειτα, τα αντικείμενα PT1 και P1 τοποθετούνται στο sprite group με την **all_sprites.add()** και τυπώνονται τυχαία πάνω στην οθόνη.

```
while True:
    P1.update()
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_SPACE:
                P1.jump()
        if event.type == pygame.KEYUP:
            if event.key == pygame.K_SPACE:
                P1.jump_cancel()
```

Εικόνα 23: Τμήμα συνθηκών τερματισμού game loop.

Το game loop εκτελείται μέχρι να εντοπίσει το **QUIT event** (βλ. Εικόνα 23). Η οθόνη γίνεται μαύρη με την **screen.fill((0, 0, 0))** και εμφανίζονται όλα τα sprites στο group που δημιουργήθηκε με τη δομή επανάληψης **for entity in all_sprites** (βλ. Εικόνα 24). Για την εμφάνιση των αλλαγών η **pygame.display.update()** ανανεώνει την οθόνη, ενώ η **FramePerSec.tick(FPS)** περιορίζει το refresh rate της οθόνης στα FPS που έχει οριστεί. Έχοντας ολοκληρώσει το αρχικό set up του παιχνιδιού, η υλοποίηση μπορεί να προχωρήσει στον προγραμματισμό της κίνησης του αντικειμένου της κλάσης **Player**.

```python
while True:
    P1.update()
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_SPACE:
                P1.jump()
        if event.type == pygame.KEYUP:
            if event.key == pygame.K_SPACE:
                P1.jump_cancel()

    if P1.rect.top > HEIGHT:
        for entity in all_sprites:
            entity.kill()
            time.sleep(1)
            screen.fill((255,5,5))
            pygame.display.update()
            time.sleep(1)
            pygame.quit()
            sys.exit()

    if P1.rect.top <= HEIGHT/3:
        P1.pos.y += abs(P1.vel.y)
        for plat in platforms:
            plat.rect.y += abs(P1.vel.y)
            if plat.rect.top >= HEIGHT:
                plat.kill()

    rand_plat_gen()
    screen.fill((0,0,0,))
    f = pygame.font.SysFont("Verdana", 20)
    g = f.render(str(P1.score), True, (123,255,0))
    screen.blit(g, (WIDTH/2, 10))

    for entity in all_sprites:
        screen.blit(entity.surf, entity.rect)
        entity.movement()

    pygame.display.update()
    FramePerSec.tick(FPS)
```

Εικόνα 24: Συνολικός κώδικας game loop.

3.3 Προγραμματισμός κίνησης

Πρώτα, για την κλάση **Player** δηλώνονται τρεις μεταβλητές: η **self.pos** για τη θέση, η **self.vel** για την ταχύτητα, και η **self.acc** για την επιτάχυνση (βλ. Εικόνα 19). Όλες οι μεταβλητές αρχικοποιούνται με (0, 0), εκτός απ' την self.pos, που μπορεί να πάρει οποιαδήποτε τιμή.

Στην παρούσα υλοποίηση επιλέγονται οι συντεταγμένες (15, 430), τοποθετώντας τον παίκτη στην κάτω αριστερή γωνία της οθόνης. Οι μεταβλητές **vel** και **acc** είναι επίσης δύο διαστάσεων, αφού θα υπάρχει κίνηση σε δύο άξονες. Ο πρώτος αριθμός είναι η τιμή για τον άξονα x και ο δεύτερος για τον άξονα y.

```
#MOVEMENT FUNCTION
    def movement(self):
        #UNIVERSAL ACCELERATION HORIZONTAL IS 0 AND
        #VERTICAL IS 0.5 FOR THE EFFECT OF GRAVITY
        self.acc = vector(0,0.5)

        pressed_keys = pygame.key.get_pressed()
        if pressed_keys[K_a]:
            self.acc.x = -ACC
        if pressed_keys[K_d]:
            self.acc.x = ACC

        self.acc.x += self.vel.x * FRIC
        self.vel += self.acc
        self.pos += self.vel + 0.5 * self.acc

        #SCREEN LOOPING
        if self.pos.x > WIDTH:
            self.pos.x = 0
        if self.pos.x < 0:
            self.pos.x = WIDTH

        #THIS LINE UPDATES THE rect() OBJECT OF THE
        # PLAYER WITH THE NEW POSITION THAT IT HAS
        # GAINED AFTER BEING MOVED
        self.rect.midbottom = self.pos
```

Εικόνα 25: μέθοδος κίνησης παίκτη.

Στη συνέχεια, κατασκευάζεται την πρώτη μέθοδο για την κίνηση του **player** με όνομα **movement**. Με την **self.acc = vector(0, 0.5)** κάνουμε reset τις τιμές της επιτάχυνσης με 0 για την οριζόντια και 0.5 για την κάθετη κίνηση, προσομοιώνοντας τη βαρύτητα. Για αρχή, είναι καλύτερο να διατηρηθούν και οι δύο τιμές στο 0 για να μην εξαφανίζεται ο παίκτης κάτω από την οθόνη, κάθε φορά που εκτελείται το πρόγραμμα. Έπειτα, ελέγχεται αν έχει πατηθεί το πλήκτρο “a” ή “d” και αλλάζει η τιμή της επιτάχυνσης στον άξονα x με την τιμή **ACC**. Οι επόμενες τρεις γραμμές κώδικα αποτελούν μια συνάρτηση κίνησης που χρησιμοποιεί την

τιμή **FRIC** για να προσομοιώσει την τριβή, ώστε να μπορεί ο παίκτης να επιβραδύνει όταν δεν είναι πατημένο το κουμπί. Παρακάτω, παρουσιάζονται δύο δομές επιλογής που ελέγχουν τη θέση του παίκτη και, αν είναι τέρμα δεξιά ή τέρμα αριστερά, τον μεταφέρουν στην αντίθετη πλευρά της οθόνης. Αυτή η δυνατότητα είναι προαιρετική και μπορεί να παραλειφθεί, αλλά αντί αυτού μπορούν τα αλλαχθούν οι δομές επιλογής ώστε να μην επιτρέπουν στον παίκτη να βγει έξω από τα όρια της οθόνης. Τέλος, η εντολή **self.rect.midbottom = self.pos** ανανεώνει τη θέση του του παίκτη μετά την κίνησή του.

```
#COLLITION DETECTION
    def update(self):
        hits = pygame.sprite.spritecollide(P1, platforms, False)
        if self.vel.y > 0:
            if hits:
                if self.pos.y < hits[0].rect.bottom:
                    if hits[0].point == True:
                        hits[0].point = False
                        self.score += 1
                    self.pos.y = hits[0].rect.top +1
                    self.vel.y = 0
                    self.jumping = False
```

Εικόνα 26: Μέθοδος ανίχνευσης συγκρούσεων (collision detection).

Με την επιτυχή εμφάνιση όλων των αντικειμένων στην οθόνη θα γίνει υλοποίηση της ανίχνευσης σύγκρουσης (collision detection), για την αλληλεπίδραση των αντικειμένων μεταξύ τους. Με την εντολή **spritecollide()** που παρέχεται από το pygame, απλοποιείται ο έλεγχος για το αν δύο sprites είναι σε επαφή. Αφού διαπιστωθεί αν υπάρχει επαφή, τότε μπορεί να παρθεί η απόφαση για την μεταξύ τους αλληλεπίδραση. Η **spritecollide(sprite, sprite_group, Boolean)** παίρνει τρία ορίσματα: το πρώτο είναι ένα **sprite**, όπως ο **player**, το δεύτερο είναι ένα **sprite group**, όπως το **all_sprites**, και το τρίτο είναι μία τιμή True ή False για την ενεργοποίηση της διαγραφής του sprite που δόθηκε στο πρώτο όρισμα. Χρησιμοποιώντας αυτή τη λογική, στη μέθοδο **update**, δηλώνεται μία μεταβλητή **hits** και της εκχωρείται το αποτέλεσμα της **spritecollide** ανάμεσα στον P1 και το sprite group των πλατφορμών. Έπειτα, σε δομή επιλογής ελέγχεται η ύπαρξη σύγκρουσης και αν υπάρχει, μηδενίζεται η κάθετη ταχύτητα του παίκτη και αυτός μεταφέρεται στο πάνω μέρος της πλατφόρμας.

```
#JUMP FUNTION CHECKS TO SEE IF THE PLAYER IS IN
    #CONTACT WITH A PLATFORM
    #THIS PREVENTS DOUBLE JUMPING
    def jump(self):
        hits = pygame.sprite.spritecollide(self, platforms, False)
        if hits and not self.jumping:
            self.jumping = True
            self.vel.y = -15
    #THE JUMP CANNCEL FUNTION ALLOWS SHORT JUMPS
    def jump_cancel(self):
        if self.jumping:
            if self.vel.y < -3:
                self.vel.y = -3
```

Εικόνα 27: Μέθοδος κίνησης άλματος για τον παίκτη.

Τώρα υποστηρίζεται η σύγκρουση μεταξύ παίκτη και πλατφορμών, ακολουθεί η κατασκευή της συνάρτησης που εκτελεί το άλμα του παίκτη, που είναι ο βασικός μηχανισμός του παιχνιδιού. Για το άλμα θα δημιουργηθεί μια καινούργια μέθοδος και θα κληθεί μέσα στο game loop κάθε φορά που πιέζεται το πλήκτρο **space**, ακριβώς όπως έγινε και για την κίνηση στο στην μέθοδο **movement**. Αντίστοιχα με την μέθοδο **movement**, δίνεται κάθετη ταχύτητα στον παίκτη αντί για οριζόντια και αφού έχει οριστεί κάθετη επιτάχυνση, για την προσομοίωση της βαρύτητας, η κάθετη ταχύτητα αυτή μειώνεται σταδιακά και προσγειώνει τον παίκτη στο έδαφος.

Αν γίνει εκτέλεση του παιχνιδιού σε αυτό το σημείο, θα παρατηρηθεί πως δίνεται η δυνατότητα στον παίκτη να αναπηδά στον αέρα, χωρίς κάποια εμφανή πλατφόρμα για να σταθεί. Αυτό διορθώνεται με μία απλή γραμμή κώδικα στην συνάρτηση **jump**, η οποία ελέγχει αν υπάρχει σύγκρουση ανάμεσα στον παίκτη και κάποια πλατφόρμα με μια δομή επιλογής, ώστε το άλμα να εκτελείται μόνο όταν εντοπίζεται πρώτα σύγκρουση ανάμεσα στα δύο αντικείμενα. Τέλος, προστίθεται μία γραμμή κώδικα στην συνάρτηση **update()** για να επιλυθεί ένα σφάλμα που δημιουργείται ανάμεσα στο στις συναρτήσεις **update()** και **jump()**, όπου μηδενίζεται η κάθετη ταχύτητα του παίκτη κάθε φορά που έρχεται σε επαφή με κάποια πλατφόρμα.

3.4 Προγραμματισμός πλατφορμών

Μέχρι αυτό το σημείο έχει δημιουργηθεί μόνο η αρχική πλατφόρμα. Στόχος τώρα είναι η κατασκευή των υπόλοιπων πλατφορμών και να δημιουργηθεί ένας τρόπος να δημιουργούνται πλατφόρμες σε τυχαίες θέσεις και σε τυχαίο μέγεθος πάνω από τον παίκτη.

```
1  for x in range(random.randint(4,5)):
2      C = True
3      pl = platform()
4      while C:
5          pl = platform()
6          C = clean_plats(pl, platforms)
7      platforms.add(pl)
8      all_sprites.add(pl)
```

Εικόνα 28: Τυχαία τοποθέτησης πλατφορμών σε κάθε νέο παιχνίδι.

Σε μια δομή επιλογής και με τη χρήση της συνάρτησης **range** και της ενότητας **random**, γίνεται εκτέλεση του κώδικα τυχαία 4 με 5 φορές για να γίνεται κάθε παιχνίδι πιο ενδιαφέρον. Έπειτα, ανανεώνεται ένα αντικείμενο πλατφόρμας, αλλάζοντας τα ορίσματα στην **self.surf** και **self.rect**, και, ξανά με τη χρήση της **random.randint**, δηλώνονται τα όρια μέσα στα οποία μια πλατφόρμα μπορεί να δημιουργηθεί σε μέγεθος και θέση στην οθόνη. Αφού όμως έγινε αλλαγή στην κλάση Platform, πρέπει να δηλωθεί ξανά η αρχική πλατφόρμα σε ξεχωριστό κώδικα εκτός από την κλάση. Τρέχοντας τον κώδικα παρατηρείται πως δημιουργούνται τυχαίες πλατφόρμες στην οθόνη, αλλά δεν υπάρχει άλλη λειτουργικότητα πέρα από την ανάβαση σε αυτές. Επόμενο μέλημα είναι ο προγραμματισμός του infinite scrolling feature, που θα επιτρέπει να την συνέχιση του παιχνιδιού επ’ άπειρον.

```
if P1.rect.top > HEIGHT:
        for entity in all_sprites:
            entity.kill()
            time.sleep(1)
            screen.fill((255,5,5))
            pygame.display.update()
            time.sleep(1)
            pygame.quit()
            sys.exit()

    if P1.rect.top <= HEIGHT/3:
        P1.pos.y += abs(P1.vel.y)
        for plat in platforms:
            plat.rect.y += abs(P1.vel.y)
            if plat.rect.top >= HEIGHT:
                plat.kill()
```

Εικόνα 29: Σχεδιασμός infinite scroll.

Σε μια δομή επιλογής μέσα στο game loop ελέγχεται η θέση του παίκτη σε σχέση με το ύψος της οθόνης. Έπειτα, με τη χρήση της **abs()** υπολογίζεται η απόλυτη θέση του παίκτη στην οθόνη, χρησιμοποιώντας το απόλυτη τιμή της ταχύτητάς του. Αυτό γίνεται για να ανανεώνεται η θέση του παίκτη σε σχέση με την οθόνη. Στη συνέχεια γίνεται το ίδιο για κάθε πλατφόρμα και αν κάποια πλατφόρμα κατέβει κάτω από τα όρια της οθόνης, καλούμε την μέθοδο **kill()** για να αφαιρεθεί από το παιχνίδι εντελώς. Εδώ πρέπει να επισημανθεί πως, επειδή δεν εμφανίζεται κάποιο αντικείμενο στην οθόνη επειδή μεταφέρθηκε εκτός των ορίων της, αυτό δεν σημαίνει πως δεν υπάρχει. Η **kill()** εκτός από το ότι εξασφαλίζει πως αν ο παίκτης έπεφτε, θα προσγειωνόταν σε μια προηγούμενη πλατφόρμα, ταυτόχρονα εξοικονομεί πόρους για να κάνει το παιχνίδι να τρέχει καλύτερα, αφαιρώντας άχρηστα αντικείμενα.

```
def rand_plat_gen():
        while len(platforms) < 6:
            width = random.randrange(50,100)
            p = platform()
            C = True

            while C:
                p = platform()
                p.rect.center = (random.randrange(0, WIDTH - width), random.randrange(-50, 0))
                C = clean_plats(p, platforms)
            platforms.add(p)
            all_sprites.add(p)
```

Εικόνα 30: Συνάρτηση παραγωγής τυχαίων πλατφορμών.

Τέλος, θα κατασκευαστεί η συνάρτηση που θα δημιουργεί τυχαία πλατφόρμες, οι οποίες θα εμφανίζονται όταν η οθόνη θα μετακινείται πιο ψηλά, ακολουθώντας την ανάβαση του παίκτη. Σε μια καινούργια μέθοδο, **rand_plat_gen()**, εκτελείται μια δομή επανάληψης, η οποία ενεργοποιείται όταν στην οθόνη υπάρχουν λιγότερες από 7 πλατφόρμες. Όταν συμβεί αυτό, δηλώνεται ένα τυχαίο πλάτος και δημιουργείται καινούργια πλατφόρμα για αυτό το πλάτος.

3.5 Ολοκλήρωση ανάπτυξης παιχνιδιού Tower Ascend

Σε αυτό το σημείο έχουν ολοκληρωθεί όλοι οι βασικοί μηχανισμοί του παιχνιδιού και ακολουθεί η προσθήκη τελευταίων λεπτομερειών που βελτιώνουν την εμπειρία χρήσης του παιχνιδιού. Η πρώτη τροποποίηση θα κάνει το άλμα πιο δυναμικό. Συγκεκριμένα, η διάρκεια του άλματος θα είναι ανάλογη του χρόνου που παραμένει πατημένο το πλήκτρο space. Αντίστοιχα, το άλμα σταματάει μόλις σταματήσει να είναι πατημένο το πλήκτρο space.

```
#THE JUMP CANNCEL FUNTION ALLOWS SHORT JUMPS
    def jump_cancel(self):
        if self.jumping:
            if self.vel.y < -3:
                self.vel.y = -3
```

Εικόνα 31: Κώδικας βελτίωσης άλματος.

Πρώτα απ’ όλα, πρέπει να δηλωθεί μια μεταβλητή **self.jumping** με αρχικοποίηση στην τιμή **False** στην κλάση player (βλ. Εικόνα 19). Μετά, μέσα στην μέθοδο **jump()**, θα προστεθεί στην συνθήκη μια έκφραση and not self.jumping πάνω στην υπάρχουσα δομή επιλογής, που

θα ελέγχει ότι η self.jumping είναι False, και μέσα στο σώμα της επιλογής, η μεταβλητή **self.jumping** τίθεται στην τιμή **True**.

Στη συνέχεια θα δημιουργηθεί μια ξεχωριστή μέθοδος με το όνομα **jump_cancel()**, που αναλαμβάνει να μειώσει την ταχύτητα του παίκτη, και θα κληθεί μέσα στο game loop σε συνδυασμό με μια δομή επιλογής, στην οποία εισέρχεται η εκτέλεση κάθε φορά που αποδεσμεύεται το πλήκτρο **space**. Η διαφορά των δύο δομών επιλογής είναι πως η μέθοδος **jump()** καλείται όταν το **event.type** είναι ίσο με **pygame.KEYDOWN**, ενώ το **jump_cancel()** καλείται όταν το **event.type** είναι ίσο με **pygame.KEYUP**. Τέλος, μέσα στην **update()** εκτελείται η εντολή εκχώρησης **self.jumping = False**.

Ένα μεγάλο πρόβλημα που έχει το παιχνίδι, μέχρι αυτό το σημείο, είναι πως υπάρχει η πιθανότητα δύο ή περισσότερες πλατφόρμες να εμφανιστούν πολύ κοντά η μία με την άλλη ή ακόμα και πάνω η μία στην άλλη οδηγώντας από μερική μέχρι και πλήρη επικάλυψή τους.

```
def clean_plats(platform, groupies):
    if pygame.sprite.spritecollideany(platform, groupies):
        return True
    else:
        for entity in groupies:
            if entity == platform:
                continue
            if (abs(platform.rect.top - entity.rect.bottom) < 40) and (abs(platform.rect.bottom - entity.rect.top)< 50):
                return True
        C = False
```

Εικόνα 32: Αποφυγή επικάλυψης πλατφόρμων.

Για να το διορθωθεί το ζήτημα αυτό, θα κατασκευαστεί μια καινούργια μέθοδος με όνομα **clean_plats()**, που παίρνει ως ορίσματα μια καινούργια πλατφόρμα και το sprite group. Μέσα στη μέθοδο ορίζεται μια δομή επιλογής που ελέγχει, με τη χρήση της **spritecollideany()**, αν δύο πλατφόρμες επικαλύπτονται. Στην περίπτωση που επικαλύπτονται, επιστρέφεται η τιμή True. Αυτή η αλλαγή και μόνο λύνει το πρόβλημα των επικαλυπτόμενων πλατφορμών, αλλά είναι εφικτή η βελτιστοποίηση, βάζοντας μια εναλλακτική επιλογή στην αρχική δομή επιλογής, που για όλες τις οντότητες στο sprite group, αν μια οντότητα είναι πλατφόρμα, ελέγχεται ο χώρος γύρω από αυτή και με τη χρήση της συνάρτησης **abs()**, ελέγχεται ότι δεν υπάρχει άλλη πλατφόρμα σε απόλυτη απόσταση 65 pixels από πάνω ή από κάτω. Τέλος, εκχωρούμε στη μεταβλητή Check την τιμή False. Για να λειτουργήσουν οι αλλαγές όμως πρέπει να τροποποιηθεί και η συνάρτηση **rand_plat_gen()**.

```
while C:
    p = platform()
    p.rect.center = (random.randrange(0, WIDTH - width),
                     random.randrange(-50, 0))
    C = clean_plats(p, platforms)
```

Εικόνα 33: Προσθήκες στην συνάρτηση rand_plat_gen().

Εδώ, για αρχή, εκχωρείται στην μεταβλητή Check η τιμή True και προστίθενται δύο νέες εντολές, η **p = platform()**, που δημιουργεί ένα νέο αντικείμενο της κλάσης Platform και μια εκχώρηση στο πεδίο **p.rect.center**, που γίνεται ίσο με το κέντρο της πλατφόρμας, μέσα σε μια δομή επανάληψης, που τρέχει όσο η **clean_plats()** επιστρέφει την τιμή True. Όταν τελικά επιστραφεί τιμή False, η πλατφόρμα τοποθετείται στο sprite group. Τέλος, προστίθενται μερικές γραμμές κώδικα μέσα στην **update()** για να διορθωθεί ένα σφάλμα που παρουσιάζεται όταν ο παίκτης δεν καταφέρνει να προσγειωθεί σε μια πλατφόρμα και φτάνει μόνο μέχρι τη μέση της διαδρομής.

```
1  if self.pos.y < hits[0].rect.bottom:
2                      if hits[0].point == True:
3                          hits[0].point = False
4                          self.score += 1
```

Εικόνα 34: Επίλυση προβλήματος τηλεμεταφοράς.

Το σφάλμα που συμβαίνει είναι πως ο παίκτης μπορεί να τηλεμεταφερθεί σε μια πλατφόρμα. Αυτό λύνεται με δύο απλές γραμμές κώδικα: η πρώτη είναι μια δομή επιλογής που ελέγχει αν η θέση του παίκτη στον κατακόρυφο άξονα είναι πάνω από το χαμηλότερο σημείο της πλατφόρμας. Τότε κάνει τη θέση του ίση με το υψηλότερο σημείο + 1.

```
if P1.rect.top > HEIGHT:
        for entity in all_sprites:
            entity.kill()
            time.sleep(1)
            screen.fill((255,5,5))
            pygame.display.update()
            time.sleep(1)
            pygame.quit()
            sys.exit()
```

Εικόνα 35: Οθόνη τερματισμού.

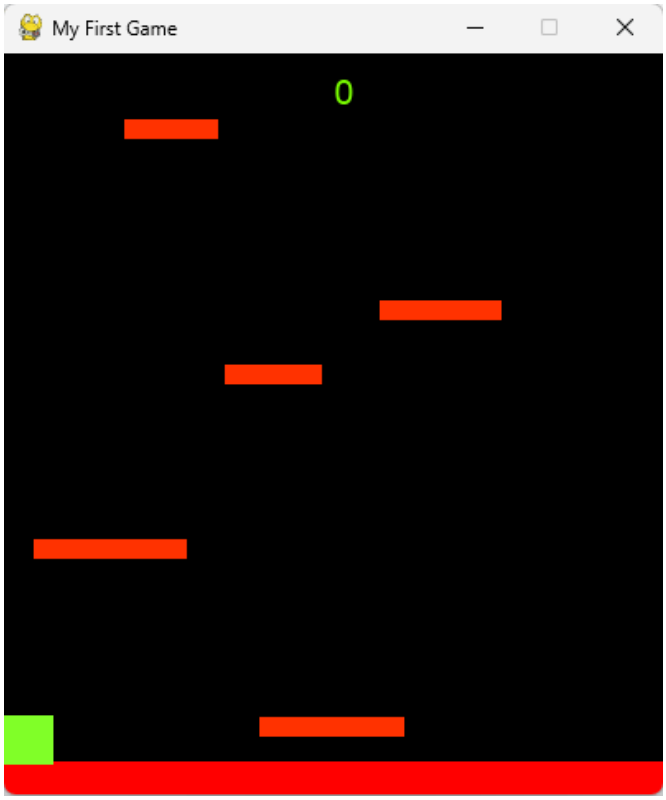
Το βασικό πρόγραμμα του παιχνιδιού είναι έτοιμο. Το μόνο που του λείπει είναι ένα game over screen και ένας τρόπος να μετριέται η πρόοδος του παίκτη. Γι' αυτό, θα γίνεται προσθήκη ενός συστήματος βαθμολόγησης (scoring system) που θα ανεβαίνει κάθε φορά που ο παίκτης προσγειώνεται σε μια καινούργια πλατφόρμα, και ένα απλό game over screen που κάνει την οθόνη κόκκινη όταν ο παίκτης πέφτει κάτω από τα όρια της οθόνης. Πρώτα, για το game over screen υπάρχει μια δομή επιλογής μέσα στο game loop, που ελέγχει το πάνω μέρος του αντικειμένου της κλάσης Player σε σχέση με το κάτω μέρος της οθόνης, σβήνει όλα τα sprites από την οθόνη και, μετά από ένα μικρό delay ενός δευτερολέπτου με την **time.sleep(1)**, χρωματίζει την οθόνη με κόκκινο, περιμένοντας άλλο 1 δευτερόλεπτο προτού τερματιστεί το game loop.

Για το high scoring system, αρχικά δηλώνεται μια καινούργια μεταβλητή στην κλάση Player ονόματι self.score = 0. Έπειτα, μέσα στην κλάση Platform, δημιουργείται η τιμή self.point και της ανατίθεται η τιμή True. Η τιμή True σημαίνει πως η πλατφόρμα θα δώσει βαθμολογία, αλλά αν είναι False, τότε δεν δίνει κάποια βαθμολογία. Μέσα στην update ορίζεται μια δομή επιλογής για τον έλεγχο του αν η self.point είναι True, και αν είναι, αυτή γίνεται False για αυτό το platform και το score αυξάνεται κατά 1. Τέλος, με την PT1.point = False ορίζεται η αρχική πλατφόρμα να μην δίνει κάποια βαθμολογία.

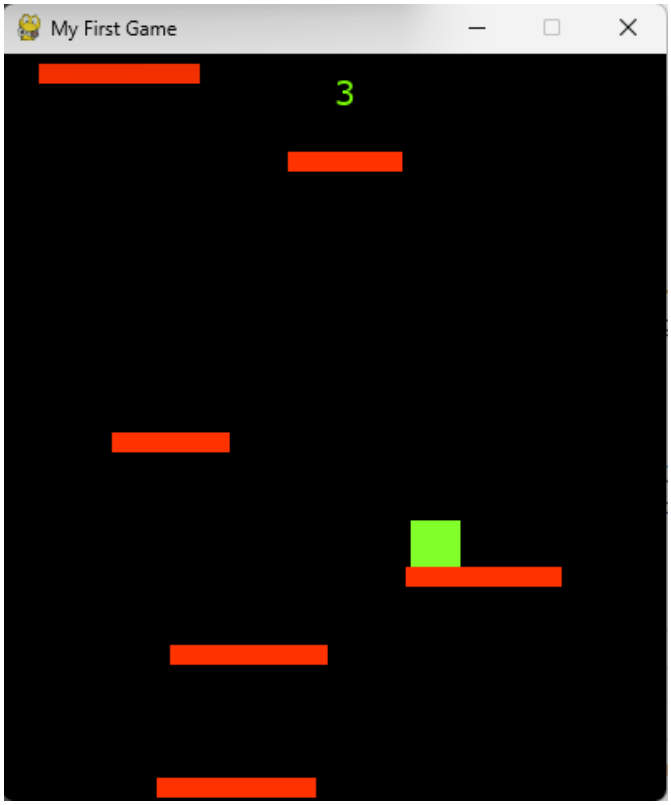
```
f = pygame.font.SysFont("Verdana", 20)
    g = f.render(str(P1.score), True, (123,255,0))
    screen.blit(g, (WIDTH/2, 10))
```

Εικόνα 36: Εμφάνιση σκορ στην οθόνη.

Το μόνο που έμεινε είναι να εμφανίζεται το score στην οθόνη. Έτσι, μέσα στο game loop, σε μια μεταβλητή font, επιλέγεται το Verdana font και το μέγεθος της γραμματοσειράς με την εντολή pygame.font.SysFont(Verdana, 20). Δημιουργείται ένα αντικείμενο g με την εντολή font.render, η οποία ως ορίσματα παίρνει το score, που πρέπει να μετατραπεί σε string με την str(), την τιμή True, που είναι για το anti-aliasing, κάνοντας τις άκρες του string πιο ομαλές, και ένα χρώμα. Τέλος, εμφανίζεται το αντικείμενο g στην οθόνη με την screen.blit(), τοποθετώντας το στο κέντρο του πάνω μέρους της οθόνης.



Εικόνα 37: Αρχική οθόνη παιχνιδιού Tower Ascend.



Εικόνα 38: Στιγμιότυπο gameplay παιχνιδιού Tower Ascend.

ΚΕΦΑΛΑΙΟ 4 Alien Invaders

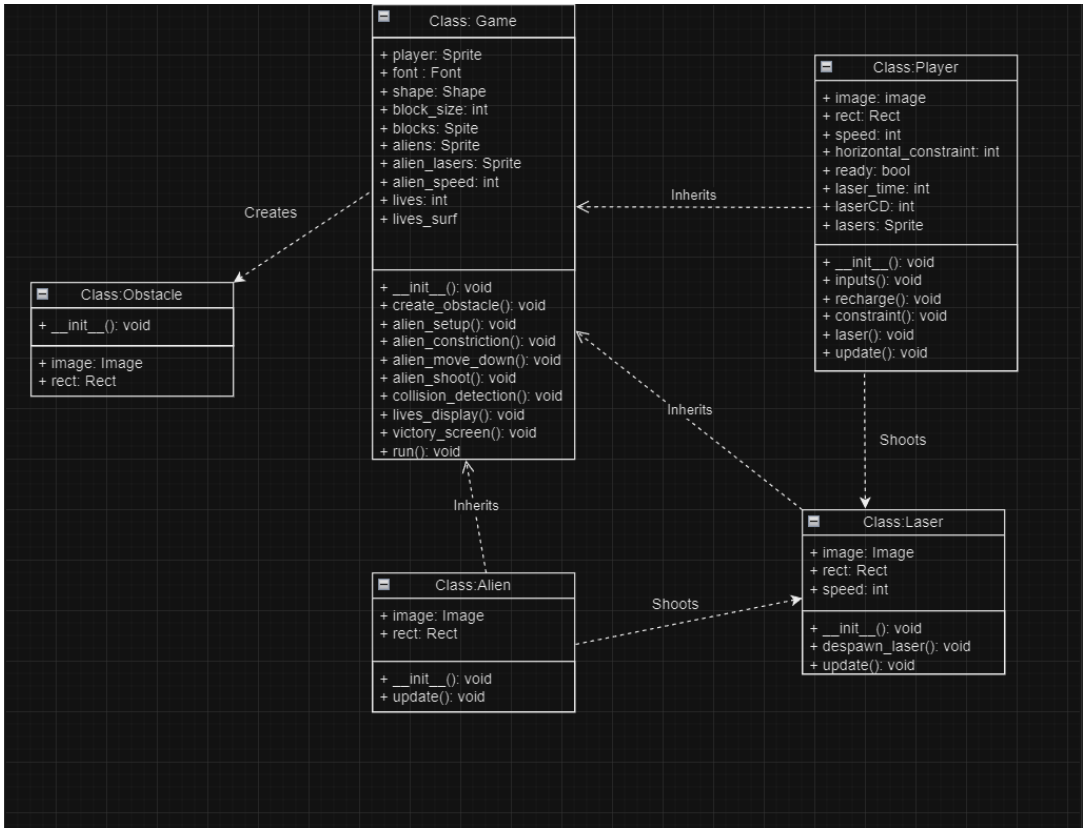
4.1 Σχεδίαση παιχνιδιού (Gameplay)

Το παιχνίδι ακολουθεί το κλασικό πρότυπο ενός shooter παιχνιδιού, όπου ο παίκτης ελέγχει ένα διαστημόπλοιο που πρέπει να καταστρέψει εχθρούς (εξωγήινους) και να αποφύγει τα εμπόδια και τις εχθρικές επιθέσεις (βλ. Εικόνα 45). Η δομή του κώδικα είναι χωρισμένη σε πέντε κύριες κλάσεις, οι οποίες περιγράφουν τα αντικείμενα και τη λειτουργικότητα του παιχνιδιού: **Game**, **Player**, **Obstacle**, **Laser** και **Alien**. Κάθε μία από αυτές τις κλάσεις έχει συγκεκριμένες αρμοδιότητες και συνεργάζεται με τις υπόλοιπες για την επίτευξη της επιθυμητής συμπεριφοράς του παιχνιδιού. Για την καλύτερη κατανόηση του gameplay σχεδιάστηκε δομικό διάγραμμα διαδικασιών, που προσφέρει μια αφηρημένη περιγραφή του παιχνιδιού (βλ. Εικόνα 39).

Εικόνα 39: Αφηρημένο δομικό διάγραμμα διαδικασιών παιχνιδιού.

4.2 Καθορισμός κλάσεων

Το σύνολο των κλάσεων περιγράφεται από το παρακάτω διάγραμμα κλάσεων (βλ. Εικόνα 40). Στο διάγραμμα παρουσιάζονται οι σχέσεις κληρονομικότητας και αλληλεπίδρασης μεταξύ των κλάσεων της υλοποίησης μας.



Εικόνα 40: Διάγραμμα κλάσεων Space Invaders.

Η κύρια κλάση **Game** διαχειρίζεται τη συνολική λογική του παιχνιδιού. Η αρχικοποίηση της κλάσης περιλαμβάνει τη ρύθμιση του παίκτη, τη δημιουργία εμποδίων και εξωγήινων, καθώς και την παρακολούθηση του συστήματος ζωών του παίκτη.

```python
#THE GAME CLASS CONTAINS ALL OFa THE GAME LOGIC
class Game:
    def __init__(self):
        #PLAYER SETUP
        player_sprite = Player((WIDTH/2, HEIGHT), WIDTH, 5)
        self.player = pygame.sprite.GroupSingle(player_sprite)

        self.font = pygame.font.SysFont('Verdana',20)
        #OBSTACLE SETUP
        self.shape = obstacle.shape
        self.block_size = 6
        self.blocks = pygame.sprite.Group()
        #CREATING THE OBSTACLES
        self.create_obstacle(50,450)
        self.create_obstacle(485,450)
        self.create_obstacle(267,420)
        #ALIEN SETUP
        self.aliens = pygame.sprite.Group()
        self.alien_lasers = pygame.sprite.Group()
        self.alien_setup(rows = 5, columns = 8) #5,8
        self.alien_speed = 1
        #HEALTH SYSTEM
        self.lives = 3
        self.lives_surf = pygame.image.load('player.png').convert_alpha()
        self.lives_position = WIDTH - (self.lives_surf.get_size()[0] * 2 + 475)
```

Εικόνα 41: Κλάση Game.

Ο παίκτης εμφανίζεται στο κάτω μέρος της οθόνης και ελέγχεται από τον χρήστη. Η μέθοδος **create_obstacle** δημιουργεί τα εμπόδια στο παιχνίδι, τοποθετώντας τα σε συγκεκριμένες θέσεις, ενώ η **alien_setup** οργανώνει τους εξωγήινους σε σειρές και στήλες με διαφορετικά χρώματα, τα οποία υποδεικνύουν την δυσκολία των εχθρών.

```
def create_obstacle(self, x_pos, y_pos):
        for row_pointer, row in enumerate(self.shape):
            for column_pointer, column in enumerate(row):
                if column == 'x':
                    x = x_pos + column_pointer * self.block_size
                    y = y_pos + row_pointer * self.block_size
                    block = obstacle.Obstacle(self.block_size, (241,79,80), x, y)
                    self.blocks.add(block)
```

Εικόνα 42: Μέθοδος δημιουργίας εμποδίων.

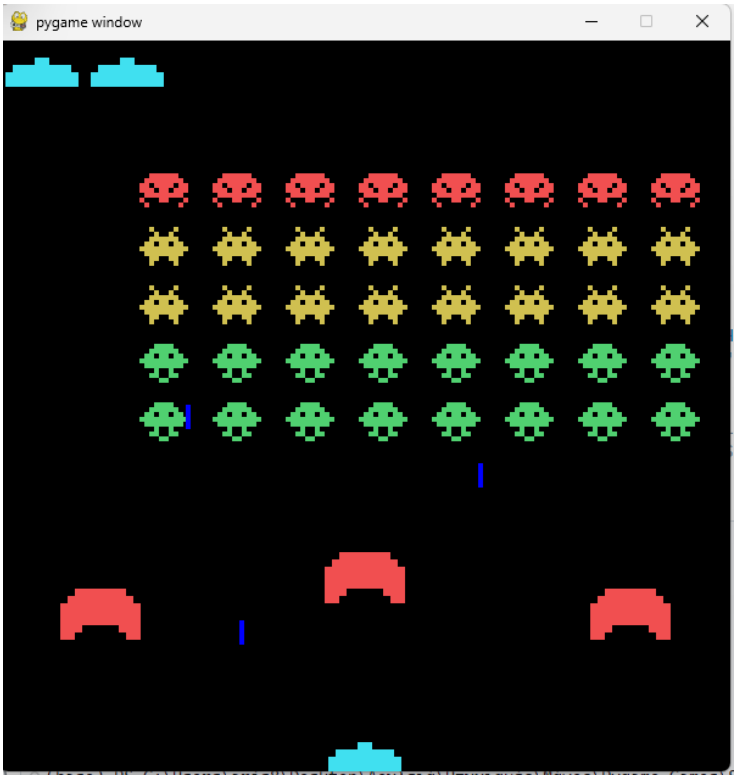
```
def alien_setup(self, rows, columns, x_dis = 60, y_dis = 48, x_offset = 70, y_offset = 100):
        for row_pointer, row in enumerate(range(rows)):
            for column_pointer, column in enumerate(range(columns)):
                x = column_pointer * x_dis + x_offset
                y = row_pointer * y_dis + y_offset

                if row_pointer == 0: alien_sprite = Alien('red',x,y)
                elif 1 <= row_pointer <=2: alien_sprite = Alien('yellow',x,y)
                else: alien_sprite = Alien('green',x,y)

                self.aliens.add(alien_sprite)
```

Εικόνα 43: Μέθοδος τοποθέτησης εξωγήινων.

Η κλάση αυτή διαχειρίζεται επίσης τον έλεγχο των συγκρούσεων μέσω της μεθόδου **collision_detection**, η οποία παρακολουθεί τις αλληλεπιδράσεις μεταξύ των λέιζερ, των εξωγήινων, των εμποδίων και του παίκτη. Στην περίπτωση που ο παίκτης δεχθεί χτυπήματα, το σύστημα ζωών μειώνεται, ενώ όταν εξαντληθούν οι ζωές, το παιχνίδι τερματίζεται. Επιπλέον, η μέθοδος **victory_screen** εμφανίζει μήνυμα νίκης αν ο παίκτης καταφέρει να καταστρέψει όλους τους εξωγήινους. Τέλος, η **run** είναι η κύρια μέθοδος που διαχειρίζεται την ενημέρωση της κατάστασης του παιχνιδιού σε κάθε **frame**.

```python
def collision_detection(self):

        #PLAYER LASERS
        if self.player.sprite.lasers:
            for laser in self.player.sprite.lasers:
                if pygame.sprite.spritecollide(laser, self.blocks, True):
                    laser.kill()

                if pygame.sprite.spritecollide(laser, self.aliens, True):
                    laser.kill()

        if self.alien_lasers:
            for laser in self.alien_lasers:
                if pygame.sprite.spritecollide(laser, self.blocks, True):
                    laser.kill()
                if pygame.sprite.spritecollide(laser, self.player, False):
                    laser.kill()
                    self.lives -= 1
                    if self.lives <= 0:
                        time.sleep(1)
                        SCREEN.fill((0,255,0))
                        pygame.display.flip()
                        time.sleep(1)
                        SCREEN.fill((255,255,0))
                        pygame.display.flip()
                        time.sleep(1)
                        SCREEN.fill((255,0,0))
                        pygame.display.flip()
                        time.sleep(1)
                        pygame.quit()
                        sys.exit()

        if self.aliens:
            for alien in self.aliens:
                pygame.sprite.spritecollide(alien, self.blocks, True)

                if pygame.sprite.spritecollide(alien, self.player, False):
                    time.sleep(1)
                    SCREEN.fill((0,255,0))
                    pygame.display.flip()
                    time.sleep(1)
                    SCREEN.fill((255,255,0))
                    pygame.display.flip()
                    time.sleep(1)
                    SCREEN.fill((255,0,0))
                    pygame.display.flip()
                    time.sleep(1)
                    pygame.quit()
                    sys.exit()
```

Εικόνα 44: Μέθοδος ανίχνευσης συγκρούσεων (collision detection).



Εικόνα 45: Στιγμιότυπο εκτέλεσης παιχνιδιού "Space Invaders".

Η κλάση **Player** περιγράφει τις ιδιότητες και τη συμπεριφορά του παίκτη.

```
1  class Player(pygame.sprite.Sprite):
2      def __init__(self, position, constraint, speed):
3          super().__init__()
4          self.image = pygame.image.load('player.png').convert_alpha()
5          self.rect = self.image.get_rect(midbottom = position)
6          self.speed = speed
7          #SCREEN CONSTRAINT FOR OUR PLAYER
8          self.horizontal_constraint = constraint
9          self.ready = True
10         self.laser_time = 0
11         self.laserCD = 600
12
13         self.lasers = pygame.sprite.Group()
```

Εικόνα 46: Δομή κλάσης Player χωρίς μεθόδους.

Ο παίκτης αναπαρίσταται με ένα sprite που φορτώνεται από αρχείο εικόνας και η θέση του καθορίζεται στο κέντρο της οθόνης. Ο παίκτης μπορεί να μετακινείται αριστερά ή δεξιά με τα πλήκτρα του πληκτρολογίου και να πυροβολεί λέιζερ με το πλήκτρο διαστήματος. Η μέθοδος **inputs** χειρίζεται την κίνηση και τις βολές του παίκτη, ενώ η μέθοδος **recharge** παρακολουθεί τον χρόνο που μεσολαβεί μεταξύ των βολών, διασφαλίζοντας ότι υπάρχει ένα χρονικό διάστημα φόρτισης πριν ο παίκτης μπορεί να πυροβολήσει ξανά. Επιπλέον, η

μέθοδος **constraint** περιορίζει την κίνηση του παίκτη εντός των ορίων της οθόνης. Όταν ο παίκτης πυροβολεί, η μέθοδος **laser** δημιουργεί ένα αντικείμενο **Laser** που προστίθεται στην οθόνη και κινείται προς τα επάνω. Η κλάση ολοκληρώνεται με τη μέθοδο **update**, η οποία συντονίζει όλες τις παραπάνω λειτουργίες για την ανανέωση της κατάστασης του παίκτη σε κάθε **frame**.

```
#MOVEMENT
    def inputs(self):
        keys = pygame.key.get_pressed()

        if keys[pygame.K_d]:
            self.rect.x += self.speed
        elif keys[pygame.K_a]:
            self.rect.x -= self.speed

        if keys[pygame.K_SPACE] and self.ready:
            self.laser()
            self.ready = False
            self.laser_time = pygame.time.get_ticks()


    def recharge(self):
        if not self.ready:
            current_time = pygame.time.get_ticks()
            if current_time - self.laser_time >= self.laserCD:
                self.ready = True

    #CHECKS THE RIGHT AND LEFT SIDE OF THE PLAYER AND MAKES SURE YOU CANT GO OUT OF BOUNDS
    def constraint(self):
        if self.rect.left <= 0:
            self.rect.left = 0
        if self.rect.right >= self.horizontal_constraint:
            self.rect.right = self.horizontal_constraint


    def laser(self):
        self.lasers.add(Laser(self.rect.center, -6, self.rect.bottom))


    #UPDATE METHOD FOR OUR VARIOUS OTHER METHODS
    def update(self):
        self.inputs()
        self.constraint()
        self.recharge()
        self.lasers.update()
```

Εικόνα 47: Μεθόδοι κλάσης Player.

Η κλάση **Obstacle** είναι υπεύθυνη για την κατασκευή των εμποδίων στο παιχνίδι. Κάθε εμπόδιο αποτελείται από μία ομάδα από μπλοκ που σχηματίζουν ένα προκαθορισμένο πρότυπο και τοποθετούνται σε συγκεκριμένες θέσεις στην οθόνη. Το χρώμα και το μέγεθος των εμποδίων καθορίζονται κατά την αρχικοποίησή τους. Η συγκεκριμένη κλάση προσφέρει απλώς τη δημιουργία των εμποδίων και δεν εμπλέκεται σε άλλες λειτουργίες πέρα από την αρχική τοποθέτηση.

```python
class Obstacle(pygame.sprite.Sprite):
    def __init__(self, size, color, x, y):
        super().__init__()
        self.image = pygame.Surface((size,size))
        self.image.fill(color)
        self.rect = self.image.get_rect(topleft = (x,y))
```

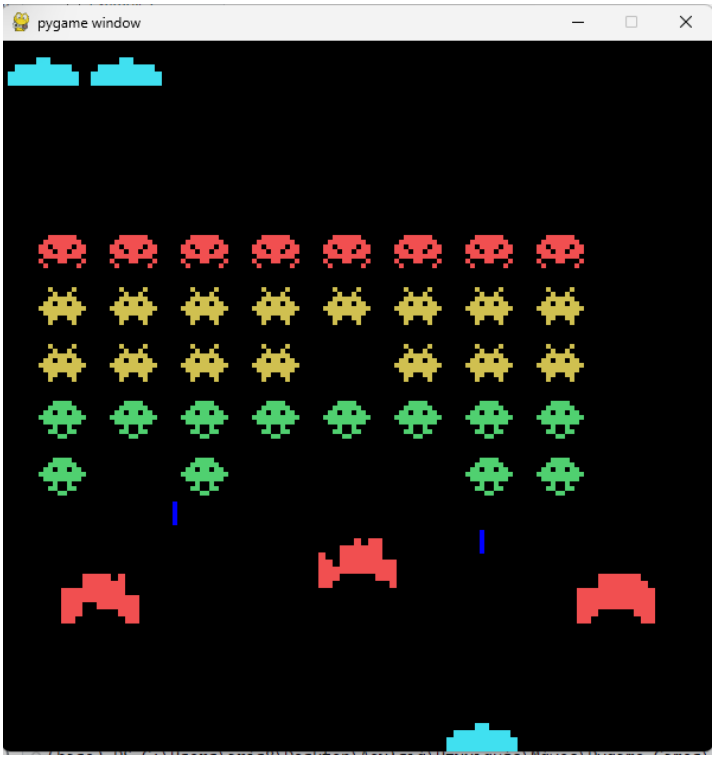
Εικόνα 48: Δομή κλάσης εμποδίου.

Η κλάση **Laser** διαχειρίζεται τα λέιζερ που πυροβολούν τόσο ο παίκτης όσο και οι εξωγήινοι. Κάθε λέιζερ εμφανίζεται ως ένα μπλε ορθογώνιο και κινείται κατακόρυφα στην οθόνη. Η μέθοδος **despawn_laser** ελέγχει αν το λέιζερ έχει εξέλθει από τα όρια της οθόνης, οπότε και αφαιρείται από το παιχνίδι. Η μέθοδος **update** ενημερώνει τη θέση του λέιζερ σε κάθε **frame**, διασφαλίζοντας ότι κινείται και αφαιρείται όταν βγαίνει εκτός οθόνης.

```python
class Laser(pygame.sprite.Sprite):
    def __init__(self, position,speed,height):
        super().__init__()
        self.image = pygame.Surface((4,20))
        self.image.fill('blue')
        self.rect = self.image.get_rect(center = position)
        self.speed = speed


    def despawn_laser(self):
        if self.rect.y <= -50 or self.rect.y >= 650:
            self.kill()


    def update(self):
        self.rect.y += self.speed
        self.despawn_laser()
```

Εικόνα 49: Δομή κλάσης Laser.

Όταν ένα Laser χτυπήσει ένα εμπόδιο, τότε το εμπόδιο αυτό φθείρεται (βλ. Εικόνα 50). Ο παίκτης πρέπει να λαμβάνει υπόψη και την μελλοντική καταστροφή των εμποδίων ώστε να αποφεύγει ενεργά τις επιθέσεις των εχθρών.



Εικόνα 50: Στιγμιότυπο εκτέλεσης παιχνιδιού "Space Invaders" με μερικώς κατεστραμμένα εμπόδια.

Τέλος, η κλάση **Alien** περιγράφει τους εχθρούς του παιχνιδιού, τους εξωγήινους. Κάθε εξωγήινος αναπαρίσταται από ένα sprite που φορτώνεται από αρχείο εικόνας, ανάλογα με το χρώμα που έχει καθοριστεί κατά τη δημιουργία του.

```
1  class Alien(pygame.sprite.Sprite):
2      def __init__(self, color, x, y):
3          super().__init__()
4          file_path = color + '.png'
5          self.image = pygame.image.load(file_path).convert_alpha()
6          self.rect = self.image.get_rect(topleft = (x,y))
7
8      def update(self, speed):
9          self.rect.x += speed
```

Εικόνα 51: Δομή κλάσης Εξωγήινων.

Οι εξωγήινοι κινούνται οριζόντια στην οθόνη και αλλάζουν κατεύθυνση όταν φτάνουν στα όρια της οθόνης. Η μέθοδος **update** διαχειρίζεται την κίνηση των εξωγήινων, η οποία εξαρτάται από την ταχύτητα που καθορίζεται από την κλάση **Game**.

Η συνολική δομή του παιχνιδιού παρέχει μια ολοκληρωμένη και λειτουργική εφαρμογή που επιτρέπει στον χρήστη να ελέγχει ένα διαστημόπλοιο, να πυροβολεί εξωγήινους και να διαχειρίζεται εμπόδια και εχθρικές επιθέσεις.

ΚΕΦΑΛΑΙΟ 5 Συμπεράσματα

Η ενασχόληση με το παρόν θέμα πτυχιακής επέτρεψε την μελέτη των τεχνολογιών υλοποίησης βιντεοπαιχνιδιών, αλλά και την εξοικείωση με τον σχεδιασμό τους. Μέσω της παρούσας πτυχιακής μελετήθηκε ο τομέας των βιντεοπαιχνιδιών, ο αντίκτυπος που έχει στην οικονομία και η συμβολή του σε άλλους επιστημονικούς τομείς, καθώς και οι τεχνολογίες που χρησιμοποιούνται για την ανάπτυξη παιχνιδιών, και πως αυτές διαφέρουν κατά το πέρασμα του χρόνου. Η επιλογή της γλώσσας Python επέτρεψε την μελέτη, τόσο της γλώσσας καθ' αυτώ, αλλά και της χρήσης εξειδικευμένων βιβλιοθηκών για την ανάπτυξη συγκεκριμένου τύπου εφαρμογών.

Από την εκπόνηση της πτυχιακής αντλήθηκαν γνώσεις για τον θεωρητικό σχεδιασμό ενός βιντεοπαιχνιδιού, για την επιλογή των κατάλληλων τεχνολογιών, πόρων και τεχνικών, όπως και για την διαδικασία προγραμματισμού και υλοποίησης του τελικού προϊόντος. Η ανάλυση και περιγραφή της βιβλιοθήκης Pygame επέτρεψε την επί των πραγμάτων ανάλυση ενός σύνθετου πακέτου λογισμικού, των διαδικασιών που αυτό προσφέρει αλλά και τον μεθοδολογιών ενσωμάτωσής του.

Κατά την αξιολόγηση των παιχνιδιών κληθήκαμε να σκεφτούμε αντικειμενικά, να επιλέξουμε τις κατάλληλες μετρικές ποιότητας, ύστερα από ανάγνωση σχετικής βιβλιογραφίας, και τον ποιοτικό σχολιασμό των υλοποιήσεων μας.

Συνολικά, η παρούσα πτυχιακή αποτέλεσε αποσαφήνιση εννοιών που είναι απαραίτητες για την εργασία πάνω στον χώρο της ανάπτυξης παιχνιδιών, παρουσίασε με απλό τρόπο την χρήση του πακέτου Pygame, ενώ παρουσίασε και σχολίασε τα αποτελέσματα του σχεδιασμού και της υλοποίησης.

Βιβλιογραφία

[1] Wikipedia Foundation, Inc., «Spacewar!,» Wikipedia Foundation, Inc., 2024. [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Spacewar!.

[2] «Atari,» Atari Inc., 2024. [Ηλεκτρονικό]. Available: https://atari.com.

[3] Smithsonian, «Video Game History,» Smithsonian, [Ηλεκτρονικό]. Available: https://www.si.edu/spotlight/the-father-of-the-video-game-the-ralph-baer-prototypes-and-electronic-games/video-game-history.

[4] «Nintendo,» Nintendo of America Inc., 2024. [Ηλεκτρονικό]. Available: https://www.nintendo.com/us/.

[5] LG Electronics, «A Brief History of Gaming & The Gaming Industry,» LG Electronics, 2024. [Ηλεκτρονικό]. Available: https://www.lg.com/ca_en/gaming/a-brief-history-of-gaming/.

[6] «Sony,» Sony Europe B.V., 2024. [Ηλεκτρονικό]. Available: https://www.sony.gr.

[7] «Microsoft,» Microsoft, 2024. [Ηλεκτρονικό]. Available: https://www.microsoft.com/el-gr.

[8] «Meta Quest,» Meta, 2024. [Ηλεκτρονικό]. Available: https://www.microsoft.com/el-gr.

[9] D. Hong, A. Christofferson, A. Videbaek, A. Egan, T. Rowland και M. Madden, «Gaming Report 2024, Meet the Moment: How Gamers Are Changing the Game,» Bain & Company, 2024.

[10] «esa: entertainment software association,» Entertainment Software Association, 2024. [Ηλεκτρονικό]. Available: https://www.theesa.com.

[11] «Teconomy,» TEConomy Partners LLC., 2024. [Ηλεκτρονικό]. Available: https://www.teconomypartners.com.

[12] M. Grueber και D. Yetter, «Video Games in the 21st Century: The 2024 Economic Impact Report,» TEConomy Partners, LLC., 2024.

[13] «PS Store,» Sony Interactive Entertainment Europe Ltd., 2024. [Ηλεκτρονικό]. Available: https://store.playstation.com/en-gr/pages/latest.

[14] «Steam,» Valve Corporation, 2024. [Ηλεκτρονικό]. Available: https://store.steampowered.com.

[15] «Epic Store,» Epic Games, Inc., 2024. [Ηλεκτρονικό]. Available: https://store.epicgames.com/en-US/.

[16] «Unity,» Unity Technologies, 2024. [Ηλεκτρονικό]. Available: https://unity.com.

[17] «Java,» Oracle, 2024. [Ηλεκτρονικό]. Available: https://www.java.com/en/.

[18] «Android,» Google LLC., 2024. [Ηλεκτρονικό]. Available: https://www.android.com.

[19] «Python,» Python Software Foundation, 2024. [Ηλεκτρονικό]. Available: https://www.python.org.

[20] «Phaser,» Phaser Studio Inc., 2024. [Ηλεκτρονικό]. Available: https://phaser.io.

[21] «Babylon.js,» Microsoft, 2024. [Ηλεκτρονικό]. Available: https://www.babylonjs.com.

[22] «Rust,» Rust Team, 2024. [Ηλεκτρονικό]. Available: https://www.rust-lang.org.

[23] «The Go Programming Language,» Google LLC, 2024. [Ηλεκτρονικό]. Available: https://go.dev.

[24] «Lua Programming Language,» PUC RIO, 2024. [Ηλεκτρονικό]. Available: https://www.lua.org.

[25] «LibreOffice,» The Document Foundation, 2024. [Ηλεκτρονικό]. Available: https://el.libreoffice.org.

[26] «OpenOffice,» The Apache Software Foundation, 2024. [Ηλεκτρονικό]. Available: https://www.openoffice.org.

[27] «Minetest,» The Minetest Team, 2015-2024. [Ηλεκτρονικό]. Available: https://www.minetest.net.

[28] «Minecraft,» Majong AB. TM Microsoft Corporation, 2024. [Ηλεκτρονικό]. Available: https://www.minecraft.net/en-us.

[29] L. Dell'Osso, B. Nardi, L. Massoni, S. Battaglini, C. De Felice, C. Bonelli, S. Pini, I. M. Cremone και B. Carpita, «Video Gaming in Older People: What Are the Implications for Cognitive Functions?,» *brain sciences,* τόμ. 14, αρ. 7, 2024.

[30] J. Wu, Z. Mao, Z. Ren, W. Zang, H. Tian, L. Huang, H. Liu, F. Liu και L. Peng, «Exploring the impact of computer game playing on cognitive function, Alzheimer's disease risk, and brain-derived neurotrophic factor levels: Basic evidence from Mendelian randomization,» *Digital Health,* τόμ. 10, pp. 1-8, 2024.

[31] «IEU OpenGWAS project,» University of Bristol, 2024. [Ηλεκτρονικό]. Available: https://gwas.mrcieu.ac.uk.

[32] D. S. Shokirovna, «Value of Games in Education,» *Modern Science and Research,* τόμ. 3, αρ. 1, pp. 246-250, 2024.

[33] N. C. Zygouris, K. Botsoglou, A. N. Dadaliaris, G. Dimitriou, D. Trontsios, G. I. Stamoulis και D. Vavougios, «Screening Executive Functions of Preschool Children via a Web Application,» σε *23rd International Conference on Interactive Collaborative Learning*, 2021.

[34] N. C. Zygouris, F. Vlachos, A. N. Dadaliaris, P. Oikonomou, G. I. Stamoulis, D. Vavougios, E. Nerantzaki και A. Striftou, «A Neuropsychological Approach of Developmental Dyscalculia and a Screening Test Via a Web Application,» *iJEP,* τόμ. 7, αρ. 4, 2017.

[35] «Visual Studio Code - Code Editing. Redefined,» Microsoft, 2024. [Ηλεκτρονικό]. Available: https://code.visualstudio.com.

[36] «Notepad++,» Don Ho, 2024. [Ηλεκτρονικό]. Available: https://notepad-plus-plus.org.

[37] «Pycharm,» JetBrains s.r.o., 2024. [Ηλεκτρονικό]. Available: https://www.jetbrains.com/pycharm/.

[38] «Pygame Front Page,» Pygame, 2024. [Ηλεκτρονικό]. Available: https://www.pygame.org/docs/.

[39] Wikipedia Foundation, Inc., «Pygame,» Wikipedia Foundation, Inc., 2024. [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Pygame.

[40] Wikipedia Foundation, Inc., «Python (programming language),» Wikipedia Foundation, Inc., 2024. [Ηλεκτρονικό]. Available: https://en.wikipedia.org/wiki/Python_(programming_language).

[41] A. Sweigart, Making Games with Python & Pygame, Al Sweigart, 2012.

[42] J. Takatalo, J. Häkkinen, J. Kaistinen και G. Nyman, «Measuring User Experience in Digital Gaming: Theoretical and Methodological Issues,» σε *Proceedings of SPIE - The International Society for Optical Engineering*, 2007.

[43] H. Korhonen, M. Montola και J. Arrasvuori, «Understanding Playful User Experience through Digital Games,» σε *International Conference on Designing Pleasurable Products and Interfaces (DPPI09)*, 2009.

[44] R. Bernhaupt, Game User Experience Evaluation, Springer, 2015.

Παράρτημα Ι

Π1. Tower Ascend

```
import pygame
import sys
from pygame.locals import *
import random
import time

pygame.init()
#A TWO DIMENTIONAL VECTOR
vector = pygame.math.Vector2


#SCREEN SETTINGS
WIDTH = 400
HEIGHT = 450

#MOVEMENT RELATED VARIABLES
ACC = 0.5
FRIC = -0.12
FPS = 80


#THE FRAME COUNTER
FramePerSec = pygame.time.Clock()

screen = pygame.display.set_mode((WIDTH, HEIGHT))
pygame.display.set_caption("My First Game")


#THIS IS THE CLASSES SECTION WHERE WE WILL BE SETTING UP THE PLAYER AND
PLATFORM CLASSES FOR OUT PLATFORMER
#THIS PART CREATES THE OBJECTS BUT IT DOESENT YET DRAW THEM ON THE SCREEN

class player(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        #SIZE OF PLAYER
        self.surf = pygame.Surface((30,30))
        self.surf.fill((128,255,40))
        #THE TOP LEFT CORNER OF OUR SCREEN IS THE 0,0 COORDS ACCOUNTING
FOR THE SIZE OF OUR PLAYER SQUARE THESE COORDS PLACE IT ON THE BOTTOM
LEFT CORNER FLAT ON THE PLATFORM
        self.rect = self.surf.get_rect()

        self.pos = vector((15,430))
        self.vel = vector(0,0)
```

```
        self.acc = vector(0,0)
        self.jumping = False
        self.score = 0

    #MOVEMENT FUNCTION
    def movement(self):
        #UNIVERSAL ACCELERATION HORIZONTAL IS 0 AND VERTICAL IS 0.5 FOR
THE EFFECT OF GRAVITY
        self.acc = vector(0,0.5)

        pressed_keys = pygame.key.get_pressed()
        if pressed_keys[K_a]:
            self.acc.x = -ACC
        if pressed_keys[K_d]:
            self.acc.x = ACC

        self.acc.x += self.vel.x * FRIC
        self.vel += self.acc
        self.pos += self.vel + 0.5 * self.acc

        #SCREEN LOOPING
        if self.pos.x > WIDTH:
            self.pos.x = 0
        if self.pos.x < 0:
            self.pos.x = WIDTH

        #THIS LINE UPDATES THE rect() OBJECT OF THE PLAYER WITH THE NEW
POSITION THAT IT HAS GAINED AFTER BEING MOVED
        self.rect.midbottom = self.pos


    #JUMP FUNTION CHECKS TO SEE IF THE PLAYER IS IN CONTACT WITH A
PLATFORM
    #THIS PREVENTS DOUBLE JUMPING
    def jump(self):
        hits = pygame.sprite.spritecollide(self, platforms, False)
        if hits and not self.jumping:
            self.jumping = True
            self.vel.y = -15
    #THE JUMP CANNCEL FUNTION ALLOWS SHORT JUMPS
    def jump_cancel(self):
        if self.jumping:
            if self.vel.y < -3:
                self.vel.y = -3

    #COLLITION DETECTION
    def update(self):
        hits = pygame.sprite.spritecollide(P1, platforms, False)
        if self.vel.y > 0:
```

```
            if hits:
                if self.pos.y < hits[0].rect.bottom:
                    if hits[0].point == True:
                        hits[0].point = False
                        self.score += 1
                    self.pos.y = hits[0].rect.top +1
                    self.vel.y = 0
                    self.jumping = False



class platform(pygame.sprite.Sprite):
    def __init__(self):
        super().__init__()
        self.surf = pygame.Surface((random.randint(50,100), 12))
        self.surf.fill((255,50,0))
        self.rect = self.surf.get_rect(center = (random.randint(0, WIDTH-
10), random.randint(0, HEIGHT-30)))

        self.point = True
        #self.moving = True

    def movement(self):
        pass

def clean_plats(platform, groupies):
    if pygame.sprite.spritecollideany(platform, groupies):
        return True
    else:
        for entity in groupies:
            if entity == platform:
                continue
            if (abs(platform.rect.top - entity.rect.bottom) < 40) and
(abs(platform.rect.bottom - entity.rect.top)< 50):
                return True
        C = False

def rand_plat_gen():
        while len(platforms) < 6:
            width = random.randrange(50,100)
            p = platform()
            C = True

            while C:
                p = platform()
                p.rect.center = (random.randrange(0, WIDTH - width),
random.randrange(-50, 0))
```

```
            C = clean_plats(p, platforms)
        platforms.add(p)
        all_sprites.add(p)



PT1 = platform()
P1 = player()

all_sprites = pygame.sprite.Group()
all_sprites.add(PT1)
all_sprites.add(P1)

PT1.surf = pygame.Surface((WIDTH, 20))
PT1.surf.fill((255,0,0,))
PT1.rect = PT1.surf.get_rect(center = (WIDTH/2, HEIGHT-10))

platforms = pygame.sprite.Group()
platforms.add(PT1)

#PT1.moving = False
PT1.point = False

for x in range(random.randint(4,5)):
    C = True
    pl = platform()
    while C:
        pl = platform()
        C = clean_plats(pl, platforms)
    platforms.add(pl)
    all_sprites.add(pl)

while True:
    P1.update()
    for event in pygame.event.get():
        if event.type == QUIT:
            pygame.quit()
            sys.exit()
        if event.type == pygame.KEYDOWN:
            if event.key == pygame.K_SPACE:
                P1.jump()
        if event.type == pygame.KEYUP:
            if event.key == pygame.K_SPACE:
                P1.jump_cancel()

    if P1.rect.top > HEIGHT:
        for entity in all_sprites:
            entity.kill()
```

```
            time.sleep(1)
            screen.fill((255,5,5))
            pygame.display.update()
            time.sleep(1)
            pygame.quit()
            sys.exit()

    if P1.rect.top <= HEIGHT/3:
        P1.pos.y += abs(P1.vel.y)
        for plat in platforms:
            plat.rect.y += abs(P1.vel.y)
            if plat.rect.top >= HEIGHT:
                plat.kill()

    rand_plat_gen()
    screen.fill((0,0,0,))
    f = pygame.font.SysFont("Verdana", 20)
    g = f.render(str(P1.score), True, (123,255,0))
    screen.blit(g, (WIDTH/2, 10))

    for entity in all_sprites:
        screen.blit(entity.surf, entity.rect)
        entity.movement()

    pygame.display.update()
    FramePerSec.tick(FPS)
```

Π2. Space Invaders

Π2.I space_invaders.py

```
import pygame, sys
from pygame.locals import *
import time
from player import Player
from laser import Laser
import obstacle
from alien import Alien
from random import choice

#THE GAME CLASS CONTAINS ALL OFa THE GAME LOGIC
class Game:
    def __init__(self):
        #PLAYER SETUP
        player_sprite = Player((WIDTH/2, HEIGHT), WIDTH, 5)
        self.player = pygame.sprite.GroupSingle(player_sprite)
```

```
        self.font = pygame.font.SysFont('Verdana',20)
        #OBSTACLE SETUP
        self.shape = obstacle.shape
        self.block_size = 6
        self.blocks = pygame.sprite.Group()
        #CREATING THE OBSTACLES
        self.create_obstacle(50,450)
        self.create_obstacle(485,450)
        self.create_obstacle(267,420)
        #ALIEN SETUP
        self.aliens = pygame.sprite.Group()
        self.alien_lasers = pygame.sprite.Group()
        self.alien_setup(rows = 5, columns = 8) #5,8
        self.alien_speed = 1
        #HEALTH SYSTEM
        self.lives = 3
        self.lives_surf = pygame.image.load('player.png').convert_alpha()
        self.lives_position = WIDTH - (self.lives_surf.get_size()[0] * 2
+ 475)

    def create_obstacle(self, x_pos, y_pos):
        for row_pointer, row in enumerate(self.shape):
            for column_pointer, column in enumerate(row):
                if column == 'x':
                    x = x_pos + column_pointer * self.block_size
                    y = y_pos + row_pointer * self.block_size
                    block = obstacle.Obstacle(self.block_size,
(241,79,80), x, y)
                    self.blocks.add(block)

    def alien_setup(self, rows, columns, x_dis = 60, y_dis = 48, x_offset
= 70, y_offset = 100):
        for row_pointer, row in enumerate(range(rows)):
            for column_pointer, column in enumerate(range(columns)):
                x = column_pointer * x_dis + x_offset
                y = row_pointer * y_dis + y_offset

                if row_pointer == 0: alien_sprite = Alien('red',x,y)
                elif 1 <= row_pointer <=2: alien_sprite =
Alien('yellow',x,y)
                else: alien_sprite = Alien('green',x,y)

                self.aliens.add(alien_sprite)

    def alien_constriction(self):
        all_aliens = self.aliens.sprites()
        for alien in all_aliens:
```

```
            if alien.rect.right >= WIDTH:
                self.alien_speed = -1
                self.alien_move_down(1)
            elif alien.rect.left <= 0:
                self.alien_speed = 1
                self.alien_move_down(1)

    def alien_move_down(self, distanse):
        if self.aliens:
            for alien in self.aliens.sprites():
                alien.rect.y += distanse

    def alien_shoot(self):
        if self.aliens.sprites():
            random_alien = choice(self.aliens.sprites())
            alien_laser = Laser(random_alien.rect.center, 7, HEIGHT)
            self.alien_lasers.add(alien_laser)

    def collision_detection(self):

        #PLAYER LASERS
        if self.player.sprite.lasers:
            for laser in self.player.sprite.lasers:
                if pygame.sprite.spritecollide(laser, self.blocks, True):
                    laser.kill()

                if pygame.sprite.spritecollide(laser, self.aliens, True):
                    laser.kill()

        if self.alien_lasers:
            for laser in self.alien_lasers:
                if pygame.sprite.spritecollide(laser, self.blocks, True):
                    laser.kill()
                if pygame.sprite.spritecollide(laser, self.player,
False):
                    laser.kill()
                    self.lives -= 1
                    if self.lives <= 0:
                        time.sleep(1)
                        SCREEN.fill((0,255,0))
                        pygame.display.flip()
                        time.sleep(1)
                        SCREEN.fill((255,255,0))
                        pygame.display.flip()
                        time.sleep(1)
                        SCREEN.fill((255,0,0))
                        pygame.display.flip()
                        time.sleep(1)
```

```
                        pygame.quit()
                        sys.exit()

        if self.aliens:
            for alien in self.aliens:
                pygame.sprite.spritecollide(alien, self.blocks, True)

                if pygame.sprite.spritecollide(alien, self.player,
False):
                    time.sleep(1)
                    SCREEN.fill((0,255,0))
                    pygame.display.flip()
                    time.sleep(1)
                    SCREEN.fill((255,255,0))
                    pygame.display.flip()
                    time.sleep(1)
                    SCREEN.fill((255,0,0))
                    pygame.display.flip()
                    time.sleep(1)
                    pygame.quit()
                    sys.exit()

    def lives_display(self):
        for life in range(self.lives - 1):
            x = self.lives_position + (life *
(self.lives_surf.get_size()[0] + 10))
            SCREEN.blit(self.lives_surf, (x,8))

    def victory_screen(self):
        if not self.aliens.sprites():
            victory_message = self.font.render('Victory', False, 'white')
            victory_rect = victory_message.get_rect(center = (WIDTH/2,
HEIGHT/2))
            SCREEN.blit(victory_message, victory_rect)
            pygame.display.flip()
            pygame.time.wait(3000)
            pygame.quit()
            sys.exit()


    def run(self):
        self.player.update()
        self.aliens.update(self.alien_speed)
        self.alien_constriction()
        self.alien_lasers.update()
        self.collision_detection()
        self.lives_display()
        self.victory_screen()
```

```
        #DRAW FUNCTION DRAWS THE SPRITES ON THE SCREEN
        self.player.sprite.lasers.draw(SCREEN)
        self.player.draw(SCREEN)

        self.blocks.draw(SCREEN)
        self.aliens.draw(SCREEN)
        self.alien_lasers.draw(SCREEN)
        #UPDATE ALL SPRITE GROUPS AND DRAW ALL SPRITE GROUPS


if __name__ == '__main__':
    pygame.init()
    WIDTH = 600
    HEIGHT = 600
    SCREEN = pygame.display.set_mode((WIDTH, HEIGHT))
    #THIS CREATES AND INSTANCE OF OUR GAME CLASS
    game = Game()
    #FRAME COUNTER
    FramesPerSec = pygame.time.Clock()
    #ALIEN LASER TIMER
    alien_laser_cooldown = pygame.USEREVENT + 1
    pygame.time.set_timer(alien_laser_cooldown, 800)

    #GAME LOOP
    while True:
        for event in pygame.event.get():
            if event.type == QUIT:
                pygame.quit()
                sys.exit()
            if event.type == alien_laser_cooldown:
                #THIS METHOD RUNS HERE SO IT ADHERES TO THE TIMER WE SET
FOR IT
                game.alien_shoot()

        SCREEN.fill((0,0,0))
        #THIS ALLOWS TO WRITE ALL OF THE NECESSARY CODE IN THE GAME CLASS
AND JUST EXECUTE IT HERE
        game.run()

        pygame.display.flip()
        FramesPerSec.tick(60)
```

Π2.ΙΙ player.py

```
import pygame
from laser import Laser
```

```
class Player(pygame.sprite.Sprite):
    def __init__(self, position, constraint, speed):
        super().__init__()
        self.image = pygame.image.load('player.png').convert_alpha()
        self.rect = self.image.get_rect(midbottom = position)
        self.speed = speed
        #SCREEN CONSTRAINT FOR OUR PLAYER
        self.horizontal_constraint = constraint
        self.ready = True
        self.laser_time = 0
        self.laserCD = 600

        self.lasers = pygame.sprite.Group()

    #MOVEMENT
    def inputs(self):
        keys = pygame.key.get_pressed()

        if keys[pygame.K_d]:
            self.rect.x += self.speed
        elif keys[pygame.K_a]:
            self.rect.x -= self.speed

        if keys[pygame.K_SPACE] and self.ready:
            self.laser()
            self.ready = False
            self.laser_time = pygame.time.get_ticks()


    def recharge(self):
        if not self.ready:
            current_time = pygame.time.get_ticks()
            if current_time - self.laser_time >= self.laserCD:
                self.ready = True

    #CHECKS THE RIGHT AND LEFT SIDE OF THE PLAYER AND MAKES SURE YOU CANT
GO OUT OF BOUNDS
    def constraint(self):
        if self.rect.left <= 0:
            self.rect.left = 0
        if self.rect.right >= self.horizontal_constraint:
            self.rect.right = self.horizontal_constraint


    def laser(self):
        self.lasers.add(Laser(self.rect.center, -6, self.rect.bottom))
```

```
    #UPDATE METHOD FOR OUR VARIOUS OTHER METHODS
    def update(self):
        self.inputs()
        self.constraint()
        self.recharge()
        self.lasers.update()
```

Π2.III obstacle.py

```
import pygame

class Obstacle(pygame.sprite.Sprite):
    def __init__(self, size, color, x, y):
        super().__init__()
        self.image = pygame.Surface((size,size))
        self.image.fill(color)
        self.rect = self.image.get_rect(topleft = (x,y))

shape = [
'  xxxxxx',
' xxxxxxxx',
'xxxxxxxxxx',
'xxxxxxxxxx',
'xxxxxxxxxx',
'xxx     xxx',
'xx       xx']
```

Π2.IV laser.py

```
import pygame

class Laser(pygame.sprite.Sprite):
    def __init__(self, position,speed,height):
        super().__init__()
        self.image = pygame.Surface((4,20))
        self.image.fill('blue')
        self.rect = self.image.get_rect(center = position)
        self.speed = speed


    def despawn_laser(self):
        if self.rect.y <= -50 or self.rect.y >= 650:
            self.kill()


    def update(self):
        self.rect.y += self.speed
        self.despawn_laser()
```

Π2.V Alien.py

```
import pygame

class Alien(pygame.sprite.Sprite):
    def __init__(self, color, x, y):
        super().__init__()
        file_path = color + '.png'
        self.image = pygame.image.load(file_path).convert_alpha()
        self.rect = self.image.get_rect(topleft = (x,y))

    def update(self, speed):
        self.rect.x += speed
```