Τρίτη προγραμματιστική άσκηση OpenGL στο μάθημα “ΜΥΥ-702 ΓΡΑΦΙΚΑ ΥΠΟΛΟΓΙΣΤΩΝ ΚΑΙ ΣΥΣΤΗΜΑΤΑ ΑΛΛΗΛΕΠΙΔΡΑΣΗΣ”

ΑΘΑΝΑΣΌΠΟΥΛΟΣ ΑΛΈΞΑΝΔΡΟΣ ΑΜ: 4020

ΜΠΟΥΛΏΤΗΣ ΠΑΝΑΓΙΏΤΗΣ ΑΜ: 4271

Η τρίτη προγραμματιστική άσκηση χτίστηκε πάνω στον έτοιμο κώδικα που μας δόθηκε για το παράδειγμα προγράμματος με φόρτωση μοντέλων .obj και υφές, με τους αντίστοιχους shaders. Συνιστάται να τρέξει το πρόγραμμα σε λογισμικό Windows λόγω της βιβλιοθήκης <Windows.h>.

Χρησιμοποιώντας τον κώδικα της δεύτερης άσκησης, το παράθυρο έχει διαστάσεις 800x800 και με τον τίτλο “Ηλιακό Σύστημα” με την χρήση του u8 στην εντολή `window = glfwCreateWindow()`. Στο παρασκήνιο, δόθηκε το μαύρο χρώμα, με την χρήση της εντολής `glClearColor()` με τιμές (0,0,0). Επίσης γίνεται ο ίδιος έλεγχος για το κλείσιμο του παραθύρου μέσω της Boolean μεταβλητής `Exit`, η οποία αρχικοποιείται στην τιμή false. Λαμβάνοντας από τον χρήστη, το πάτημα του κεφαλαίου πλήκτρου **“Q”** (με την χρήση είτε του **“Caps Lock”** είτε του **“Shift”**), η Exit γίνεται True, άρα τερματίζει η do while που είναι υπεύθυνη για το σχηματισμό του παραθύρου.

Για την τροποποιημένη κάμερα:

Στη προηγούμενη άσκηση, δεν είχε δοθεί περιστροφική κίνηση που χρειαζόταν για να κατευθυνόμαστε πιο εύκολα στη σκηνή, οπότε έπρεπε να υπάρξουν τροποποιήσεις. Οι κινήσεις στους άξονες και προς το κέντρο του Ήλιου γίνονται με τα ίδια πλήκτρα και ο υπολογισμός τους πραγματοποιείται στο controls.cpp. Πιο συγκεκριμένα:

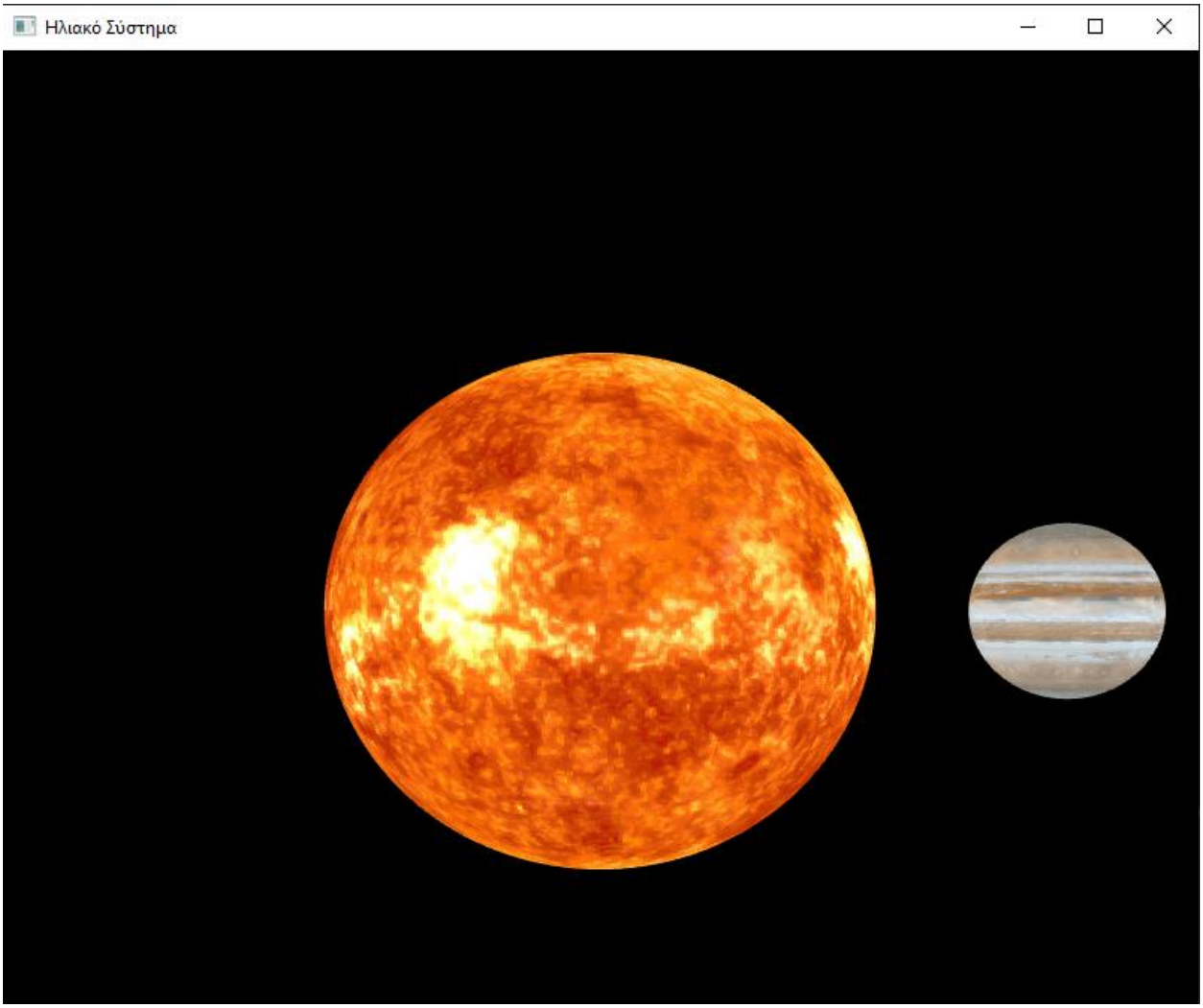
- Για την κίνηση προς το κέντρο του Ήλιου: Ο τύπος του υπολογισμού της απλοποιήθηκε στον τύπο : `(center - camera_position)`
  για να κάνουμε zoom in και `-(center - camera_position)` για το zoom out, όπου camera_position το διάνυσμα με την τοποθεσία της κάμερας.
  Ομοίως με πριν, οι κινήσεις γίνονται με τα πλήκτρα **"="**,**"-"** και των **"+"** και **"-"** του πληκτρολογίου num.
- Και για τον άξονα x και τον y γίνεται χρήση του διανύσματος `direction` , το οποίο είναι η κανονικοποιημένη μορφή του `(center - camera_position)`.

- Για την κίνηση στον άξονα x: Χρησιμοποιείται το διάνυσμα `right` , το οποίο είναι η κανονικοποιημένη μορφή του εξωτερικού γινομένου διανυσμάτων `up_angle` και `direction` (όπου `up_angle` το διάνυσμα με συντεταγμένες (0,1,0)). Με την πρόσθεση και αφαίρεση του `camera_position` με το `right` (πλήκτρα **"A"** και **"D"** για κίνηση προς τα αριστερά και προς τα δεξιά αντίστοιχα) παρατηρούμε πως η κάμερα περιστρέφεται γύρω από την σκηνή, καθώς έχει ως δεύτερο όρισμα το `center` στην συνάρτηση `glm::lookAt()`.

- Για την κίνηση στον άξονα y: Ομοίως με πριν, γίνεται χρήση του διανύσματος `up`, το οποίο είναι η κανονικοποιημένη μορφή του εξωτερικού γινομένου διανυσμάτων `right` και `direction`. Τα πλήκτρα **"W"** και **"X"** προσδίδουν την κίνηση προς τα πάνω και προς τα κάτω αντίστοιχα.

Σημειώσεις: Εφόσον ο Ήλιος έχει μια αρκετά μεγάλη ακτίνα, η κάμερα αρχικοποιείται στην θέση (0,0,80) ώστε να φαίνεται στο σκηνικό και ο Ήλιος και ο πλανήτης. Επίσης το ratio στην `glm::perspective()` έχει αλλάξει σε 1:1, καθώς με το 4:3 όλοι οι πλανήτες παρουσιαζόντουσαν σε αυγοειδή μορφή.

Παρακάτω υπάρχει ένα στιγμιότυπο με την αρχική παρουσίαση του σκηνικού:

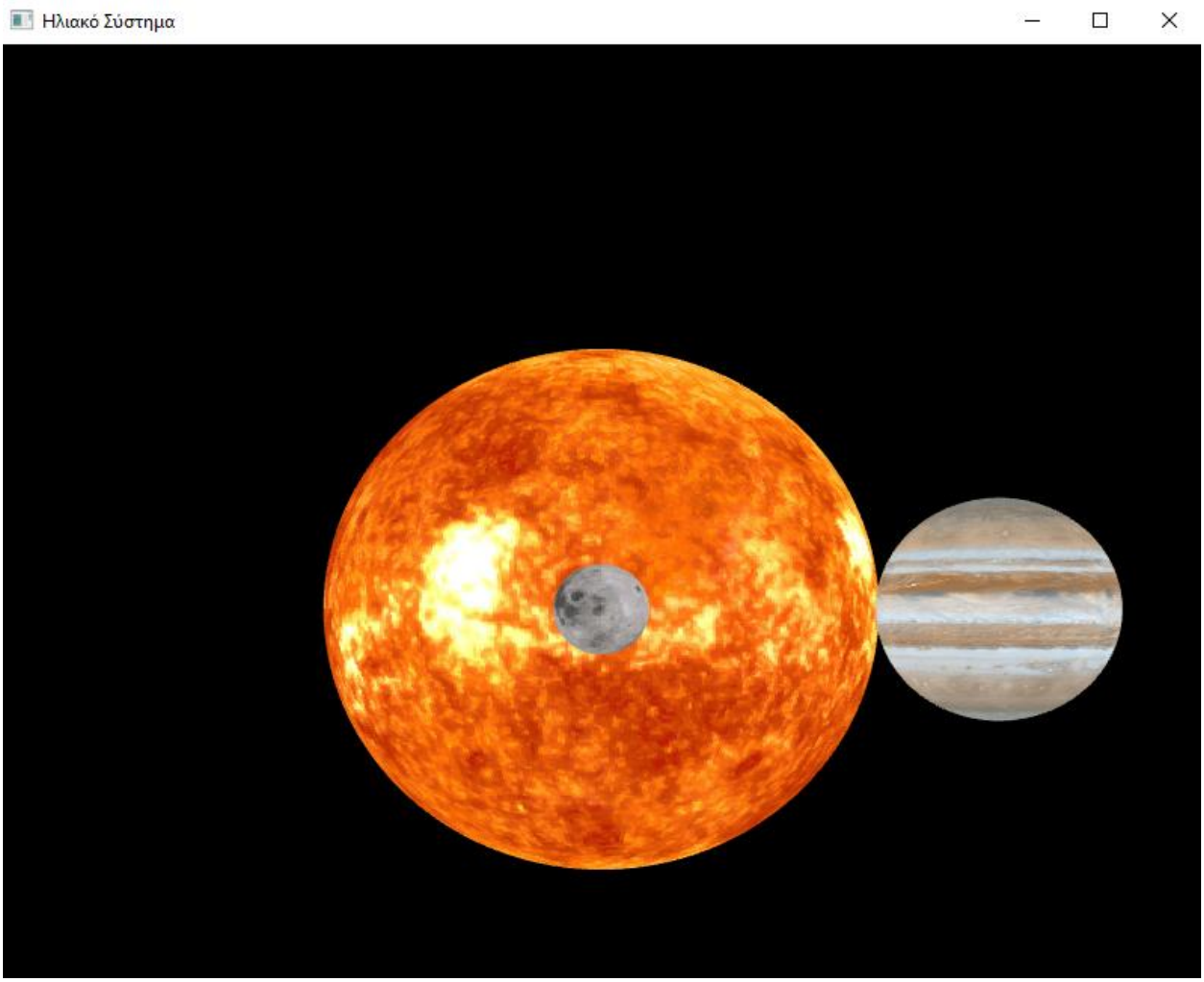


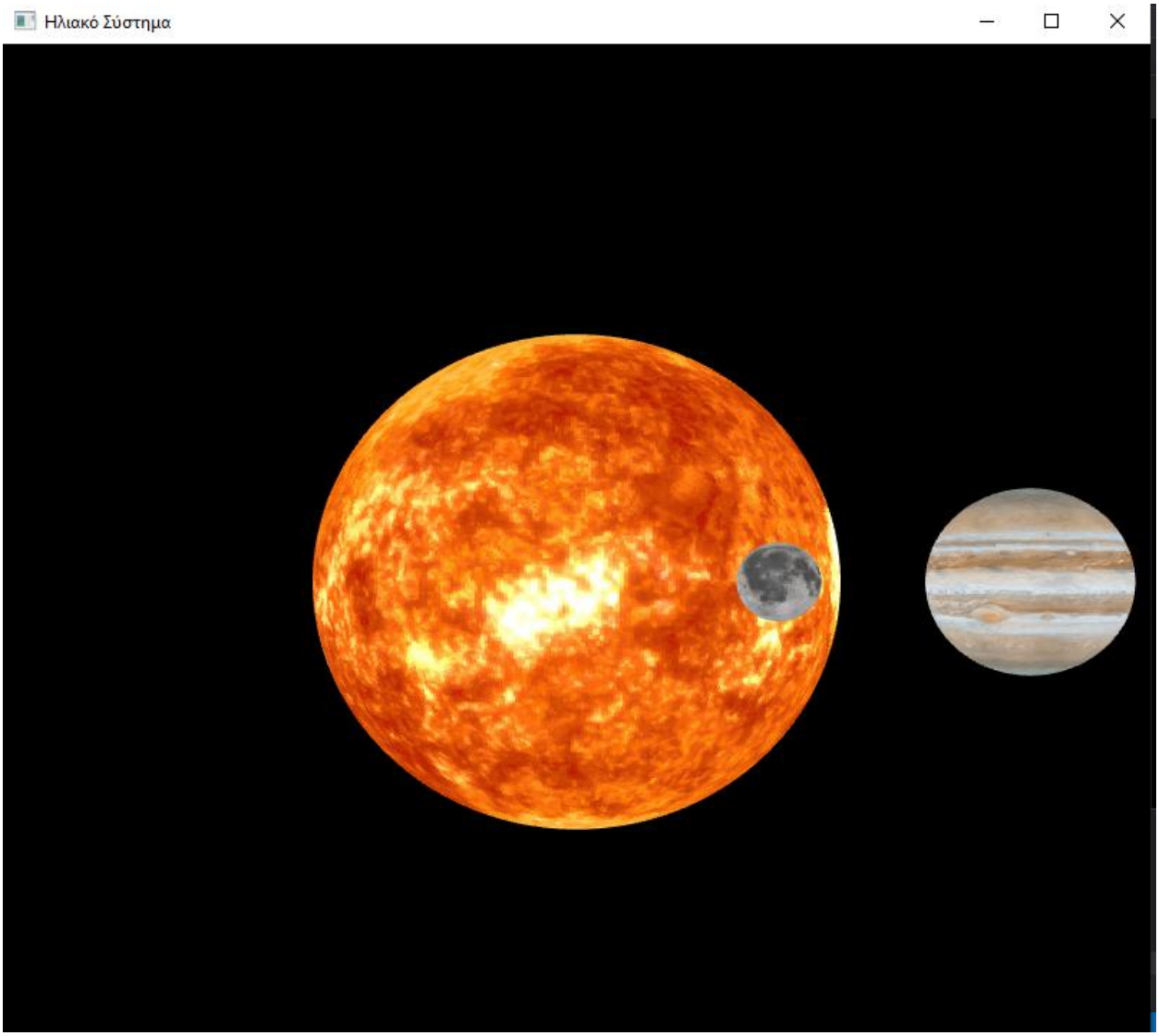
Για το φόρτωμα και την δημιουργία των αντικειμένων:

Για τη δημιουργία των αντικειμένων στηριχτήκαμε στο παράδειγμα που μας δόθηκε, αλλάζοντας τις αντίστοιχες ονομασίες των αντικειμένων και υφών.
Πιο συγκεκριμένα, για να φορτώσουμε την υφή του ηλίου αλλάξαμε το πρώτο όρισμα του stbi_load στο `data`, δηλαδή: `data = stbi_load ("sun.jpg", &width, &height, &nrChannels, 0);`
Στη συνέχεια, για να φορτώσουμε το αντικείμενο του ηλίου αλλάξαμε ξανά το πρώτο όρισμα στο loadOBJ του res. Συγκεκριμένα, η εντολή τροποποιήθηκε ως εξής: `res = loadOBJ("sun.obj", vertices, uvs, normals);`
Προκειμένου να φτιάξουμε τον μετεωρίτη, ακολουθήσαμε όμοια διαδικασία με τον ήλιο. Οι εντολές παρέμειναν ίδιες, ωστόσο αλλάξαμε τις περισσότερες

παραμέτρους, όπως: width, height nrChannels κλπ.
Περίπου τα ίδια ακολουθήσαμε και για τον πλανήτη και τον μετεωρίτη, κρατώντας τις μεθόδους κι αλλάζοντας τις παραμέτρους, προσθέτοντας ωστόσο κάποιες επιπλέον εντολές ώστε να υλοποιηθούν πλήρως τα υποερωτήματα.
Και στις δύο περιπτώσεις κύριες αλλαγές αποτελούσαν τα ορίσματα στο `data` και res, ώστε να μη φορτώνουν τα ίδια αρχεία.
Στον πλανήτη, χρησιμοποιήθηκε και η εντολή translate, ώστε να αλλάξουμε το κέντρο του. Η εντολή υλοποιήθηκε ως εξής: glm::mat4 ModelMatrix_planet = glm::translate(ModelMatrix, planet_position); , με planet_position = (25,0,0) αρχικά, επειδή θέλαμε το κέντρο του να βρίσκεται στο σημείο αυτό.

Παρακάτω είναι στιγμιότυπα του σχήματος με όλα τα στοιχεία στην σκηνή:





Για την κίνηση του πλανήτη και του μετεωρίτη:

Για τις κινήσεις αυτές χρησιμοποιούμε παρόμοια λογική με την κίνηση της κάμερας, μόνο που αυτή τη φορά στον άξονα x κινείται ο πλανήτης, δηλαδή γύρω από τον ήλιο και η κίνηση του μετεωρίτη είναι σαν να έχουμε συνεχώς πατημένο το πλήκτρο "+" και να κάνουμε zoom in. Επομένως:

- Έχουμε τις μεταβλητές `planet_position` και `meteor_position` για τις συντεταγμένες του πλανήτη και μετεωρίτη. Είναι αντίστοιχες του camera_position.
- Για να κινείται συνεχώς ο πλανήτης, προσθέσαμε στον κώδικα τη γραμμή `planet_position += right * deltaTime * planet_speed;` όπου πετυχαίνει να αλλάζει συνεχώς τη θέση του χωρίς κάποια αλληλεπίδραση του χρήστη(λόγω του deltaTime που μεταβάλλεται με το πέρασμα του χρόνου).

- Το planet_speed αρχικοποιείται στην τιμή 3, αλλά θα αυξομειώνεται από τον χρήστη, με τα πλήκτρα **“u”** και **“p”.** Με το **“u”** αυξάνουμε την ταχύτητα κατά 0.05 , ενώ με το **“p”** το πολλαπλασιάζουμε με το 0.999 ώστε η ταχύτητα να μειωθεί, αλλά να μην γίνει αρνητική, καθώς έτσι θα άλλαζε κατεύθυνση.

- Ομοίως για την συνεχή κίνηση του μετεωρίτη προσθέσαμε τη γραμμή `meteor_position += (center - meteor_position) * deltaTime * 0.3f;` (όπου το `center` έχει συντεταγμένες (0,0,0) δηλαδή είναι το κέντρο του Ήλιου) εφόσον φυσικά ο χρήστης πατήσει το πλήκτρο **“Space”**.

Οι κινήσεις αυτές πραγματοποιούνται καθώς το ModelMatrix του κάθε αντικειμένου κάνει μετατόπιση στην θέση που του δίνουμε με την `glm::translate(ModelMatrix, <position>);`

Για την αλληλεπίδραση του μετεωρίτη με τα αντικείμενα της σκηνής:

Αρχικά για να εκτοξεύσουμε τον μετεωρίτη, χρειάζεται να γνωρίζουμε την θέση του παρατηρητή, δηλαδή την τοποθεσία της κάμερας στη σκηνή. Η πληροφορία αυτή βρίσκεται στο διάνυσμα `camera_position` στο πρόγραμμα controls.cpp. Επομένως, τροποποιώντας και το controls.hpp, δημιουργήθηκε η συνάρτηση `glm::vec3 getCameraPosition()` η οποία επιστρέφει το `camera_position` , για να μπορέσουμε να το χρησιμοποιήσουμε στην Main.cpp.

Επίσης, προκειμένου να ξέρουμε την απόσταση του μετεωρίτη με τα άλλα δυο αντικείμενα, δημιουργήθηκε η συνάρτηση `compute_distance()` στην Main.cpp, η οποία υπολογίζει την απόσταση δύο αντικειμένων στον τρισδιάστατο χώρο, μέσω του τύπου $d = \sqrt{(x_2 - x_1)^2 + (y_2 - y_1)^2 + (z_2 - z_1)^2}$.

Ο πλανήτης και ο μετεωρίτης έχουν ο καθένας από μια boolean μεταβλητή (Planet_flag, Meteor_flag) που μας ενημερώνει για το αν το αντικείμενο πρέπει να ζωγραφίζεται στην σκηνή. Αν οι μεταβλητές αυτές είναι true, ικανοποιούν αντίστοιχες συνθήκες if στο do while της main και πραγματοποιούνται τα σωστά `glDrawArrays()` ,αλλιώς δεν εμφανίζονται στη σκηνή.
Αυτό φυσικά είναι για τον χειρισμό δημιουργίας ή καταστροφής των αντικειμένων, δηλαδή για το αν εκτοξεύουμε τον μετεωρίτη και αν αυτός χτυπήσει στον Ήλιο και καταστραφεί ή αν χτυπήσει τον πλανήτη και καταστραφεί μαζί του.

Συνδιάζοντας όλα τα παραπάνω:

- Το Planet_flag αρχικοποιείται στην τιμή true, καθώς θέλουμε να ζωγραφίζεται από την έναρξη της εκτέλεσης, ενώ το Meteor_flag αρχικοποιείται στην τιμή false, καθώς δεν το έχουμε εκτοξεύσει ακόμα.
- Αν ο χρήστης πατήσει το πλήκτρο **"Space"** εκτοξεύεται ο μετεωρίτης, δηλαδή το Meteor_flag παίρνει την τιμή true, και το `meteor_position` την θέση της κάμερας ώστε να τροποποιηθεί σωστά το `glm::translate()` για το `ModelMatrix_meteor` και δλδ την θέση του μετεωρίτη.
- Αν ο μετεωρίτης χτυπήσει τον Ήλιο, καταστρέφεται, δηλαδή αν η απόσταση του μετεωρίτη με τον Ήλιο είναι μικρότερη από 15 (η ακτίνα του Ήλιου) τότε το Meteor_flag γίνεται ξανά false και δεν ζωγραφίζεται πλέον.
- Ομοίως αν η απόσταση του μετεωρίτη και του πλανήτη είναι μικρότερη από 5 (ακτίνα του πλανήτη) και τα δύο flags γίνονται false και δεν ζωγραφίζονται άλλο.

<u>Σημείωση</u>: Πέρα από την υλοποίηση του bonus κομματιού της άσκησης για την ταχύτητα περιστροφής του πλανήτη, προσθέσαμε στον κώδικα μια επιπρόσθετη λειτουργία.
Κατά τη διάρκεια του ελέγχου, παρατηρήσαμε πως όταν καταστρεφόταν ο πλανήτης ο μόνος τρόπος για να ελέγξουμε πως έγινε σωστά ήταν να τρέξουμε ξανά το πρόγραμμα, κάτι σχετικά χρονοβόρο.
Επομένως, προσθέσαμε την ιδιότητα ο χρήστης με το πλήκτρο **"R"** να ξαναεμφανίζει τον πλανήτη στην αρχική του θέση με την αρχική του ταχύτητα, να κάνει ένα "reset" στο πρόγραμμα για να επαναλάβουμε το πείραμα όσες φορές θέλουμε χωρίς να χρειαστεί να το ξανατρέξουμε.