# Introducere

Imaginaţi-vă că faceţi animaţie pentru un personaj în Maya. În timp ce creaţi animaţia pentru acest personaj, veţi începe să repetaţi următorii paşi:

* Referinţele personajului.
* Importarea unui motion capture animation.
* Setarea frame range.
* Referinţele pentru fundalul scenei.
* Configurările camerelor.

Dacă se lucrează pe o producţie mai mare, între 10 şi 50 animatori, aceşti simpli paşi pot cauza mici probleme. Dacă întrerupeţi acest proces, vă daţi seama că sunt multe locuri în care aţi putea să folosiţi tool-uri cum ar fi:

* În primul rând, animatorii trebuie să caute calea corectă către scenă respectivă. Această cale se poate schimba în orice moment, iar animatorii pot deschide un alt fişier. Un simplu tool cu o listă de personaje poate face acesta sarcină simplă şi rezonabilă.
* Mai departe, îţi doreşti un tool care să organizeze şi să administreze importarea lui "motion capture date" corect pe controlul unui personaj. Acest tool poate de asemenea să seteze frame range pentru animaţie şi să mute camera într-o locaţie corectă în timpul procesului.
* Ultimul pas reprezintă referinţele scenei de fundal, pot dura cel puţin un minut pentru fiecare animator, căutând manual pentru fişierele corecte. Se poate crea un alt tool simplu care să afişeze o listă cu toate fundalurile pentru a putea alege unul.

Vă veţi da seama că acest gen de tool-uri vor economisi timp, selectarea fişierului va deveni mult mai precisă şi va permite animatorilor să se concentreze pe sarcinile lor. Cel mai bun mijloc pentru creaţia acestui tip de tool este folosirea unui limbaj de programare încorporat în programul Maya, mai exact Python.

Python este un limbaj de scripting ce a fost iniţial dezvoltat înafara programului Maya, deci el oferă un set puternic de funcţionalităţi şi o uriaşă bază de useri. În Maya 8.5, Autodesk a adăugat oficial suport pentru Python scripting. Includerea acestui limbaj de scripting a dus la dezvoltarea interfeţelor de programare din Maya. În timp ce Maya Embedded Language (MEL) este folosit pentru scripting în Maya de mulţi ani, întrebarea este de ce s-a ales şi implementarea limbajului de scripting Python. O perspectivă mai largă ne dezvăluie cu uşurinţă foarte multe avantaje esenţiale:

* Comunitatea: MEL are o bază de utilizatori foarte mică în comparaţie cu Python, deoarece doar Maya dezvoltă limbajul de scripting MEL. Python este folosit de toţi oamenii care se implică în dezvoltarea software cu multe tipuri de aplicaţii.
* Putere: Python este un limbaj de scripting mult mai avansat şi îţi permite să realizezi lucruri care nu sunt posibile în MEL. Python este un limbaj de programare complet orientat pe obiecte şi are abilitatea de a comunica foarte uşor cu amândouă: Maya Command Engine şi C++ API, permiţându-ţi să scrii scripturi şi plugin-uri cu un singur limbaj de programare. Chiar dacă scrii plugin-uri în C++, Python îţi permite să testezi cod API interactiv în Maya Script Editor.
* Cross-platform: Python se poate executa de pe orice sistem de operare, orice arhitectură a procesorului sau versiune de software. De fapt, scripturile create în Python nu trebuie să fie compilate deloc!
* Industria standard: Din cauza avantajelor pe care le oferă Python, acesta este foarte uşor de integrat în multe alte aplicaţii importante. Librăriile pot fi împărţite uşor între Maya şi alte aplicaţii în departamentul de pipeline, precum şi MotionBluider.

# Capitolul 1. Ce înseamnă CGI?

Imaginează-ţi imposibilul! Când oamenii se gândesc la CGI (imaginar, imagistică generat sau generată de calculator), ei se gândesc la efectele speciale din filme. Dar CGI poate fi de asemenea folosit pentru a vizualiza concepte tehnice care ar fi dificil de ilustrat prin alte moduri. CGI este de asemenea un mod de a crea fotografii realiste de o calitate foarte înaltă.

Ce înseamnă Computer Generated Imagery? Gândeşte-te în felul următor: CGI încearcă să creeze imagini virtuale folosind numere, aşa cum foloseşte o cameră lumină. În proiectarea acestor medii virtuale, noi creăm şi poziţionăm modele, camere şi lumini, aşa cum am face-o într-un studio real (chiar şi când discutăm despre setări ca “depth of field” sau “light fall-off”). Dar, se pot crea şi efecte stranii care sunt imposibil de realizat în realitate. Lumea noastră nu depinde de graniţe precum gravitaţia sau alte constrângeri fizice. Cu aceste posibilităţi nelimitate, imaginaţia şi arta devin foarte importante.

## Computer-generated imagery (CGI) [7][8][9]

CGI reprezintă punerea în aplicaţie a domeniului de grafică pe calculator, în mod special grafică 3D, pentru efecte speciale. CGI este folosit în filme, programe tv, reclame, simulări şi mass-media. Grafică pe calculatoare în timp real, precum şi jocurile video, sunt rareori menţionate că şi CGI. Scenele vizuale pot fi dinamice sau statice, şi pot fi bidimensionale (2D), deşi termenul de CGI este cel mai frecvent folosit pentru a desemna grafică pe calculator tridimensională (3D). Termenul de animaţie pe calculator se referă la "randarea" dinamică a imaginilor generate de calculator la fel ca şi un film. Softurile de grafică pe calculator sunt folosite pentru a realiza CGI pentru filme, desene animate, etc. Disponibilitatea utilizatorilor pentru softurile CGI şi "upgradarea" calculatorului pentru a rula cu o viteză mai mare a permis artiştilor "freelancer" şi companiilor mici să producă filme, jocuri şi arta de calitate profesională folosind calculatoarele personale. În Figura 1.1 este prezentată o maşină într-un deşert, maşina şi deşertul fiind create complet pe calculator de către un artist "freelancer" folosind doar un calculator, câteva softuri de grafică şi puţină inspiraţie. Acest lucru a dus la creaţia unei subculturi bazată pe Internet care are propriile seturi de celebrităţi la nivel mondial, clişee şi un vocabular tehnic foarte amplu. Evoluţia CGI a dus la apariţia cinematografiei virtuale în anii 90 în care camerele de filmat nu mai sunt fizice, ci virtuale.



**Figura.1.1 - Exemplu de imagine generată pe calculator**

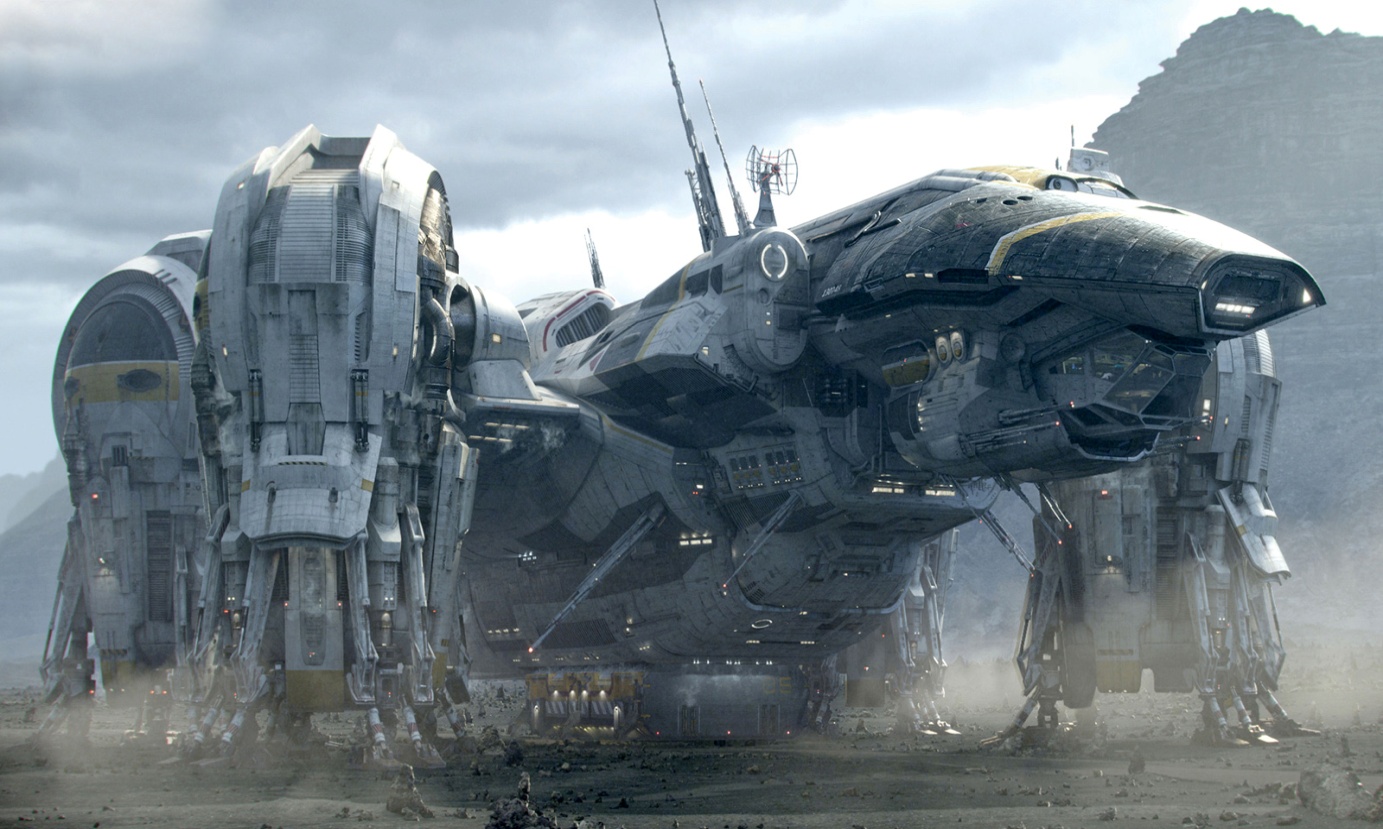
*(Sursa:* www.bedistinct.co.uk/gallery/mars/MarsAngle1Final2.jpg)

CGI este mult mai des folosit deoarece imaginile produse pe calculator în spaţiul virtual sunt mult mai ieftine din punct de vedere financiar decât metodele fizice, cum este şi realizarea unor super detalii pentru efectele din filme şi fotografii sau pentru angajarea unui număr mare de oameni pentru a realiza scene aglomerate. În Figura 1.2 este prezentată o scenă din filmul 300 2006 unde majoritatea oamenilor au fost creaţi 3D. Pentru a crea această scenă s-au folosit doar patru oameni cu scopul de a le copia mişcările corpului. Iar după aceste mişcări s-au animat oamenii creaţi 3D. CGI permite crearea unor imagini ce nu pot fi realizabile folosind orice altă metodă şi permite unui singur artist să producă o imagine sau un film fără a se folosi de actori sau de alţi contribuitori la proiect. În Figura 1.3 este prezentată o navă spaţială creată şi animată 3D din filmul Prometheus 2012. Acesta navă a fost creată la un nivel foarte înalt de detaliu şi nu poate fi realizată în alte medii, ci doar în lumea virtuală folosind CGI.



**Figura.1.2 - Exemplul unei mulţimi de oameni generaţi pe calculator**

*(Sursa:* static.comicvine.com/uploads/original/7/74798/1724644-300\_wallpaper\_q.jpg)



**Figura.1.3 - Exemplu de scena imposibil de realizat în lumea reală**

*(Sursa:* wagthemovie.files.wordpress.com/2012/06/prometheus2.jpg)

2D CGI a fost pentru prima dată folosit în filmele din 1973, Westworld, deşi prima utilizare a imaginilor 3D a fost folosită în 1976, Futureworld- când studenţii Edwin Catmull şi Fred Parke de la Universitatea Utah au creat pe calculator o mână şi o faţă. Primele două filme pentru care s-au făcut investigaţii mari în CGI au fost Tron (1982) şi The Last Starfighter (1984), ambele au fost eşecuri comerciale din cauza modului în care au fost create. Photorealistic CGI nu a avut succes în industria filmului până în 1989 când Abyss a câştigat The Academy Award pentru efecte speciale. Industria Light and Magic a produs apoi efecte speciale photorealistice CGI, incluzând creaţia unui efect de apă numit Weenie.

2D CGI a apărut din ce în ce mai mult în filmele de animaţie “tradiţionale” cu scopul de a aduce ceva în plus atunci când erau utilizate celulele ilustrate (desenate) manual. Întrebuinţările sale pot fi de la “digital tweening motion between frames” la “eye-catching quasi-3D effects” cum ar fi scena balului din Frumoasa şi Bestia.

În 1995 a apărut primul film creat în totalitate pe calculator- Toy Story creat în studioul Pixar, un adevărat succes pentru industria filmului. În plus, celelalte studiouri cum este şi Blue Sky Studios (Fox) sau Pacific Date Images (Dreamworks SKG) au început producerea filmelor folosind CGI, iar companiile existente cum este Disney au început să schimbe desenele animate create în mod traditional cu CGI.

Între anii 1995 şi 2005 media bugetului pentru efecte a crescut foarte mult de la 5 milioane la 40 milioane de dolari. Conform unui director de studio, începând cu anul 2005, mai mult de jumătate din filme aveau efecte speciale semnificative.

La începutul anilor 2000, imaginile generate de calculator au devenit forma dominantă a efectelor speciale. Tehnologia a progresat până în punctul în care a devenit posibilă includerea cascadorilor în mediul virtual, ce au fost aproape indistinctibili de actori reali înlocuiţi. CGI a început să fie utilizat pe scară largă în scenele aglomerate. Cronologia CGI în filme prezintă lista detaliată a utilizărilor iniţiale pentru imaginarul procesat de calculator în film şi televiziune.

Imaginile generate de calculator sunt randate de obicei între 1.4 şi 6 megapixeli. Toy Story, de exemplu, a fost randat la rezoluţie de 1536x922. Timpul de randare al unui frame este în jur de 2-3 ore, şi de 10 ori mai mult în scenele complexe. Acest timp nu s-a schimbat mult în ultimii ani, deoarece calitatea imaginilor a progresat în acelaşi ritm cu îmbunătăţirea hardware-ului.

Dezvoltatorii de tehnologii CGI iau parte în fiecare an la " SIGGRAPH", o conferinţă anuală despre grafica pe calculatoare şi tehnicile interactive, la aceste conferinţe iau parte peste zece mii de profesionişti în calculator.

Dezvoltatorii de jocuri pe calculator şi plăci video 3D depun eforturi uriaşe pentru a obţine aceiaşi calitate vizuală pe calculatoarele personale în timp real cum este posibil pentru CGI în filme şi desene animate. Prin avansarea rapidă a calităţii de randare în timp real, artiştii au început să folosească motoarele de randare folosite în jocuri pentru a produce filme non-interactive. Această formă de artă se numeşte "Machinima".

**Imagini statice şi peisaje**

În CGI nu se creează numai imagini animate ci şi peisaje ce indică natura, ca şi peisaje fractale ce sunt generate de algoritmi. O metodă simplă de a genera feţe fractale este folosirea unei extensii a metodei "thriunghiular mesh". În Figura 1.4 este prezentat un peisaj generat de calculator. Cele mai folosite softuri de generare a peisajelor sunt: Adobe Photoshop şi Vue.



**Figura.1.4 - Exemplu de peisaj CGI**

*(Sursa:* images.wallpaperloft.com/32012/1920x1080-desert-highlands.jpg)

**Scene arhitecturale**

Arhitecţii moderni folosesc servicile firmelor care produc grafică pe calculator ca să creeze modele 3D pentru clienţi şi constructori. Aceste calculatoare generează modele ce pot fi mult mai precise decât desenele tradiţionale. Animaţia arhitecturală (ce oferă filme animate cu clădiri, mai degrabă decât imagini interactive) pot fi de asemenea folosite pentru a vedea mediul înconjurător şi împrejurimile cartierului în care clădirea va fi construită. Realizarea de spaţii arhitecturale, fără folosirea hârtiei şi creionului, este acum un mediu acceptat în proiecţia de arhitectură folosind softuri.

Un soft în care se poate crea modelare arhitecturală permite arhitectului să vizualizeze un spaţiu şi să îndeplinească "walk-throughs" într-o manieră interactivă, astfel medii interactive fiind oferite atat la nivel urban cât şi în construcţii. Aplicaţiile specifice în arhitectură, nu numai că includ specificaţii despre structura clădirii cum ar fi pereţii şi ferestrele, afişarea interioarelor, dar include şi efectele de lumina cum ar fi răsăritul sau apusul soarelui în modul cum vor afecta un design specific la un moment de timp din zi.

Softurile de modelare arhitecturală au fost dezvoltate şi sunt din ce în ce mai folosite prin intermediul internetului. Cu toate acestea, calitatea sistemelor bazate pe internet au încă întreruperi din cauza scenelor ample şi sofisticate, ceea ce duce la modelarea arhitecturală prin intermediul local (softurile sunt instalate pe calculatoarele personale).

În unele aplicaţii, CGI este folosit în reconstrucţia clădirilor vechi. De exemplu, a fost reconstruită o mănăstire din Georgenthal din Germania, ruine mănăstirii folosindu-se pentru a afla suprafaţa clădirii, oferind o imagine de ansamblu cum arăta mânăstirea în zilele acelea.

Cel mai folosit program de proiectare este Autodesk Autocad. În zilele de azi, înaintea construirii unei clădiri mai ample, unei maşini, al oricărui obiect sau construcţie unde dimensiunea acestuia este calculată în milimetrii, mai întâi se va crea un plan 3D cu formă şi dimensiunile reale după care se va crea obiectul respectiv. În Figura 1.5 este prezentată o clădire creată în Autodesk Autocad înainte de proiectarea ei în realitate. După această imagine 3D, directorii companiei au realizat o imagine în ansamblu despre cum va arăta noua clădire, şi pot decide ce îmbunătăţiri se mai pot aduce.



**Figura.1.5 - Exemplu de clădire creată în Autodesk Autocad**

*(Sursa:* www.blblgroup.com/wp-content/uploads/2013/08/Garden-and-building-city-wallpaper-3D-view.jpg)

**Modele anatomice**

Modelele generate pe calculator folosite pentru a crea un schelet nu sunt întodeauna realizate corect anatomic. Cu toate acestea, organizaţii ca "Scientific Computing and Imaging Institute" au creat modele anatomice corecte pe calculator. Generarea de modele anatomice pe calculator poate fi folosită atat în scopuri de instruire cât şi operaţionale. Până în prezent, un număr mare de artişti au produs imagini medicale pentru a fi folosite de studenţii de la medicină. Cu toate acestea, un număr mare de modele anatomice au devenit disponibile pe internet. Razele X aplicate pe un pacient nu reprezintă imagini generate de calculator, chiar şi în cazul radiografiilor digitale. Totuşi, în aplicaţiile care implică scanarea CT, un model 3D este automat produs de o gamă largă de radiografii unice, producând imagini generate pe calculator. Aplicaţiile implică imagistica prin rezonanţă magnetică de asemenea aducând împreună un număr de capturi, în acest caz prin impulsuri magnetice se produce o imagine internă.

În aplicaţiile medicale moderne, modelele specifice ale pacientului sunt construite cu ajutorul "chirurgiei asistate de calculator". De exemplu, pentru o înlocuire completă a unui genunchi, se construieşte un model detaliat specific pacientului pentru a planifica cu atenţie operaţia.

**Simulare şi vizualizare interactivă**

**"**Vizualizarea interactivă" este un termen general ce se aplică la randarea de date care pot varia în mod dinamic şi permite utilizatorului vizualizarea datelor din mai multe perspective. Aplicaţiile pot varia semnificativ, de la vizualizarea unui model dinamic fluid până la vizualizarea unor modele dinamice foarte detaliate, cum ar fi un întreg oraş creat 3D. Datele randate pot corespunde scenelor vizuale specifice mediului înconjurător cum ar fi simularea unui zbor realizată prin folosirea exclusivă a tehnicilor CGI pentru a reprezenta lumea.

**Animaţie pe calculator**

În timp ce peisajele generate pe calculator sunt statice, termenul de "animaţie pe calculator" reprezintă doar imaginile dinamice care seamănă cu un film. Totuşi, în general termenul "animaţie pe calculator" se referă la imaginile dinamice care nu permit utilizatorului să interacţioneze cu mediul respectiv, iar termenul "lume virtuală" este folosit pentru interacţionarea cu mediile animate.

Animaţia pe calculator este în esenţă un succesor digital pentru arta de animaţie "stop-motion" de modele 3D şi animaţii 2D "frame-by-frame". Animaţiile bazate pe calculator sunt mult mai uşor controlabile decât alte procese bazate mai mult pe fizică, cum ar fi construirea de miniaturi pentru efectele fotografiilor sau extragerea unor detalii din scenele aglomerate, şi permit crearea unor imagini imposibil de realizat prin orice alte tehnologii. Un singur artist ce lucrează în grafică poate crea o astfel de scenă fără a se folosi de artişti sau de alte materiale fizice, doar prin intermediul unui calculator performant din cadrul unui studio sau chiar şi de acasă. Pentru a crea iluzia de mişcare, imaginea este afişată pe ecranul calculatorului şi schimbată rapid de o altă imagine ce este similară cu prima avansând încet în timp. De obicei se crează între 24 şi 30 de cadre pe secundă. Această tehnică este identică cu modul în care este creată iluzia de mişcare cu ajutorul aparatului de fotografiat şi cea folosită în trecut.

Unul dintre cele mai folosite softuri ce crează CGI este Autodesk Maya, despre care vom vorbi în subcapitolul următor.

**Lumea virtuală**

O lume virtuală este un mediu simulat ce permite utilizatorului să interacţioneze cu personajele animate sau cu alţi utilizatori prin intermediul folosirii caracterelor animate cunoscute drept "avatare". Lumile virtuale sunt destinate utilizatorilor să "locuiască" şi să interacţioneze, iar în ziua de azi acest termen a devenit sinonim cu interacţionarea mediilor virtuale 3D în care utilizatorii îşi folosesc "avatarele" pentru că identitatea lor să fie recunoscută de restul utilizatorilor. Aceste "avatare" sunt reprezentate 2D sau 3D ca o textură, deşi sunt posibile şi alte forme, cum ar fi cele auditive sau senzitive. Există lumi virtuale care permit şi care nu permit prezenţa mai multor utilizatori.

## Autodesk Maya - soft de creare CGI [10]



**Figura.1.6 - Autodesk Maya**

*(Sursa:* www.autodesk.com/products/autodesk-maya/overview)

*"Maya este o aplicaţie software, destinată modelării grafice tridimensionale şi animaţie. Produs de firma Alias, dar aflat acum în posesia firmei Autodesk Media & Entertainment. Este utilizat pe scară largă în producţia efectelor speciale în cinematografie, în animaţie, cât şi în producţia jocurilor de calculator. Firma Autodesk Media & Entertainment a achiziţionat acest program în octombrie 2005, după cumpărarea firmei Alias Systems Corporatio**n."[[1]](#footnote-1)*

Autodesk Maya este cel mai folosit soft pentru crearea de imagini generate de calculator şi rulează pe toate cele trei platforme: Windows, Mac OS şi Linux. Utilizatorii lucrează într-un spaţiu virtual unde implementează un proiect particular. Scenele se pot salva sub diverse tipuri de format şi se pot importa în alte softuri de grafică cum ar fi 3DS Max, OctaneRender, Modo, etc. În Figura 1.7 este prezentată interfaţa programului Maya şi mediul virtual în care se modelează, texturează, animează şi randează obiectele virtuale.

Maya expune o arhitectură grafică de tip nod. Elementele din scenă sunt bazate pe moduri (pointeri), fiecare nod având propriile atribute şi propriile elemente specifice. Drept rezultat, reprezentarea virtuală a scenei este bazată în întregime pe o reţea de noduri interconectate, în funcţie de informaţiile celorlalte noduri.

Utilizatorii studenţi sau profesori pot descărca o versiune completă educativă de pe comunitatea "Autodesk Education". Versiunile disponibile de pe această comunitate sunt licenţiate doar în caz de folosire necomercial, iar rezultatele proiectelor conţin o inscripţie care contestă faptul că proiectul a fost creat cu o astfel de licenţă. Autodesk Maya este unul dintre cele mai scumpe softuri din cauza performanţei şi abilităţii de creare, precum şi a numărului mare de atribute existente obiectelor.

****

**Figura.1.7 - Interfaţa Autodesk Maya**

*(Sursa:* getintopc.com/wp-content/uploads/2013/07/download-autodesk-maya-2014-free.jpg)

**Dezvoltare**

Maya a fost iniţial un produs de animaţie de ultimă generaţie dezvoltat de "Alias Research", bazat pe codul creat de "The Advanced Visualizer", "PowerAnimator" şi "Alias Sketch!". Codul a fost portat pe sistemul IRIX şi au fost adăugate caracteristicile de animaţie. "Walt Disney Feature Animation" a colaborat cu dezvoltatorii softului Maya în timpul producţiei desenului animat Dinosaur 2000. Studioul Disney a cerut că interfaţa programului Maya să se poată modifica astfel încât studiourile să aibă o interfaţă personalizată. Aceasta a influenţat în mod deosebit arhitectura softului Maya, utilizatorii având posibilitatea de a înţelege mai bine funcţionalitatea softului şi motivul pentru care softul a devenit popular în industria filmului.

După ce compania "Silicon Graphics" a cumpărat companiile "Alias" şi "Wavefront Technologies" (Wavefront dezvoltând tehnologii de ultima generaţie) a continuat dezvoltarea tehnologiilor şi le-au unit într-un singur soft numit Maya. Achiziţia companiei "Silicon Graphics" a fost o reacţie pentru achiziţionarea softului "Softimage" de către "Microsoft Corporation". După unirea firmelor, noua organizaţie a fost numită "Alias|Wavefront".

În primele zile de dezvoltare , softul Maya a început cu limbajul de scripting TCL având în vedere crearea unui limbaj de scripting ca şi cel al lui Unix. Dar, după ce s-a unit cu "Wavefront", limbajul de scripting din softul Dynamation deţinut de compania Wavefront, numit Sophia, a fost ales ca şi baze ale limbajului de scripting MEL.

Prima versiune de Maya a fost lansată în februarie în anul 1998. În urma unei serii de achiziţii, softul Maya a fost cumpărat de către Autodesk în anul 2005. Sub numele companiei părinte, Maya a fost redenumit Autodesk Maya. Totuşi, utilizatorii softului Autodesk Maya au continuat să numească softul doar "Maya".

În data de 1.03.2003, Alias a fost onorat de către "Academy of Motion Picture Arts and Sciences" cu un premiu "Academy Award for Technical Achievement" pentru realizarea ştiinţifică şi tehnică în dezvoltarea softului Maya.

În 2005, Jos Stam, un dezvoltator al companiei "Alias|Wavefront", a câştigat împreună cu Edwin Catmull şi Tony DeRos premiul "Academy Awared for Technical Achievement" pentru inventarea aplicaţiei de creare a subdiviziunilor suprafeţelor.

În 08.02.2008, Duncan Brinsmead, Jos Stam, Julia Pakalns şi Martin Werner au primit premiul "Academy Awared for Technical Achievement" pentru crearea designului şi implementarea sistemului "Maya Fluid Effects".

**Utilizare**

Maya a fost folosit pentru a crea grafică multor filme de cinema, incluzând câştigătorii companiei "Academy Award" pentru producerea celor mai vândute filme din lume The Girl with the Dragon Tattoo, Avatar (2009), Hugo, şi Rango. Softul este de asemenea folosit pentru a crea efecte vizuale pentru programe de televiziune, din care fac parte Game of Thrones, The Walking Dead, Once Upon a Time, Bones şi Boardwalk Empire.

Cu ajutorul acestui soft au început să se creeze efecte vizuale şi pentru jocuri video cum ar fi Halo 4.

În zilele de azi, Maya este cel mai folosit soft pentru crearea efectelor speciale din cauza faptului că prin cu acest program este posibil realizarea unui film de la zero fără a se folosi alte softuri.

## Plugin - cheia dezvoltării softurilor de grafică [11][12]

*"Un plugin sau insert/Add-on este un program care poate să se integreze într-un alt program (de bază) pentru a îndeplini funcţii specifice. Exemple tipice de plugin-uri sunt cele pentru afişarea formatelor grafice (de ex. SVG dacă browserul nu include implicit acest format), pentru a executa fişiere multimedia, pentru a cripta/decripta e-mail-uri (de exemplu PGP), sau pentru a filtra imagini în programe grafice. Programul de bază (browserul sau un clientul de poştă electronică ş.a.) setează un standard pentru schimbul de date cu plugin-ul, permite accesul plugin-urilor la datele din program şi execută cererile plugin-urilor. Mulţi producători de software cu renume oferă proiectanţilor de programe API-uri ("Application Programming Interface") pentru plugin-uri, pentru a mări utilitatea produsului de bază."[[2]](#footnote-2)*

Cele mai cunoscute plugin-uri incluse în browser sunt Adobe Flash Player, QuickTime Player şi Java, care pot lansa un "user-activated Java applet" pe o pagină web pentru executarea unei maşini virtuale locale Java.

"Add-on" este termenul general pentru ceea ce îmbunătăţeşte o aplicaţie. El cuprinde snapin-uri, plugin-uri, teme şi skin-uri.

***"Scopuri si exemple***

*Aplicaţiile suportă plugin-uri pentru foarte multe motive. Cele mai importante motive sunt:*

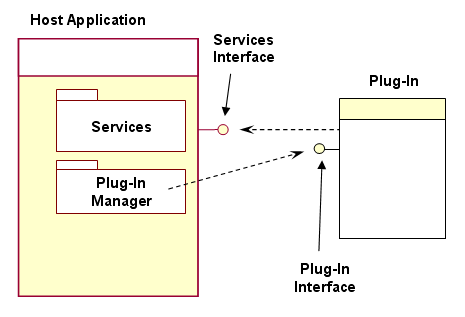
* *pentru a permite dezvoltatorilor terţi să creeze moduri pentru a extinde o aplicaţie existentă;*
* *pentru a adăuga cu uşurinţă îmbunătăţirii;*
* *pentru a reduce dimensiunea unei aplicaţii;*
* *pentru a separa codul sursă de aplicaţie din cauza licenţelor software incompatibile.*

*Exemple specifice de aplicaţii în care se folosesc plugin-uri:*

* *softuri de editare audio folosesc plugin-uri pentru a genera, procesa şi analiza sunetul*
* *email-urile clienţilor folosesc plugin-uri ca să cripteze şi să decripteze email-uri.*
* *softurile de grafică folosesc plugin-uri ca să citească mai multe tipuri de format şi să proceseze imagini*
* *Media players folosesc plugin-uri ca să citească mai multe tipuri de format şi pentru a crea filtre pentru afişarea mai bună a informaţiei*
* *Microsoft Office foloseşte plugin-uri (mai bine ştiute ca şi add-ins) ca să extindă abilitatea aplicaţiilor adăugând comenzi personalizate şi specificaţii specializate.*
* *Packet sniffers foloseşte plugin-uri ca să decodifice pachete."*[[3]](#footnote-3)

**Mecanism**

O aplicaţie gazdă oferă servicii pe care un plugin le poate folosi, incluzând o modalitate pentru plugin-uri ca să se înregistreze singure în aplicaţia gazdă şi un protocol pentru schimbarea de date între aplicaţia gazdă şi plugin. O astfel de relaţie este prezentată în Figura 1.8. Plugin-urile depind de serviciile oferite de către aplicaţia gazdă şi de obicei ele nu funcţionează de unele singure. În mod invers, aplicaţia gazdă funcţionează independent de plugin-uri, făcând posibil ca utilizatorii să adauge şi să updateye plugin-uri dinamic fără necesitatea de a face schimbări în aplicaţie gazdă.



**Figura.1.8 - Exemplu de Plugin Framework**

*(Sursa:* en.wikipedia.org/wiki/File:Plug-InExample.png*)*

***" Diferenţa dintre Plugin şi extensie***

*Extensiile diferă semnificativ faţă de plugin-uri. Plugin-urile de obicei au un set mic de competenţe. De exemplu, impulsul original din spatele dezvoltării softului Mozilla Firefox a fost urmărit de o mică aplicaţie de bază, lăsând funcţionalitate personală să fie implementată de către extensii pentru a evita viitoare defecţiuni tehnice. Acest lucru este în contrast cu abordarea "kitchen sink" a predecesorlori săi, Mozilla Application Suite şi Netscape 6 şi 7. Din fericire, după integrare, extensiile pot fi văzute ca şi o parte din aplicaţie în sine, adaptate de la un set de module opţionale.*

***Istoria şi dezvoltarea Plugin-ului***

*Plugin-ul a apărut la începutul anilor 1970 când EDT text editor funcţiona pe sistemul de operare Unisys VS/9 folosind seriile Univac 90/60 şi oferind calculatorului abilitatea de a rula un program în text editor şi permiţându-i unui astfel de program accesul la "editor buffer", dându-i posibilitatea astfel unui program extern să acceseze o secţiune de editare în memorie. Programul plugin poate face apel la editor pentru efectuarea servicilor de editare pe "buffer", pe care editorul îl împarte cu plugin-ul. Compilatorul Waterloo Fortran a folosit această îmbunătăţire pentru a permite compilarea interactivă a programelor Fortran editate de EDT.*

*Pe sistemul de operare Macintosh au fost introduse pentru prima dată aplicaţiile software ce includeau funcţionalitatea plugin-urilor încorporate HyperCard şi QuarkXPress, amândouă lansate în 1987. În 1988, Silicon Bach Software a inclus funcţionalitatea plugin-ului în Digital Darkroom şi SuperPaint, iar Ed Bomke a inventat termenul de Plugin.*

*În prezent, programele ce beneficiază de funcţionalitatea implementării plugin-urilor folosesc librării comune instalate obligatoriu într-un loc prescris de aplicaţiile gazdă. HyperCard suportă facilităţi similare, dar mult mai comun includ codul plugin-ului în documentele însuşi HyperCard."[[4]](#footnote-4)*

**Crearea unui Plugin pentru Maya**

La fel cum a fost menţionat mai sus, accesul la codul sursă al softurilor este interzis pentru utilizatori. În acelaşi timp utilizatorii softului Autodesk Maya, în cele mai multe cazuri utilizatorii făcând parte dintr-un studio, doresc să facă modificări softului pentru a realiza finalizarea unui proiect. Aceşti utilizatori fac parte din departamentul numit "Pipeline", şi au ca scop întreţinerea funcţionalităţii în întregime a softurilor şi rezolvarea problemelor tehnice apărute în timpul producţiei. Din cauza numărului mare de artişti care participă la un proiect, sau chiar numărul total de artişti angajaţi într-un studio, departamentul Pipeline ţine sub control rezolvarea problemelor folosind diverse plugin-uri ce sunt instalate pe toate calculatoarele (sau într-un studio mai modern, plugin-urile sunt instalate şi utilizate de către artişti direct de pe servere).

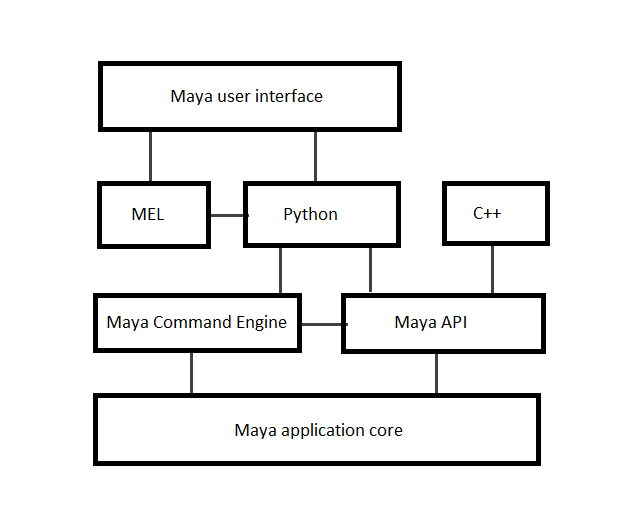
Cu ajutorul acestei abilităţi, artiştii îşi crează Plugin-uri pentru a le uşura munca, şi totodată pentru a preveni generarea unor erori inevitabile. În capitolele următoare se va prezenta limbajul de programare Python folosit pentru crearea unui astfel de script, integrarea limbajului de programare Python în softul Autodesk Maya, descrierea şi funcţionalitatea unui astfel de plugin plus modul de utilizare al acestuia.

# Capitolul 2. Programarea orientată pe obiecte în Python. Interacţiunea Python cu Maya

Pentru a înţelege ce se poate crea cu Python în Maya, trebuie mai întâi să înţelegem cum a fost Maya proiectată. Sunt câteva moduri în care utilizatorii pot interacţiona sau modifica programul Maya. Metoda standard este capacitatea de a folosi Maya's graphical userinterface (GUI). Această lucrează ca oricare altă aplicaţie software: utilizatorii apasă butoane sau selectează elemente din meniuri ce crează sau modifică documentele ori spaţiul de lucru al lor. Pe lângă multe alte aplicaţii software, programul Maya este unic în multe feluri. Maya este un program open product, creat pentru a fi capabil să suporte noi modificări, acestea pot include de la o simplă comandă care afişează numele unui obiect până la modificarea interfeţei.

## Interacţionări cu Maya [1]

Autodesk a creat 4 interfeţe de programare care interacţionează cu Maya, folosind 3 limbaje de programare diferite. Orice obiect creat în Maya va folosi câteva combinaţii dintre aceste interfeţe ca să creeze rezultatul văzut în workspace. În Figura 2.1 se ilustrează cum aceste interfeţe interacţionează cu Maya.



**Figura.2.1 - Arhitectura interfeţei programului Maya**

*(Sursa: Maya Python for Games and Film. A complete Reference for Maya Python and the Maya Python API, Adam Mechtley, Ryan Trowbridge, Morgan Kaufmann, Unite State of America, 2012)*

**Maya Embedded Language (MEL) [1]**

Maya Embedded Language (MEL) a fost dezvoltat pentru modificarea programului Maya şi a fost utilizat exclusiv pe tot parcursul programului. MEL script defineşte şi crează Maya GUI. Maya's GUI executa MEL instrucţiuni şi comenzi Maya. Utilizatorii pot, de asemenea, să scrie propriile lor scripturi în MEL pentru a îndeplini sarcini comune. Scripturile MEL sunt relativ uşor de creat, modificat şi executat, dar sunt folosite numai de către Maya şi sunt limitate din punct de vedere tehnic. Şi anume, MEL nu deţine suport pentru programarea obiect orientată. MEL poate să comunice cu Maya doar prin nişte interfeţe definite în Command Engine.

**Python [1]**

Python este un limbaj de scripting ce a fost oficial introdus în Maya în versiunea 8.5. Python poate executa aceleaşi comenzi Maya ca şi MEL folosind Maya's Command Engine. Cu toate acestea, Python este mult mai viguros decât MEL deoarece este un limbaj de programare obiect-orientat. Mai mult Python a fost creat încă din 1980 şi are o librărie extinsă precum si o comunitate largă în afara utilizatorilor care folosesc Maya.

**C++ Application Programming Interface [1]**

Maya C++ application programming interface (API) este cea mai flexibilă cale de a adăuga noi modificări în Maya. Utilizatorii pot adăuga noi obiecte pentru Maya care pot face execuţia programului sau a scriptului mult mai rapidă. Cu toate acestea, tool-urile dezvoltate folosind C++ API trebuie să fie compilate pentru noile versiuni de Maya şi, de asemenea, pentru fiecare sistem de operare. Deoarece se cere compilare, C++ API nu poate fi folosit interactiv cu Maya userinterface.

**Python API [1]**

Când Autodesk a întrodus Python în Maya, de asemenea, a creat un "înveliş" pentru multe dintre clasele Maya C++ API. Ca atare, dezvoltatorii pot folosi funcţionalitatea API pentru Python. Scopul de a accesa clasele prin Python API a crescut şi s-a îmbunătăţit cu fiecare versiune nouă de Maya. Datorită acesteia, utilizatorii pot manipula foarete uşor obiectele Maya API în scripturi simple, precum creaţia de plugin-uri pentru Maya într-un mod foarte uşor.

## Python versus MEL în Maya [1]

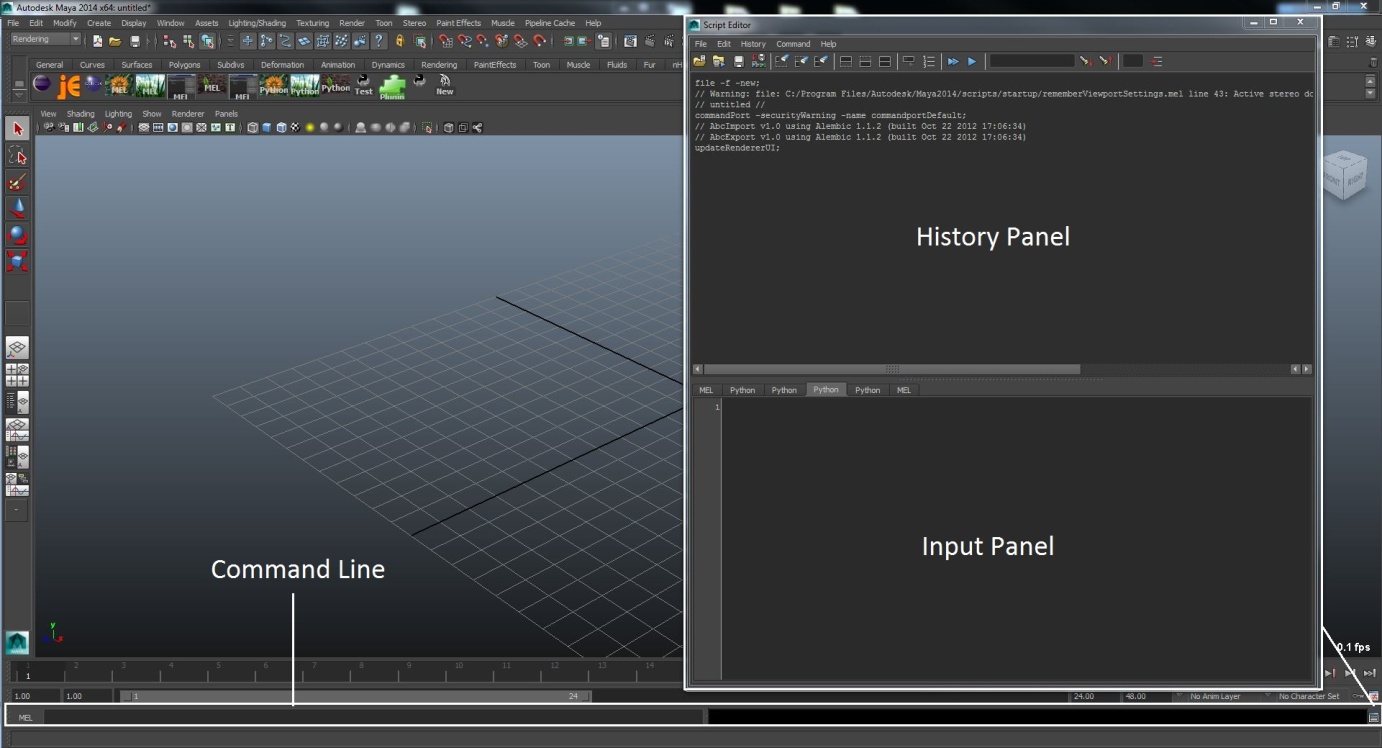
Sunt multe motive pentru care trebuie să folosim Python în loc de MEL, dar asta nu înseamnă că trebuie să renunţăm total la MEL. Dacă studioul are deja câteva scripturi scrise în MEL nu există niciun motiv ca ele să fie convertite în Python dacă ele îşi îndeplinesc treaba. Se pot integra uşor scripturile scrise în Python pentru dezvoltarea pipeline-ului cu scripturile scrise în Python. Python poate accesa scripturile MEL şi MEL poate accesa scripturile Python. Python este un limbaj complex la fel ca şi C++, dar spre deosebire de C++, Python are o sintaxă simplă şi foarte uşor de folosit.

Cu toate acestea, Python deţine date mult mai complexe decât MEL. Programele MEL câteodată încearcă să imite structuri de date complexe, dar făcându-se foarte dese cereri de modificare a scriptului, se creează un cod "dezordonat" , iar extinderea lui poate deveni neplăcută. În timp ce Python e obiect-orientat şi permite variabile "curate", poate deţine aceste situaţii mult mai uşor. Mult mai mult, Python poate accesa fişiere şi sisteme de date mult mai repede decât MEL, făcând tool-urile mult mai rapide pentru artişti în producţie. Programatorii au mult mai multe opţiuni în Python decât în MEL. În timp ce Python este un program de programare mult mai vechi, nu este necesar crearea unor module deoarece ele se pot găsi gata create.

## Executarea limbajului de programare Python în Maya [1]

Maya are multe tool-uri construite în GUI ce permite utilizatorilor să execute cod în Python. Aceste tool-uri sunt:

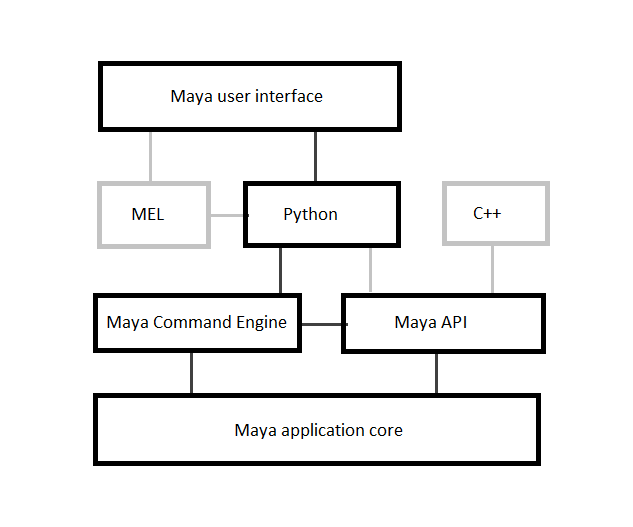
* **Command Line** localizat de-a lungul interfeţei grafice în partea de jos. Figura 1.2. Command Line apare în Maya GUI ca default. De obicei în Command Line se introduc linii de cod simple.
* **Script Editor** este cel mai important tool pentru cei care programează în Maya. Script Editor este o interfaţă pentru crearea unor scripturi scurte care interacţionează cu Maya. El conţine 2 ferestre (Figura 2.2 dreapta). În partea de sus se află History Panel în care sunt afişate toate comenzile executate şi, de asemenea, toate rezultatele comenzilor executate. În partea de jos se află Input Panel, în care scriem codul în Python sau în MEL.



**Figura 2.2 - Interfaţa de programare în Maya GUI**

(Sursa: imaginea autorului realizată cu programul Autodesk Maya)

Deşi, Maya este open product, datele din core nu se pot accesa de către utilizatori. Inginerii Autodesk pot face modificări la nivel de core de la o versiune la alta, dar utilizatorii pot să acceseze aplicaţiile din core numai cu ajutorul unui set de interfeţe pe care Autodesk le asigură. O astfel de interfaţă ce poate comunica cu nucleul este Command Engine.



**Figura 2.3 Python interacţionează cu Maya Command Engine**

*(Sursa: Maya Python for Games and Film. A complete Reference for Maya Python and the Maya Python API, Adam Mechtley, Ryan Trowbridge, Morgan Kaufmann, Unite State of America, 2012)*

În Figura 2.3 putem vedea cum Python interacţionează cu Maya Command Engine. În timp ce Python poate să folosească comenzi gata create pentru a găsi datele din core, el poate, de asemenea, să cheme comenzi create de utilizatori care folosesc interfeţe API ca să manipuleze şi să regăsească date din core. Aceste date se pot returna într-o interfaţa de scripting prin Command Engine. Această abstracţie permite utilizatorilor să apeleze comenzi de bază (care stau la baza corului) prin intermediul limbajului de scripting.

MEL are acces la peste 1000 de comenzi cu ajutorul căruia s-au creat aproape toate interfeţele Maya's GUI. În timp ce Python are acces la aproape toate aceleaşi comenzi (asta asigură ca Python poate fi folosit pentru a recrea Maya GUI la fel cum a fost creată în MEL), există un set de comenzi neaccesibile de către Python. Aceste comenzi ce nu sunt accesibile de către Python includ specificaţii legate de MEL sau de sistemul de operare. Dar, cu toate acestea, din cauză că Python are o bibliotecă largă de utilităţi ce a crescut în mulţi ani, această diferenţă dintre Python şi MEL cu timpul poate dispărea.

Maya are documentaţie pentru toate comenzile din Python şi este uşor de găsit care comenzi sunt disponibile. Cu toate acestea, se poate executa un script ce deţine şi cod în MEL şi cod în Python cu ajutorul unor comenzi simple. Mai mult decât atât, într-un script scris în Python sau în MEL se poate apela la un alt script scris în MEL respectiv Python. O altă calitate importantă a Maya Command Engine este cât de uşor se pot crea comenzi ce lucrează cu MEL şi cu Python în acelaşi timp. Programul Maya a fost creat în aşa fel încât, în momentul când se creează o comandă nouă, ea va fi adăugată şi disponibilă automat pentru ambele limbaje de programare- MEL şi Python. Noile comenzi pot fi create cu Maya C++ API sau cu Python API.

## Diferenţele principale dintre Python şi alte limbaje de programare

**Sintaxa:**

Foarte multe limbaje de programare nu ţin cont de spaţii şi taburi, în schimb folosesc mecanisme cum ar fi acoladele pentru indicarea blocurilor de cod. Python pe de altă parte foloseşte spaţiile pentru a structura blocurile de cod. În majoritatea limbajelor de programare un *for* poate arăta aşa:

*for (int $i=0; $i<5; $i++){*

*int $j=$i+1;*

*print ($j+"\n");*

*}*

Sau aşa:

f*or (int $i=0; $i<5; $i++){ int $j=$i+1; print ($j+"\n");}*

În schimb ce în limbajul de programare Python un *for* arată aşa:

f*or i în range (5)*

*j = i+1*

*print j*

**Timpul de executare:**

Diferenţa dintre timpul de executare al limbajului de programare MEL şi al limbajului de programare Python este foarte mare pentru scenele complexe. Executarea unui script în Python poate dura şi de 200 de ori mai puţin decât executarea aceluiaşi script scris în MEL, deoarece Python a fost creat să folosească Maya Python API. Este mult mai complicat folosirea API (Application programming interface) decât comenzile crate special pentru Maya în MEL, cu toate acestea MEL nu poate folosi API direct, ci numai cu ajutorul acestor comenzi încapsulate.

**Modelul de date din Python**

Pentru o eficienţă maximă, pentru tipurile de variabile statice diferite se alocă o calitate specifică de memorie. Deoarece această cantitate de memorie este specificată de la început, nu se poate atribui simplu un nou tip la o anumită variabilă la un moment ulterior. Pe de altă parte, limbajul Python permite să se schimbe tipul unei variabile oricând. În Python, variabilele sunt doar nişte nume care indică date. Toate datele sunt obiecte, şi fiecare obiect are o identitate, un tip şi o valoare.

* O identitate obiect descrie o adresă de memorie.
* Un tip obiect (care este el în sine un obiect) descrie un tip de date de referinţă.
* O valoare obiect descrise conţinutul real al datelor.

Când se crează o nouă variabilă şi i se atribuie o valoare, variabila este numai un nume ce indica spre un obiect cu aceste trei proprietăţi: o identitate, un tip şi a valoare. Deşi variabilele din Python sunt referinţe către date, ele nu pot fi folosite ca şi pointeri. Python întotdeauna introduce parametrii prin valoare, dar valoarea unei variabile în Python este o referinţă. Consecinţa există atunci când se reatribuie o variabilă în interiorul unei proceduri, ea nu are niciun efect înafara procedurii.

Exemplu:

*var = 5*

*var = 'Hello!'*

Prima dată i se atribuie variabilei *var* un număr întreg, după care i se atribuie variabilei un string. În acest caz, nu se schimbă tipul de date care sunt la bază, ci variabila indică spre o parte de date diferită în acelaşi timp.

## Ce este un modul în Python? [1]

Un modul este un fişier Python independent ce conţine o secvenţă de definiţii şi declaraţii executabile. Modulele pot conţine atât de puţine sau atât de multe linii de cod cât sunt cerute, şi des focusandu-se pe o sarcină specifică. De asemenea modulele pot importa alte module, acordând organizarea codului prin funcţionalitate şi reutilizarea funcţionalităţii dincolo de tool-uri. În timp ce modulele permit uşor refolosirea codului în această situaţie, ele sunt de asemenea create să protejeze codul. Modulele sunt nişte documente de tip text cu extensie ".py". Când se importă un modul pentru prima dată, în acelaşi timp, un alt fişier, de tip ". pyc" ce îi corespunde va fi generat. Acest fişier este un modul de compilare. Un modul de compilare conţine bytecode, mai degrabă decât cod sursă. În timp ce codul sursă este "human readable", bytecode este convertit într-o formă compactă care sunt mult mai eficientă pentru interpretorul Python să o execute. Ca atare, se pot folosi modulele ori în fişiere de tip "py" ori în fişiere de tip "pyc". Modulele care se execută în Maya sunt numai de tip bytecode pentru protecţia conţinutului.

Python însuşi colaborează cu o largă bibliotecă de module standard. În această bibliotecă există tot felul de module pentru diverse sarcini, inclusiv cele ce ţin de matematică, cele ce lucrează cu e-mail-uri, chiar şi cele ce folosesc interfaţa sistemului de operare. În documentaţia despre Python există o listă plină cu module deja create şi descrierea funcţionalităţilor lor. De multe ori există module deja create ce rezolvă unele probleme ce pot lua mult timp pentru a fi rezolvate.

Pe lângă aceste module direct create, sunt multe alte module folositoare disponibile în comunitatea Python. De exemplu, pachetul "numpy" conţine module folositoare despre ştiinţa calculatoarelor şi algebră lineară, în timp ce pachete ca "pymel" oferă tool-uri puternice în mod special pentru extinderea comenzilor din Maya şi modulelor API.

**Domeniul** este, de asemenea, important când lucrăm cu module. Când se execută o linie de cod în Script Editor, se lucrează pe un anumit domeniu. Python are 2 funcţii încorporate, **globals ()** şi **locals ()**, ce permite vizualizarea obiectelor existente în domeniile curente global sau local. Dacă variabilele fiecărui script creat sunt adăugate într-un singur domeniu, va fi foarte dificil să se ţină evidenţa datelor. De exemplu, dacă se vor executa 2 scripturi unul după altul, şi ele conţin variabile cu acelaşi nume, s-ar putea că scripturile să intre în conflict. Această problemă apare foarte des în limbajul de programare MEL. Spre deosebire de tabela de simboluri globale, tabela de simboluri locale, accesata cu funcţia locals (), afişează toate numele simbolurilor definite în domeniul curent, oricare ar fi el. Dacă se execută funcţia local () în primul nivel, locals () şi global () vor fi identice, diferenţa nefiind evidenta. Un domeniu local se poate diferenţia numai dacă este executat în interiorul unei funcţii. Tabela de simboluri locale într-o funcţie conţine numai variabilele declarate în acea funcţie, acel domeniu.

**Încapsularea şi atributele modulelor**

Modulele sunt create sa încapsuleze date în tabela lor proprie de simboluri, eliminând "grijile" referitoare la rescrieri de date accidental. Fiecare modul are propria tabelă de simboluri, care se poate accesa folosind funcţia global (). În mod implicit, importarea unui nou modul este adăugat în tabela de simboluri globală, dar variabilele lui şi alte date sunt accesibile doar prin folosirea numelui atribuit lor în spaţiul de lucru curent. Variabilele şi funcţiile care sunt parte dintr-un modul se numesc atribute. De când Python încapsulează datele într-un modul, se poate importa în siguranţă un modul în alt modul. Totuşi, este important de ştiut că atributele unui modul nu sunt private. Este posibil că valoarea atributelor să fie modificată extern. Acest aspect al limbajului Python este de fapt incredibil de folositor în unele cazuri, astfel încât îţi permite să extinzi orice atribute, incluzând şi funcţii. De exemplu, poţi extinde stringuri când se afişează obiectele API în consolă, totuşi, cu această mare "putere de modificare" vin şi mari responsabilităţi. În acest caz, verificăm efectiv existenţa valorii altei variabile şi o reverificăm când terminăm.

**Atributele implicite**

Maya Script Editor, de asemenea, pune datele introduse de utilizator într-un modul. Acest modul se numeşte **\_\_main\_\_**. Fiecare modul are mai multe atribute standard, unul dintre acestea se numeşte **\_\_name\_\_**. Acest atribut afişează numele modulului în care se lucrează. Maya Script Editor este modulul root şi toate celelalte module se ramifică de la el. Acest concept este ilustrat în Figura 2.4. În Maya, scripturile create de utilizatori nu pot fi module \_\_main\_\_, aşa încât Maya acţionează ca modul \_\_main\_\_. Crearea unui modul este simplu de realizat. Codul se poate scrie în orice text editor şi salvat cu extensie ".py".



**Figura 2.4 - Ierarhia a trei module importate**

*(Sursa: Maya Python for Games and Film. A complete Reference for Maya Python and the Maya Python API, Adam Mechtley, Ryan Trowbridge, Morgan Kaufmann, Unite State of America, 2012)*

Pe lângă atributul \_\_name\_\_, mai există alte 4 atribute, ce încep şi se termină cu "\_\_", ce sunt incluse în orice modul creat, chiar dacă ele nu sunt create în mod special. Acestea sunt:

* **\_\_builtins\_\_** - acest atribut accesează un modul ce conţine toate funcţiile încorporate ca şi print(), globals (), etc; şi obiecte ca int, float, etc. Acest modul este încărcat întotdeauna automat şi nu cere pentru accesarea atributelor lui folosirea denumirii modulului în faţa atributelor atunci când este folosit într-un spaţiu de lucru.
* **\_\_file\_\_** - acest atribut al unui modul este un simplu string ce indică calea absolută către un modul dintr-un fişier cu extensie ".py" pe sistemul de operare.
* **\_\_package\_\_** - acest atribut a unui modul returnează numele pachetului din care face parte. Dacă modulul nu face parte dintr-un pachet se va returna valoarea *None*.
* **\_\_doc\_\_** - acest atribut este foarte folositor când este folosit corect. Într-un limbaj Python, primul comentariu după definirea modulului, este documentarea despre modulul respectiv. De asemenea, fiecare funcţie are propriul atribut \_\_doc\_\_. Dacă adăugaţi comentarii la un modul în acesta locaţie, se poate folosi funcţia încorporată în Python, **help ()**, pentru a obţine informaţii.

**Pachetele**

În plus faţă de părţile obişnuite, modulele individuale, Python permit organizarea modulelor într-o structura ierarhică specială numită **pachete.** Pachetele sunt nişte foldere tratate ca şi modulele. Pachetele permit dezvoltarea tool-urilor într-un mod mult mai organizat, mai degrabă decât să le impună utilizatoriilor umplerea unui folder cu o mulţime de module. Pachetele asigură avantaje tehnice peste modulele obişnuite. Pentru început, se evită conflictele legate de nume prin punerea modulelor într-un pachet, chiar şi pachete în pachete, reducând probabilitatea conflictelor cu alte tool-uri. În al doilea rând, modulele introduse într-un pachet nu trebuie să aibă locaţia pe disk explicit definită adăugată în calea de căutare a limbajului Python. Deoarece folderul pachet însuşi este tratat ca şi un modul, se pot accesa modulele din pachet fără a avea adăugată locaţia specifică a lor în "calea de căutare" a limbajului Python. Pachetele sunt foarte uşor de creat. Cerinţa de bază este aceea de a adăuga un fişier, \_\_init\_\_.py, într-un folder ce va fi tratat ca un pachet sau un subpachet. După aceea, se poate importa folderul ca şi cum ar fi un modul sau importă orice modul din folder folosind punctul de delimitare. În fişierul \_\_init\_\_.py se scrie doar fişierele care se importă din pachetul respectiv. În practică, se folosesc pachete pentru a organiza tool-uri complexe pe o bază funcţională. Un exemplu ar fi următorul: se pot pune toate tool-urile proiectului într-un pachet, iar acesta are de asemenea subpachete în interiorul lui pentru animaţie, efecte, modelare, rigging, utilităţi s.a.m.d. Uneori se pot face greşeli când sunt create module care au acelaşi nume ca modulele încorporate; şi, chiar dacă acest lucru are loc, pachetele au grijă ca tool-urile să nu intre în conflict cu celelalte tool-uri instalate de persoanele care folosesc pachetul.

**Importarea modulelor**

Prima dată când se importă un modul, un fişier de tip ". pyc" se va genera automat. Acest fişier ". pyc", ce conţine bytecode, este citit de către interpretor în timpul executării. O consecinţă importantă la implementarea aceasta este aceea că toate importările ulterioare unui modul în timpul unei singure secţiuni, din modulul \_\_main\_\_ sau orice alte module, se vor raporta la fişierul bytecode dacă acesta există. Calculatorul citeşte mult mai repede fişiere de tip bytecode decat fişierele normale. În cazul unor modificări in fisierele de tip ".py", trebuie folosită funcţia ***reload ()*** pentru a se face modificarea şi în fişierul "pyc"***.***Această funcţie va recompila fişierul bytecode pentru modulul care a fost trecut ca şi un argument, deşi nu va recompila automat niciun modul ce este importat în modulul respectiv. Prin urmare, folosind funcţia *reload ()*, nu numai că va repeta orice declaraţie executabilă în modul, de asemenea, va adapta orice atribut în modul ce reflectă orice modificare făcută în codul sursă. Apelând funcţia *reload ()* într-un spaţiu de lucru nu va afecta instanţele existente ale modulului în alte spaţii de lucruri. În modul în care Python gestionează memoria, celelalte variabile vor indica încă spre datele modulului înainte de reîncărcare. Mai mult, reîncărcând un pachet nu va reîncărca automat toate subpachetele lui. Este important ca utilizatorii limbajului de programare MEL să se confrunte cu funcţia *reload ()* din Python şi să o compare cu instrucţiunea *source* din MEL. În timp ce funcţia *reload ()* din Python va recompila şi reexecută modulul specificat, instrucţiunea source din MEL, doar va reexecuta conţinutul scriptului, dar nu îl va recompila. Este posibil să se folosească comanda *eval* din MEL în colaborare cu instrucţiunea source pentru a declanşa recompilarea scriptului MEL.

Uneori un modul este adânc încorporat într-un pachet, ori are un nume foarte lung. Pentru a scurta scrierea şi citirea unui script se poate folosi cuvântul cheie "**as"** ce permite atribuirea unui nume ales modulului când este adăugat tabelei de simboluri globale în timpul importării. De exemplu la importarea modulului *maya.cmds* putem să îi atribuim un alt nume şi să folosim acest nume pe parcursul scriptului: *import maya.cmds as mc; mc.polySphere ().*

Python asigură de asemenea sintaxe speciale pentru a importa atribute specifice dintr-un modul în tabela de simboluri globală. De exemplu, modulul *math* conţine o funcţie numită *acos()*. Dacă în modulul în care lucrezi nu există nicio funcţie cu acest nume poate fi folosită doar importând aceste atribute direct în tabela de simboluri globală pentru a le accesa fără prefix. Python permite pentru realizarea acestei sarcini folosirea cuvântului cheie **form**. Acest model permite să se specifice individual, delimitate prin virgulă, atributele pe care vrem să le importăm sau printr-un asterix să importăm toate atributele din modul. Dar trebuie să fim precauţi când folosim acest tip de parametrii. Sistemul spaţiului de lucru al limbajului Python este oferit ca avantaj şi protejat. Acest lucru nu trebuie văzut ca o ameninţare. Spaţiul de lucru se poate proteja de orice conflict neprevăzut, în mod special când se lucrează pe proiecte mari cu alte dezvoltări. De exemplu, se pot importa toate atributele unui modul într-o tabelă de simboluri globale, iar un alt utilizator, mai târziu, poate adăuga un atribut, a cărui nume poate intra în conflict cu numele unui atribut deja importat. În multe cazuri, este mai sigură folosirea cuvântului cheie as şi atribuirea modulului importat unui nume scurt. În Python multe atribute sunt prefixate cu liniuţe de tip underscor, care indică că ele sunt simboluri interne. Ele se pot trata ca şi cum ar fi private. Această convenţie se asigură de asemenea că aceste atribute nu sunt importate în tabela de simboluri globală când se foloseşte sintaxa de importare a tuturor atributelor modulului cu cuvântul cheie form şi asterix. Acest program a fost conceput aşa, deoarece atributele implicite ce sunt create în modulul principal sa nu se rescrie în momentul importarii altui modul.

## Calea în Python [1]

În limbajul de programare MEL este posibilă folosirea instrucţiunii *source* şi adăugarea căii către locaţia în care se află scriptul, oriunde s-ar afla aceasta, pentru a fi încărcat scriptul. Nu există o opţiune alternativă ca şi în MEL pentru a importa calea în limbajul de programare Python. Când se încearca importarea unui modul, Python caută în toate folderele care au fost specificate, în fişierul în care sunt declarate toate căile către folderele ce deţin fişiere de tip "py", numit "calea Python". În Python, toate modulele trebuie să fie localizate în unul dintre aceste foldere sau în interiorul unui pachet din aceste foldere. Există o varietate de opţiuni disponibile pentru adăugarea folderelor în "calea Python".

**Sys.path** -unul dintre mecanismele de a lucra cu "calea Python" este atributul **path** în modulul sys. Acest atribut conţine o listă cu toate folderele în care Python "caută" fişiere cu extensie "py" sau "pyc", şi se poate lucra interactiv cu ea ca şi cu orice altă listă, incluzând adăugarea în ea. Căile către folderele din această listă sunt disponibile de îndată ce sesiunea Python a început.

**Adăugarea temporară a unei căi** - deoarece atributul path este o listă simplă, este uşor de adăugat în ea ca şi într-o altă listă. Se poate beneficia de această funcţionalitate pentru a adăuga temporar un folder în calea Python în timpul unei secţiuni din Maya. Trei dintre cele mai comune opţiuni sunt:

* **scriptul userSetup** - un lucru folositor în Maya este abilitatea de a crea un script userSetup. Un simplu script userSetup trebuie să existe într-o locaţie proprie pe disk (pe sistemul de operare Windows această locaţie este C:\Users\UserName\Documents\maya\2014-x64\scripts), şi Maya îl va executa la pornirea programului de fiecare dată. Numai un script userSetup se ia în considerare, fie MEL fie Python. Un script userSetup nu este complet încapsulat ca şi modulele obişnuite. Toate declaraţiile lui sunt executate în modulul principal, modulul *\_\_main\_\_,* folosind funcţia incorporată *execfile (),* deci orice modul importat din userSetup este automat disponibil în *\_\_main\_\_* unde Maya este iniţializată.
* **modulul sitecustomize -** acest modul este întotdeauna executat când secţiunea Python începe. Acesta modul este similar cu scriptul userSetup. Acest modul permite importarea altor module, adăugând în *sys. Path*, şi executarea declaraţiilor. Cu toate acestea, spre deosebire de scriptul userSetup, un *modul sitecustomize* încapsulează atributele lui ca şi orice alt modul, şi astfel, orice modul este importat în el, nu este automat în tabela de simboluri globale pentru *\_\_main\_\_*. Avantajul principal de a folosi un *modul sitecustomize* este acela că el oferă oportunitatea de a seta calea Python fără să interfereze cu scriptul userScript. Prin urmare, se poate implementa un *modul studiowide sitecustomize* de configurare a căilor utilizatorilor care indică spre foldere din reţea şi aşa mai departe. În acelaşi timp, artiştii vor putea să seteze scripturile lor individuale userScript. Utilizatorii vor fi nevoiţi să importe doar un singur pachet pentru aceste tool-uri, şi vor putea să încarce şi alte tool-uri. În această situaţie se poate oferi flexibilitatea configurării mediului de lucru individual, în timp ce, de asemenea, se reduce probabilitatea ca utilizatorii, să modifice accidental modulul de căutare a căilor. *Modulul sitecustomize* este importat înainte de folderele din Maya ce sunt adăugate în "calea Python".
* **setarea unei variabile din mediul PYTHONPATH** - Setând o variabilă de tip mediu permite specificarea folderelor în "calea Python" ce vor fi disponibile înainte că programul Maya să fie iniţializat, astfel se elimină nevoia de a adăuga în lista *sys.path*. Se poate seta o variabilă de tip mediu specifică pentru Maya, sau se poate seta una la nivel de sistem. Oricum, din cauza folderelor adăugate printr-o variabilă de tip mediu, aceasta face parte din "calea Python" înainte că programul Maya să fie iniţializat. De asemenea, se poate pune şi *modulul sitecustomize* într-un folder care este specificat folosind o variabilă de tip mediu.
  + **Variabile de tip mediu specifice Maya cu Maya. Env** - metoda cea mai uşoară de configurare a unei variabile de tip mediu este setarea uneia ce este exclusivă programului Maya. Programul Maya are un fişier, Maya. Env, disponibil într-o locaţie unde se poate modifica, ce îţi permite configurarea variabilelor de tip mediu. Se poate adăuga o variabilă de tip mediu, PYTHONPATH, la acest fişier, şi setând valorile ei la o colecţie de foldere ce se pot include în "calea Python".
  + **Variabile de tip mediu la nivel de sistem şi configurări avansate** - sistemul de operare permite definirea variabilelor de tip mediu care sunt partajate peste toate aplicaţiile. Din cauza setării unei variabile de tip mediu la nivel de sistem, setările din Maya.env se vor rescrie, şi deoarece va fi folosită de toate aplicaţiile, este riscant de folosit, dar de asemenea destul de puternică dacă utilizatorul ştie ceea ce face. Variabilele de tip mediu la nivel de sistem de operare oferă încă o opţiune pentru dezvoltarea codului comun peste un câmp de aplicaţii. O opţiune pentru rezolvarea acestei probleme este modificarea variabilelor de tip mediu specifice aplicaţiilor fiecărui interpretor care să indice către *module sitecustomize* individuale în folderele din reţea, un director ce ţine un *fişier sitecustomize* pentru fiecare versiune a fiecărei aplicaţii. În acest scenariu, fiecare aplicaţie are propria variabila PYTHONPATH indicând spre un director din reţea unde se poate dezvolta uşor şi updata un *modul sitecustomize* ce adaugă noi directoare la calea Python. Fiecare *modul sitecustomize* poate adăuga noi foldere din reţea privitoare la versiunea aplicaţiilor Python şi aşa mai departe. Cu toate acestea, versiunea poate fi incomodă în momentul în care se doreşte adăugarea unor noi softuri în pipeline sau actualizarea softurilor existente. Se poate, de asemenea, seta un *modul sitecustomize* pe un folder din reţea care este specificat printr-o variabilă de tip mediu la nivel de sistem de operare. Să foloseşti o configurare cu variabile de tip mediu la nivel de sistem de operare ar permite de asemenea să se seteze valori pentru MAYA\_SCRIPT\_PATH, care este folosit pentru a localiza scripturi scrise în limbaj de programare MEL. O asemenea configurare va permite să se poată profita de construcţia "calea Python" în scopul folosirii scripturilor golbale de tip userSetup pentru toată compania. Folosindu-se scripturi globale de tip userSetup, se pot importa automat tool-uri şi apoi se poate căuta un script userSetup individual pentru execuţie, de asemenea, ce va permite utilizatorilor o libertate completă pentru scripturile lor userSetup fără a se importa alte tool-uri. În această configurare, *modulele globale sitecustomize* pot folosi atribute în modulul *sys* pentru a afla care aplicaţie o importă cu ajutorul funcţiei *sys.executable* şi care versiune de Python rulează cu ajutorul funcţiei *sys.version*.

## Programarea obiect-orientat în Maya [1]

Limbajul de programare Python poate fi îmbinat cu comenzile din Maya pentru a construcţia unei varietăţi de funcţii şi tool-uri folositoare. Limbajul de programare Python diferă de limbajul de programare MEL, iar îmbunătăţirile nu se pot observa imediat. În limbajul de programare Python există conceptul de modul care este mult mai puternic decât scrierea scripturilor obişnuite în MEL. Motivul pentru care este atât de folositor este pentru că modulele permit încapsularea funcţiilor şi variabilelor numite atribute, în spaţiul de lucru. Un lucru important de reţinut este faptul că modulele sunt obiecte, la fel ca şi orice cod din Python, includ numere întregi, şiruri, liste, dicţionare şi chiar funcţiile. De fapt, Python este un limbaj de programare obiect-orientat şi o mare parte din el este de tip incorporat ce are atribute şi funcţii care se pot accesa mai mult ca în module.

### Programarea obiect-orientată versus programarea procedurală [1]

Programarea orientată pe obiecte înseamnă programarea folosind clase. Pentru a înţelege mai bine ce înseamnă programarea orientată pe obiecte şi care sunt beneficiile acestor clase şi obiecte trebuie scoase în evidenţă diferenţele dintre programarea orientată pe obiecte şi programarea procedurală.

Pe scurt un program înseamnă o operaţie pe un set de date care produce un rezultat. În programarea procedurală, un program este alcătuit din variabile şi funcţii proiectate să opereze pe o structură de date. Un exemplu de conversie a unui string, din litere mici în litere mari, în limbajul de programare MEL folosind programare procedurală este:

string *$numeleTau = "Tom";*

*string $noulNume = `toupper ($numeleTau) `;*

*print ($nouNume)*

*//Printează TOM//*

Acest exemplu ilustrează perfect regula principală a programării procedurale: Datele şi codul care procesează datele sunt ţinute separat. Variabilele *$numeleTau* şi *$noulNume* sunt nişte colecţii simple de date. Procesând datele se solicită funcţii şi procedurii externe, în acest caz, funcţia *toupper ().*

Programarea orientată pe obiecte se raportează la un model de programare care se bazează pe obiecte spre deosebire de programarea procedurală. Pentru înţelegerea acestui principiu, se ia în considerare exemplul de mai sus, convertit în Python, folosind programarea orientată pe obiecte:

n*NumeleTau = 'Tom'*

*noulNume = numeleTau. Upper ()*

*print (noulNume)*

*//Printează TOM//*

Funcţia *upper()* este asociată variabilei *noulNume*. Variabilele în Python sunt obiecte cu valoare, tip şi identitate. Un obiect poate fi considerat ca încapsularea ambelor: date şi funcţionalitatea necesară pentru a acţiona asupra lor. Deci, dacă programarea procedurală este definită drept colecţie de variabile şi funcţionalităţi care operează pe un set de date, un program orientat pe obiecte poate fi definit ca o colecţie de clase care le conţin pe amândouă, date şi funcţionalităţi. Termenul de obiect se adresează apariţiei unei clase unice, o instanţă. Un lucru important este că în interiorul limbajului de programare Python este reprezentat ca un singur obiect.

### Notiuni de baza ale implementării unei clase în Python [1]

Pe lângă obiectele încorporate în Python, utilizatorii pot crea propriile lor obiecte prin definirea claselor. Mecanismul de bază este folosirea cuvântului cheie **class**, urmat de numele clasei. Un exemplu de clasă poate fi:

c*lass NewClass (object):*

*pass*

După ce s-a definit clasa, se poate crea o instanţă a clasei prin chemarea clasei şi returnând valoarea într-o variabilă. Exemplu:

i*nstance1 = NewClass ()*

Această creaţie de apel returnează o instanţă de tip NewClass validă şi unică. Fiecare instanţă este un obiect separat imutabil, numai valoarea care îi stă la bază nu se poate schimba. Ca atare, fiecărei instante îi este garantată o identitate consistentă, de asemenea de-a lungul existenţei sale. Totuşi, folosind programarea orientată pe obiecte, se pot transforma eficient instante prin intermediul atributelor.

**Atributele unei clase**

La fel precum modulele, clasele au atribute. Totuşi, atributele din clase sunt mult mai pronunţate. Un atribut poate fi o variabilă sau o funcţie şi îi poate aparţine unei clase sau unei instante a clasei. Modelul de bază pentru a crea un atribut este definirea sa în definirea clasei, ca şi în exemplul următor:

c*lass NewCalss ():*

*atribut\_data = None*

*def atribut\_prcedura (\*arg):*

*pass*

Python permite adăugarea atributelor unei clase pe parcursul programării, după ce s-a definit clasă. Exemplu:

*newClass. Alt\_atribut = None*

De asemenea se pot adăuga atribute individual la instanţe după ce acestea au fost create, care nu vor afecta alte instanţe ale clasei. Un exemplu:

*instanta1 = NewClass ()*

*instanta1. Instanţa\_atribut = None*

*instanta2 = NewClass ()*

Numai *instanta1* va avea atributul *instanta\_atribut.* Atributele pot fi de asemenea funcţii. Prin urmare, se poate atribui numele oricărui funcţii unui atribut. O funcţie care este un atribut pentru o clasă este prevăzută ca o **metodă**. În timp ce atributele ce sunt adăugate unei clase devin disponibile instanţei imediat, adăugând atribute clasei însuşi, acestea le consideră disponibile imediat la toate instanţele curente şi viitoare a aceleaşi clase. În final, reexecutarea unei definiţii a unei clase nu va afecta instanţele existente ale clasei. Datorită modului în care modelele de date din Python lucrează, instanţele vechi sunt încă asociate unui tip de obiect în altă parte a memoriei.

Atributele unei clase se clasifică în mai multe tipuri:

1. **Atribute de tip date** - se pot adăuga atribute de tip date în multe feluri. Un mecanism comun pentru definirea lor este de a le declara în metoda **\_\_init\_\_()**, ce este chemată când o instanţă este creată prima dată. Deoarece Python caută automat această funcţie de îndată ce o instanţă este creată, atributele ce sunt create în acesta metodă sunt atributele instanţei, şi necesită accesarea lor din instanţă.   
    În timp ce programatorii din alte limbaje de programare sunt tentaţi să considere *\_\_init\_\_()* ca fiind o funcţie constructor, este de fapt un pic diferită de ea. Deşi lista ei de parametri prezintă cerinţele sintaxei pentru instanţiere, instanţa a fost deja construită în timpul apelării *\_\_init\_\_().* Parametri specificaţi pentru *\_\_init\_\_()* restricţionează construirea tuturor cerinţelor sintaxei.

Numele *self* este folosit ca o convenţie, şi se referă la instanţa obiect chemând această funcţie. Acest argument este întotdeauna folosit în metodele obiectelor. Prin urmare, toate metodele obişnuite vor lua instanţa deţinută ca şi prim argument.

1. **Methoda** - este o funcţie existentă într-o clasă. De obicei primul argument este întotdeauna instanţa însuşi. Metodele sunt de asemenea un tip de atribute. De exemplu, în timp ce **len ()** este o simplă funcţie incorporată ce se poate folosi cu şiruri, **upper ()** este o metodă ce este accesibilă pe obiecte de tip string. Se pot adăuga metode claselor create de utilizatori cu condiţia că primul argument va fi instanţa clasei (*self*).

Metodele sunt una dintre tool-urile primare ce diferenţiază programarea obiect-orientata de cea procedurală. Ele permit dezvoltatorilor să primească şi să modifice datele unei instanţe în moduri sofisticate.

1. **Atributele clasei** - pot fi accesate folosind numele clasei şi nu cer să fie creată o instanţă. Atributele clasei pot fi folositoare pentru adăugarea datelor sau metodelor unei clase ce pot fi folosite de alte funcţii sau clase în timp ce nu este necesară crearea unei noi instanţe a clasei. Se pot accesa folosind operatorul punct cu numele clasei. Comportamentul atributelor clasei este la fel ca şi cel al atributelor instanţelor.   
   Uneori atributele clasei pot părea nefolositoare, deoarece ele se pot fi definite ca atribute în afara clasei. Ele se folosesc mai mult în momentul când sunt create mai multe clase într-un singur modul şi vream să ţinem informaţia cât mai bine partajată. Ba mai mult, numele variabilelor pot intra în conflict în momentul în care vrem să aveam acelaşi nume cu 2 valori.
2. **Metodele statice şi metodele clasei -** permit apelarea unei metode asociate cu clasa fără a solicita o instanţă a clasei. Aceste tipuri de metode se pot defini folosind *@staticmethod* şi *@classmethod* deasupra funcţiilor definite. Singura diferenţa dintre aceste două opţiuni este faptul că metoda implicită îşi trece clasa însuşi ca prim argument (*self)* ce permite să accesezi orice atribut din clasă, exemplu *instanţă. Numele\_atributului* mai degrabă decât *NumeleClasei.numele\_atributului*. Metodele clasei pot fi folosite pentru a implementa metode de creare alternative pentru o clasă sau în alte situaţii unde accesul la datele clasei sunt cerute separat de instanţă.

### Moştenirea [1]

Alt concept important în programarea obiect-orientata este moştenirea. La un nivel înalt, moştenirea permite unei noi clase să fie derivată dintr-o clasă părinte. Automat nouă clasă poate accesa toate atributele clasei părinte. De asemenea nouă clasă poate rescrie funcţionalităţile de care avem nevoie în clasa descendentă. Decât să copiem metodele unei clase intr-o ata clasa, mai degraba folosim mostenirea. Python poate crea o nouă clasă bazată pe clasă existentă ce prezervă toate funcţionalităţile clasei originale, în timp ce permite noi funcţionalităţi să fie introduse. Un exemplu de moştenire este:

c*lass Vehicul (object):*

*def \_\_init\_\_ (self):*

*self. Viteza = 100*

*class Maşina (Vehicul):*

*def \_\_init\_\_ (self):*

*Vehicule. \_\_init\_\_ (self)*

*Self. Greutate = 1000*

*BMV = Maşină ()*

*print BMV.Viteză*

Sunt două puncte importante legate de derivarea clasei:

* Primul este legat de declararea clasei: *class Maşină (Vehicul).* În locul obiectului incorporat *object*, clasa *Maşină* foloseşte clasa *Vehicul* ca şi părinte sau ca şi clasă de bază. Această sintaxă este primul pas ce asigură faptul că clasa copil reţine funcţionalităţile implementate în *Vehicul.*
* Al doilea punct este iniţializarea clasei *Maşină.* Această situaţie este una dintre puţinele instanţe unde metoda clasei *\_\_init\_\_()* trebuie să fie chemata direct. Dacă iniţializatul clasei de bază nu este apelat, funcţionalităţile clasei derivate pot fi imprevizibile. De exemplu, o clasă copil va moşteni metodele clasei părinte. Dacă aceste metode nu sunt specific implementate în clasa copil, unele dintre ele pot cere prezenţa datei unui atribut unei instanţe ce este creat în metoda clasei *\_\_init\_\_()*, mai degrabă decât în definiţia clasei de bază.

Un alt avantaj al moştenirii este faptul că se poate crea uşor altă clasă derivată cu propriile ei funcţionalităţi, care poate fi complet neimportantă pentru alte clase. Derivarea claselor este o cale puternică încă flexibilă pentru a refolosi codul şi a ţine organizat bazele codului.

### PyMEL [13]

O deficienţă a modulului *maya.cmds* identificată de programatori este faptul că *maya.cmds* nu prezintă programare obiect-orientată propriu-zis cum s-a sperat. Din fericire, câţiva programatori au creat un set de tool-uri numite PyMEL, care încercă să abordeze această problemă. În *pachetul pymel* există câteva funcţii ce acţionează ca şi cele din *maya.cmds*. Este un avantaj folosirea acestor funcţii deoarece ele sunt create pe baza modelului de programare orientată pe obiect. Pentru a lucra cu aceste funcţii, se instalează un pachet şi se importă în spaţiul de lucru la fel ca şi *maya.cmds*: *import pymel.core as pm.* Următorul exemplu clarifică diferenţa dintre *pyMEL* şi *maya.cmds*:

*# folosind PyMEL*

s*cene\_nodes = pm.ls (type='transform')*

*print scene\_nodes*

*#Afişează: nt. Transform (u'front'), nt. Transform (u'persp'), nt. Transform (u'side'), nt. Transform (u'top')*

*# folosind maya.cmds*

*scene\_nodes = maya.cmds.ls (type='transform')*

*print scene\_nodes*

*[u'front', u'persp', u'side', u'top']*

În timp ce comanda *ls* în modulul *maya.cmds* returnează o listă de şiruri Unicode, apelarea modulului *PyMEL* returnează o listă a unui tip de clasă custom, ce încapsulează date legate de nodurile din Maya.

*print (type (scene\_nodes [0]))*

*# Afişează**: <class 'pymel. Core. Nodetypes. Transform'>*

Aceste este un aspect al PyMEL, numit clasă *PyNode.* Toate comenzile în modulul *pymel. Core* ce imită sintaxa *maya.cmds* returnează obiecte *PyNode* în loc de şiruri. Motivul pentru acesta abordare este acela de a accesa comenzile în modulul *pymel. Core*, chiar dacă acestea sunt echivalent sintactic în *maya.cmds*, nu sunt uşor împachetate. Multe dintre acestea sunt derivate din importări sau manual împachetate. Aceste tehnici permit: funcţionalitate garantată; integrare API; accesare uşoară; recrearea comenzilor voluminoase în comenzi cu mai puţin cod; şi chiar şi crearea de funcţii unice în PyMEL.

Obiectul **PyNode** este o o clasă împachetată pentru un obiect API ce stă la baza. Fiecare tip de nod în scenă este reprezentat în Maya API, de aceea clasă *PyNode* este disponibilă să interacţioneze uşor cu multe metode din API. Această abordare le asigură programatorilor cu câteva funcţionalităţi interesante:

* codul poate fi scri mult mai " pythonically", precum atributele din *PyNode* sunt accesate prin intermediul atributelor referinţelor.
* obiectele *PyNode* încorporează metodele împachetate API ce pot câteodată fi mult mai rapide decât cele din *maya.cmds.*
* deoarece obiectele din *PyNode* au identităţi unice, este uşoară folosirea lor în cod şi mult mai sigure decât atunci când lucrăm cu numele obiectelor.
* operaţiile bazate pe API tind să fie mult mai rapide decât operaţiile de procesare a şirurilor.

**Avantaje şi dezavantaje**

*PyMel* oferă câteva avantaje în comparaţie cu *maya.cmds*, care îl pot face o investiţie utilă pentru proiectele de orice dimensiune. Unele dintre aceste avantaje sunt:

* programatorii experimentaţi în limbajul de programare Python pot învăţa într-un timp mult mai scurt *PyMEL* datorită faptului că este obiect-orientat.
* sintaxa *PyMEL* tinde să fie un pic mai clară şi codul creat un pic mai organizat.
* viteza de executare este garantată în unele cazuri datorită hibridizării API.
* pachetele *pymel* sunt "open source" ceea ce înseamnă faptul că utilizatorii le pot modifica după nevoile lor şi le pot repara în cazul în care nu funcţionează corect.

În schimb *PyMEL* are şi câteva dezavantaje în comparaţie cu *maya.cmds*:

* programatorii din MEL au foarte mult de învăţat despre programarea orientată pe obiect ceea ce poate duce la descurajare, şi la o perioadă în care producţia poate fi blocată.
* comunitatea *PyMEL* este foarte mică ceea ce înseamnă găsirea rezolvării problemelor poate fi dificilă.
* în unele cazuri, viteza de executare poate scădeaş procesarea unui număr mare de componente, pentru instanţe, poate fi mult mai înceată utilizând *PyMEL*, depinzând de operaţia specifică.
* dezvoltatorii softului Autodesk Maya nu au implementat acest modul ca modul oficial deoarece acesta reprezintă neclarităţi.
* deoarece pachetele *pymel* sunt "open source", este posibilă situaţia în care un fişier să fie diferit de la o versiune la alta.

# Capitolul 3. Plugin - "Creare Mediu". Descriere şi funcţionalitate

În capitolele 1 şi 2 s-a făcut cunoscut ce reprezintă CGI (Computer generated imagery), ce putem realiza cu aplicaţia Autodesk Maya, ce înseamnă cuvântul Plugin în sine, cum este introdus, cum interacţionează şi cum se execută limbajul de programare Python în software Autodesk Maya. În plus, s-a mai discutat despre diferenţele principale dintre limbajul de programare Python şi alte limbaje de programare, despre programarea obiect-orientată în Maya, diferenţele dintre programarea obiect-orientată şi cea procedurală, despre cum se creează un modul şi importanţa căii în Python. Totodată, am reliefat multe despre noţiunile de bază ale implementării unei clase în Python, despre moştenire şi despre setul de tool-uri PyMEL care a fost creat pentru a accesa clase încorporate în software Maya.

În acest capitol se va prezenta funcţionalitatea unui Plugin care trece prin toate punctele de teorie prezentate în capitolele anterioare.

## Descrierea şi funcţionalitatea Plug-inului "Creare Mediu"

**Mediu**

Codul sursă al Plugin-ului "Creare mediu" este scris în limbajul de programare Python folosind Eclipse Clasic şi Script Editor din softul Autodesk Maya. Pentru crearea interfeţei s-a folosit softul PyQt GPL, iar pentru executarea şi afişarea rezultatului s-a folosit softul Autodesk Maya.

**Scop**

Scopul acestui Plugin este de a crea un mediu înconjurător (pământ, iarbă, flori, pietre şi copaci) automat într-un timp foarte scurt. Se introduc nişte date de către utilizatorul plugin-ului în câmpurile obligatorii, şi la apăsarea butonului de creare se generează automat mediul dorit.

**Aplicabilitate**

Plugin-ul este foarte uşor de pus în aplicare, deoarece el este încapsulat în clase, clasele sunt scrise în module iar modulele sunt adăugate într-un pachet. Pentru a folosi acest script este necesar copierea unui folder într-un loc unde softul Autodesk Maya să îl poată accesa pentru a importa modulele, importarea în sistemul de operare a căii către pachetul Plughin-ului şi importarea pachetului în spaţiul de lucru.

Pentru a rula acest plugin este necesar să aveţi instalat pe calculatorul respectiv softul Autodesk Maya 8 (sau orice versiune mai nouă), softul Python 2.7 (sau orice versiune mai nouă de Python 2) şi softul PyQt GPL pentru Python 2.7 (sau pentru versiunea de Python instalată).

## Descrierea instrucţiunilor de bază

Ţinând interfaţa programului separată de funcţionalitatea Plugin-ului, interfaţa se poate, mai târziu, mult mai uşor modifica. Prin această metodă se poate refolosi mai târziu părţi ale codului mult mai uşor în alte softuri cum ar fi folosirea interfeţei pentru a crea alte Plugin-uri, sau folosirea unor clase ce pot fi accesate de mai multe Plugin-uri în acelaşi timp. Acesta este unul dintre avantajele de bază ce ajută la organizarea şi întreţinerea scripturilor, şi unul din motivele principale pentru care a fost integrat limbajul de programare Python în Autodesk Maya. Una dintre cele mai importante clase este clasa *Mesaje* ce este accesată de toate Plugin-urile existente.

Plugin-ul "Creare Mediu" are o structură bine organizată. Pachetul *PluginCreareMediu* deţine următoarele module:

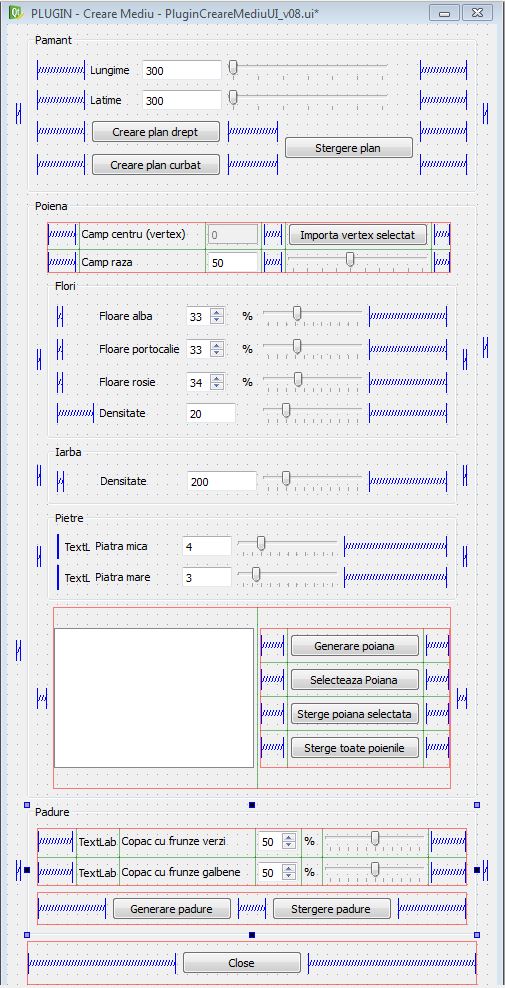
**\_\_init\_\_.py** - modulul ce indică faptul că acest folder este un pachet în limbajul de programare Python. În acest modul se introduc numele fişierelor ce se importă automat în momentul importări pachetului.

**InterfataPlugin.py** - acest modul deţine o clasă numită *InterfataPluginClass ()* ce încarcă interfaţa Plugin-ului creată în PyQt (PluginCreareMediuUI.Ui, Figura 3.1) şi conectează toate obiectele (butoane, câmpuri, liste, etc) de alte clase prin intermediul unor metode implementate. În acesta clasă sunt importate doar câteva funcţii din modulul PyQt4, funcţii din sistemul de operare şi clasa PluginClass. Un lucru important este faptul că nu avem importat niciun modul dintr-un anumit soft, deci această clasă se poate folosi, fără a fi

modificată, în orice soft ce permite ca limbaj de programare Python.

Metodele de bază din această clasă sunt:

1. *loadImage (labelPath, labelName)* - această metodă încarcă o imagine în interfaţa creată cu PyQt; se dau că parametrii calea către imagine de pe hard disc şi numele câmpului în care se va încărca imaginea;
2. *createIrregularPlan () -* această metodă apelează o altă metodă din clasa *PluginClass ()* pentru a genera planul pământului neuniform.
3. *createUniformPlan ()* - acesta metodă este aproape identică cu metoda.
4. *createIrregularPlan () -* diferenţa făcându-o parametrii daţi metodei din clasa *PluginClass ().*
5. *deletePlan ()* - această metodă şterge planul pământului.
6. *importSelectedVertex ()* - această metodă memorează valoare returnată de către metoda.
7. *selectedVertex () -* din clasa *PluginClass ()* şi o importă în câmpul *centruFieldSpinBox.*
8. *generateMeadow ()* - în acesta metodă se verifică mai întâi procentul pentru culoarea florilor introdus de către utilizator, iar dacă acesta nu este corect se recalculează, apoi se apelează metoda *generateMeadow ()* din clasa *PluginClass ()* cu toţi parametrii necesari (raza, procentele florilor, densitatea florilor, a ierbii, a pietrei mari şi a pietrei mici).
9. *selectMeadow ()* - această metodă selectează grupul ce este selectat în listă.
10. *deleteSelectedMeadow ()* - această metodă şterge grupul ce este selectat în listă.
11. *deleteAllMeadows ()* - această metodă şterge toate poienile existente.
12. *generateForest ()* - această metodă mai întâi verifică procentul introdus de către utilizator şi apoi generează pădurea apelând o metodă din clasa *PluginClass ().*
13. *deleteForest ()* - această metodă şterge pădurea creată.

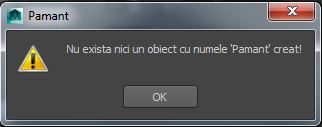
**  
Figura 3.1 - Interfaţa Plugin "Creare Mediu" în PyQt4**

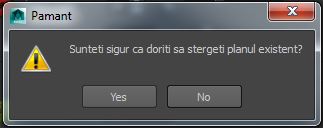
(Sursa: imaginea autorului realizată cu ajutorul programului

PyQt4 GPL v4 pentru Python v2.7)

**Plugin.py -** acest modul este modulul principal ce deţine o clasă numită *PluginClass()* unde sunt importate toate celelalte module care crează mediul. La fel ca şi în modulul *InterfataPlugin.py*, nu se importă nicio metodă dintr-un modul ce ţine de softul respectiv. Prin urmare acest modul se poate folosi şi de către alte softuri în care limbajul de programare Python este integrat fără a se face nicio modificare în cod.

**UserErrorMessages.py** - acest modul deţine o clasă numită *UserErrorMessage()* în care avem definite două metode principale: *showWindowMessage()* şi *showYesNoMessage()*. Acesta clasă importă doar câteva funcţii din modulul *PyQt4*şi este folosită aproape de către toate Plugin-urile create. Metoda *showWindowMessage()* este folosită pentru a înştiinţa utilizatorul printr-un mesaj de eroare sau avertizare despre progresul Plugin-ului respectiv. Un exemplu de astfel de fereastră reprezintă Figura 3.2. Iar metoda *showYesNoMessage()* este utilizată în momentul în care utilizatorul doreşte să facă modificări majore asupra ceea ce softul a generat, programul explicându-i într-un mod mai detaliat şi cerându-i o ultimă confirmare. O astfel de fereastră reprezintă Figura 3.3. Aceste mesaje sunt afişate într-o fereastră nou creată. Până în momentul în care utilizatorul nu confirmă citirea mesajului prin apăsarea butoanelor disponibile, programul nu permite continuarea utilizării softului.

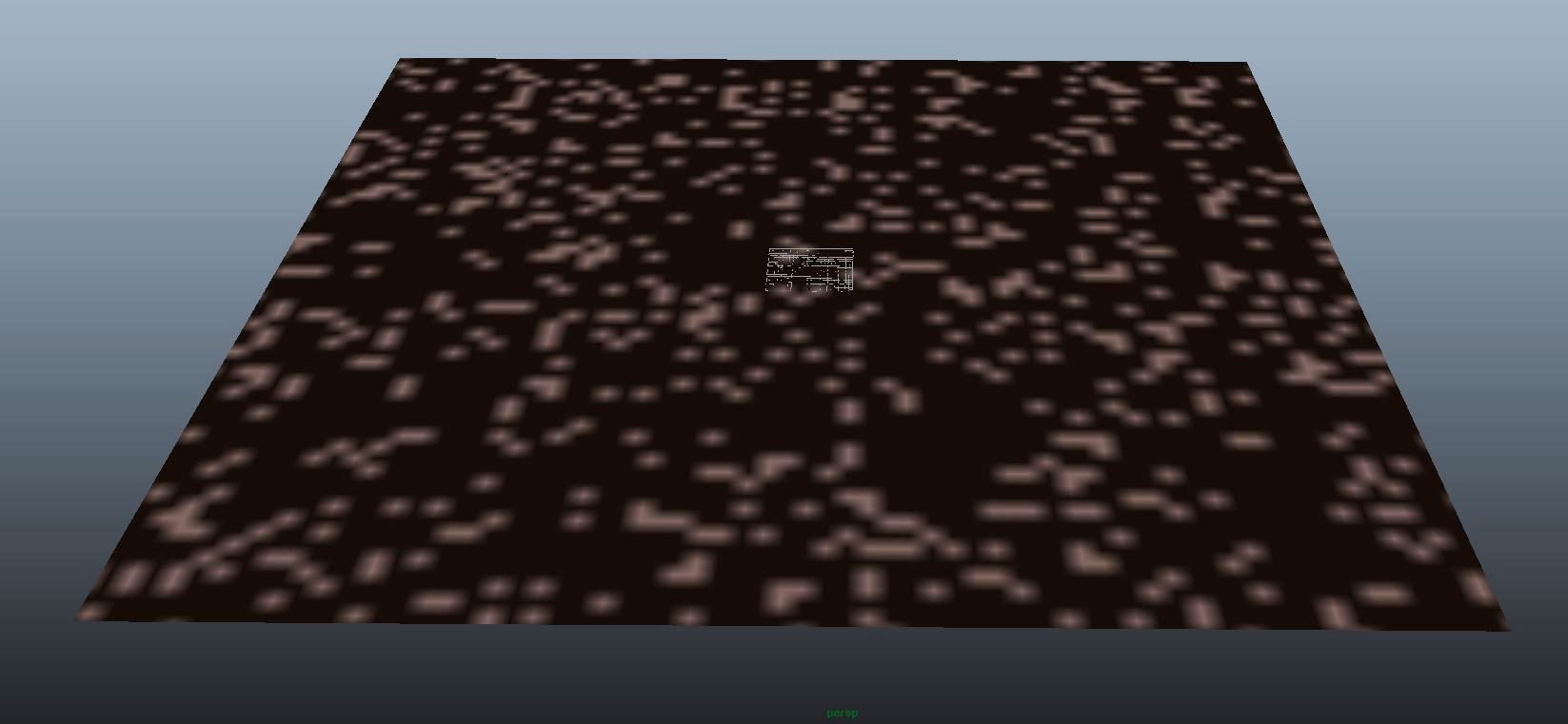
  
**Figura 3.2 - UserErrorMessages - *showWindowMessage()***

(Sursa: imaginea autorului realizată cu programul Autodesk Maya) **  
 Figura 3.3 - UserErrorMessages - *showYesNoMessage()***

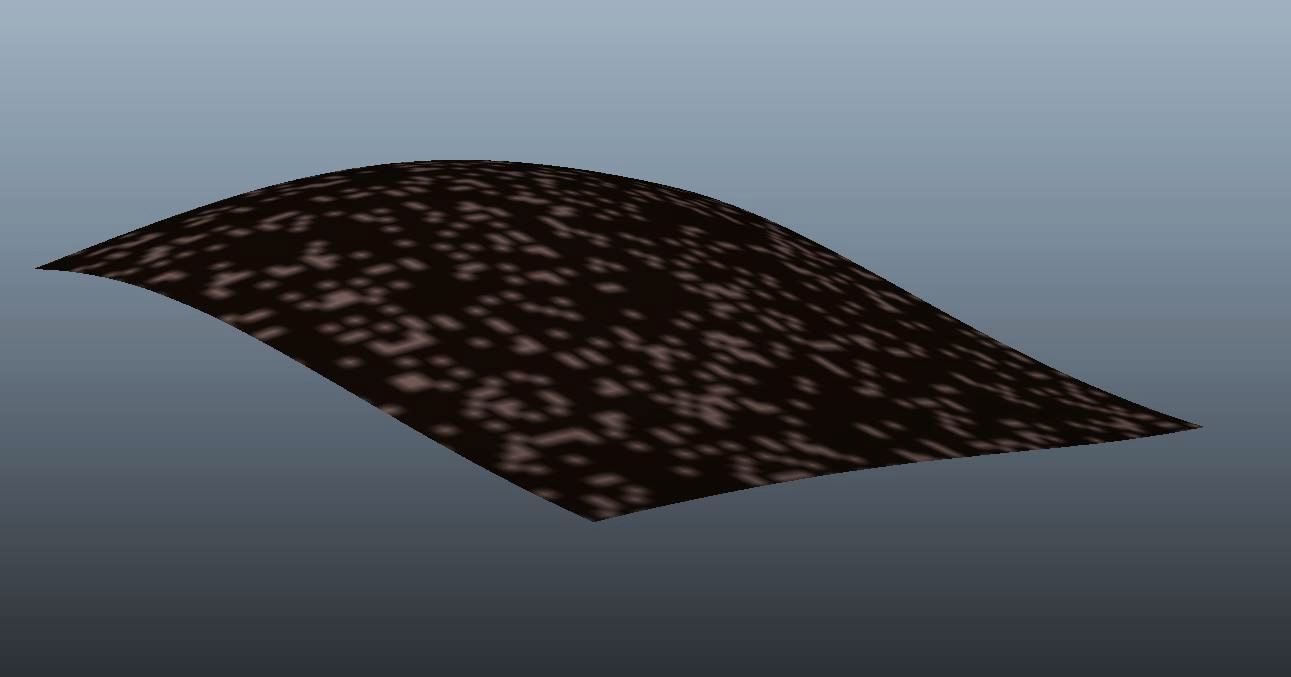
(Sursa: imaginea autorului realizată cu programul Autodesk Maya)

**MayaUserErrorMessages.py -** acest modul deţine o clasă numită *MayaUserErrorMessage()* în care avem definite două metodele principale *showWarning()*  şi *showError()*. Aceste metode sunt folosite pentru a personaliza erorile afişate de către soft, modificându-le într-o maniera în care orice utilizator le poate "digera". Aceste erori nu sunt afişate ca şi în clasa *UserErrorMessage()* printr-o fereastră nou creată, ci sunt afişate în text editor a softului respectiv. Acesta clasă importă funcţii dintr-un soft specific şi nu poate fi folosit în mai multe softuri fără a fi modificat codul.

**Ground.py** - acest modul deţine o clasă numită *GroundClass()*în care avem metodele de generare a unui plan pentru pământ. Metodele principale sunt *createPlan()*, în care se creează planul propriu zis, şi *makePlanIrregular()* în care se aleg random vertex-uri din plan şi li se modifică coordonata Y. Această clasă este specifică softului Autodesk Maya. Figura 3.4 reprezintă generare unui plan al pământului drept iar Figura 3.5 reprezintă generarea unui plan al pământului curbat.



**Figura 3.4 - Generarea unui plan drept al pământului**(Sursa: imaginea autorului realizată cu programul Autodesk Maya)

**Figura 3.4 - Generarea unui plan curbat al pământului**(Sursa: imaginea autorului realizată cu programul Autodesk Maya)

**Flower.py** - acest modul deţine o clasă numită *FlowerClass()* în care avem mai multe metode:

1. *crearePetals()* - această metodă crează petalele florii;
2. *createCore() -* acesta metodă crează corul florii;
3. *rotateHeadFlower()* - această metodă roteşte capul florii random pentru că imaginea generată să pară cât mai reală;
4. *createStalk()* - acesta metodă crează tulpina florii; la fel ca şi capul florii, el este îndoit şi dimensionat în mod random pentru a da o iluzie mai bună.
5. *flowerPosition()* **-** această metodă poziţionează floare pe nivel 0, înainte de a fi mutată în locul dorit.

Această clasă este specifică softului Autodesk Maya şi nu poate fi folosită în alt soft fără ca, codul să fie modificat.

În Figura 3.5 sunt reprezentate cele 3 tipuri de flori.

****

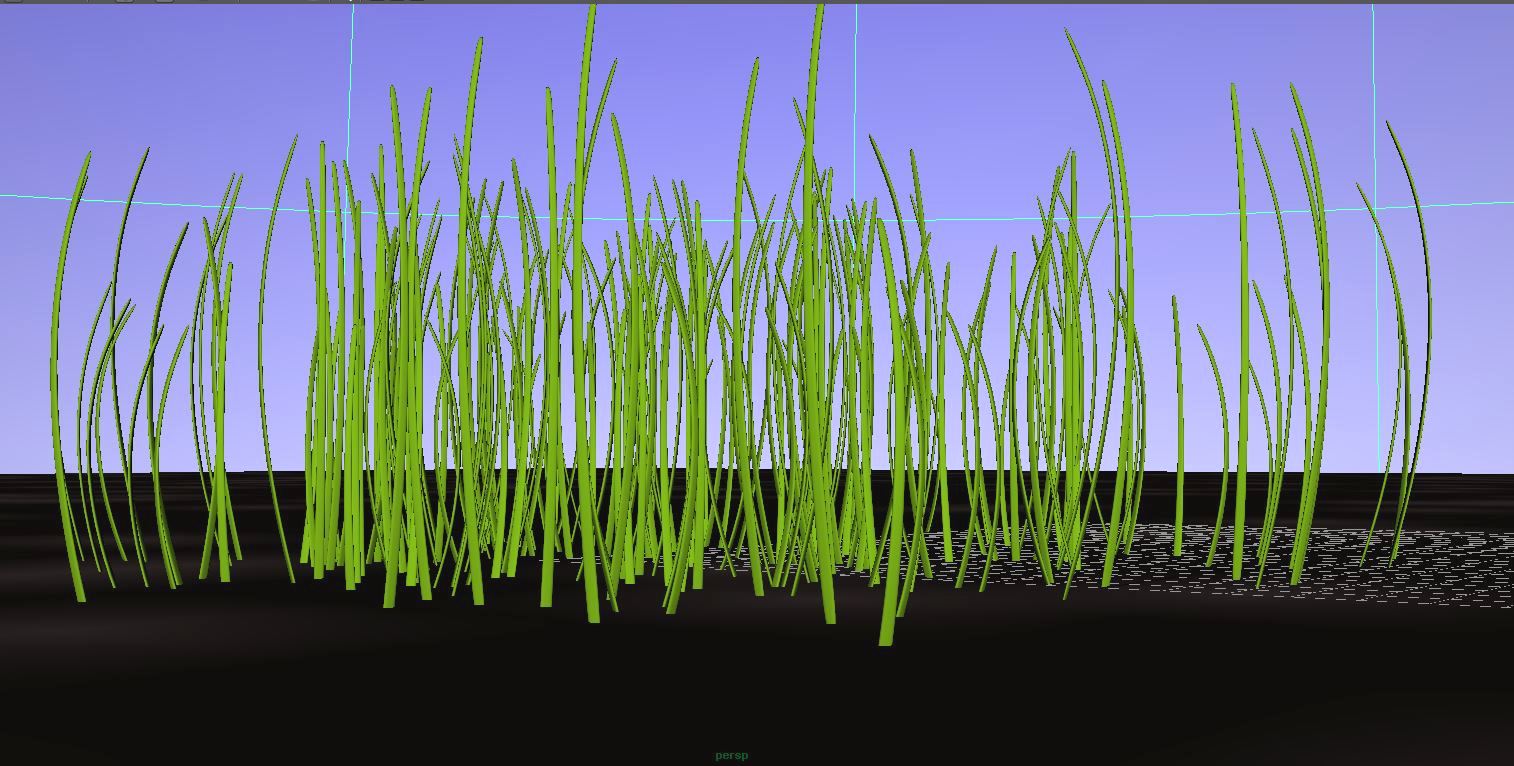
**Figura 3.5 - Generarea celor 3 tipuri de flori**

(Sursa: imaginea autorului realizată cu programul Autodesk Maya)

**Grass.py** - acest modul deţine o clasă numita *GrassClass()* în care sunt definite următoarele metode:

1. *createBladeOfGrass()* - această metodă creează un fir de iarbă; acesta este îndoit şi dimensionat random pentru a fi personalizat;
2. *changePositionBlade()* - această metodă muta firul de iarbă pe poziţia 0 şi îl roteşte;
3. *createBunchOfGrass()* - această metodă multiplică firul de iarbă;
4. *changePositionBunch() -* această metodă mută fiecare fir de iarbă multiplicat random pe o anumită rază;

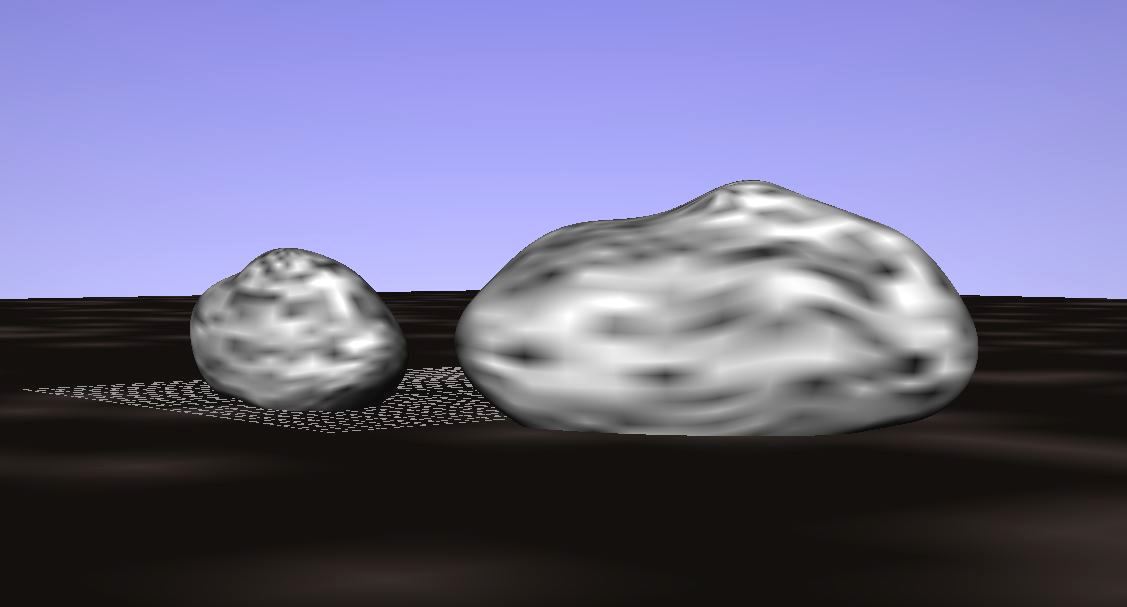
Această clasă este specifică softului Autodesk Maya şi nu poate fi folosită în alt soft fără ca, codul să fie modificat. În Figura 3.5 sunt reprezentate nişte fire de iarbă ce au fost generate cu ajutorul Plugin-ului.

****

**Figura 3.5 - Generarea unor fire de iarbă**

(Sursa: imaginea autorului realizată cu programul Autodesk Maya)

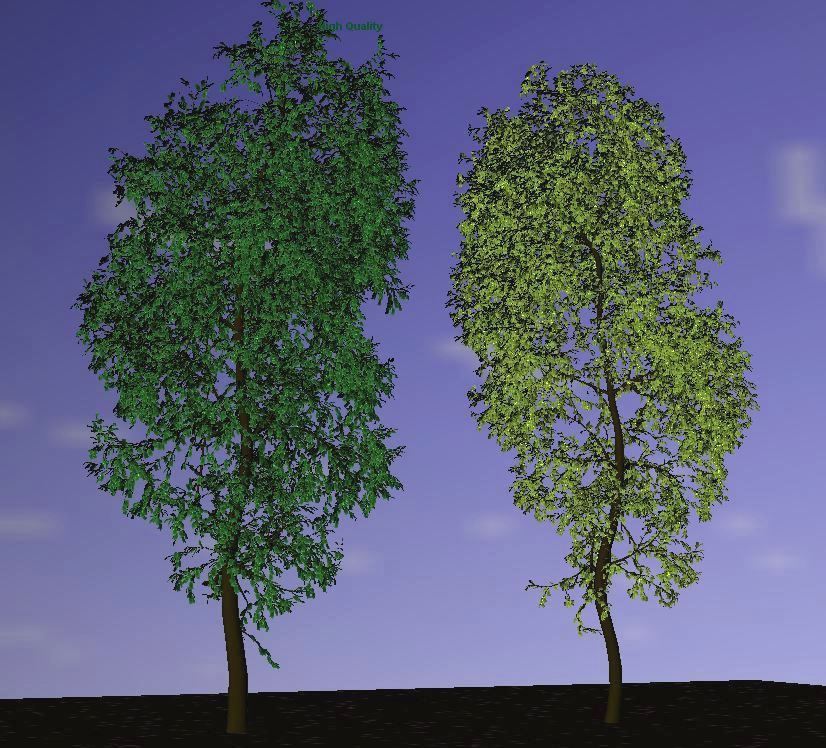
**Stone.py -** acest modul deţine o clasă numită *StoneClass()* ce crează nişte sfere, iar prin intermediul modificării lor random dau forma unor bolovani. Acesta clasă importă modulul *maya.cmds*, *random* şi *maya.mel***.** Modulul *maya.mel* este folosit pentru a putea executa funcţii predefinite din limbajul de programare MEL în limbajul de programare Python. În Figura 3.6 sunt reprezentate două pietre, una mică şi una mare, generate cu ajutorul scriptului.

****

**Figura 3.6 - Generarea unei pietre mici si unei pietre mari**

(Sursa: imaginea autorului realizată cu programul Autodesk Maya)

**Tree.py** - acest modul deţine o clasă numită *TreeClass()* în care sunt definite două funcţii *importTree()* şi *changePosition()*. Prima metodă *importTree()* importă dintr-o altă scenă creată cu Autodesk Maya un copac în scena curentă în care se utilizează scriptul. A doua metodă, *changePosition(),* modifică copacul importat, rotindu-l şi scanându-l pentru a crea o imagine cât mai apropiată de realitate. În Figura 3.7 sunt reprezentate cele două tipuri de copaci.

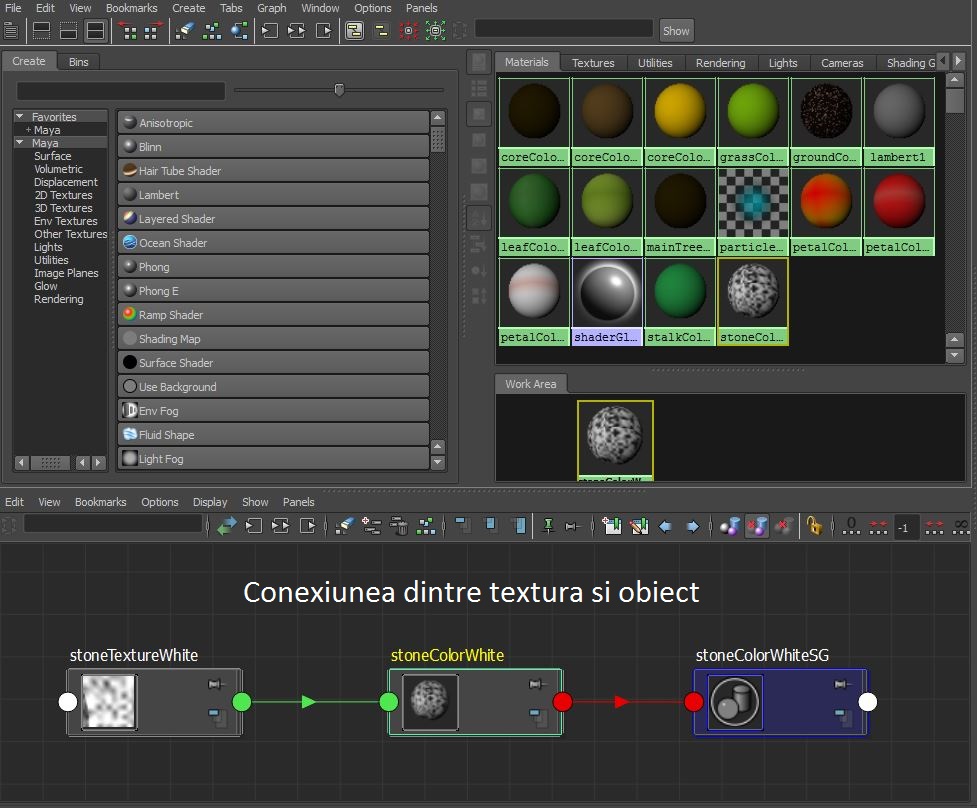
**Figura 3.7 - Generarea unui copac verde si unui copac galben**

(Sursa: imaginea autorului realizată cu programul Autodesk Maya)

**Shader.py** - acest modul deţine o clasă numită *ShaderClass()* în care sunt definite funcţii ce generează şi conectează mai multe tipuri de shadere. Pe scurt, un "shader" în softul Autodesk Maya este un material ce se aplică asupra unei suprafeţe. Acestui material i se poate schimba culoarea, transparenţa, incandescenţa, "ambient color", i se poate adăuga orice tip de textură, iar prin adăugarea de textură 3D poate lua forma oricărui obiect din viaţa reală cum ar fi: apă, lemn, piatră, zăpadă, piele, păr şi multe alte forme. Cu ajutorul texturii nu este necesară modelarea obiectelor în cele mai profunde detalii.

În Plugin-ul "Creare Mediu" s-au folosit shadere normale pentru a colora florile, iarbă, frunzele copacilor, s-au folosit texturi de tip 2D pentru forma pietrelor şi cea a copacilor şi textura 3D pentru pământ.

În Figura 3.8 sunt reprezentate materialele create pentru Plugin-ul "Creare Mediu". De asemenea în partea de jos este reprezentată conexiunea dintre un material şi un obiect.

**Figura 3.8 - Reprezentarea materialelor create pentru Plugin-ul "Creare Mediu"**(Sursa: imaginea autorului realizată cu programul Autodesk Maya)

**Position.py** - acest modul deţine o clasă numită *PositionClass()* ce moşteneşte clasa *GroundClass()* din modulul *Ground.py*. Clasa *PositionClass()* este derivată din clasa părinte *GroundClass()* deoarece clasa *PositionClass()* accesează atributele clasei părinte şi mai degrabă decât să avem o copie sau o reimplementare a clasei originale, nouă clasă este bazată pe clasă existentă ce prezervă toate funcţionalităţile clasei părinte.

În acest modul sunt create şi variabile globale. Aceste variabile sunt necesare deoarece, în momentul în care Plugin-ul este închis (softul rămâne în permanenţă deschis), dacă variabilele nu sunt globale, datele se pot pierde la redeschiderea Plugin-ului, deoarece se crează alte obiecte.

Pe scurt poziţia se calculează în funcţie de planul pământului. Planul este creat din puncte, laturi şi suprafeţe. În Figura 3.10 sunt reprezentate punctele unui plan de dimensiune 10x10. Fiecare punct deţine într-o listă 3 numere (reprezentarea grafică 3D). În Figura 3.11 sunt reprezentate coordonatele (x,y) fiecărui punct din plan.

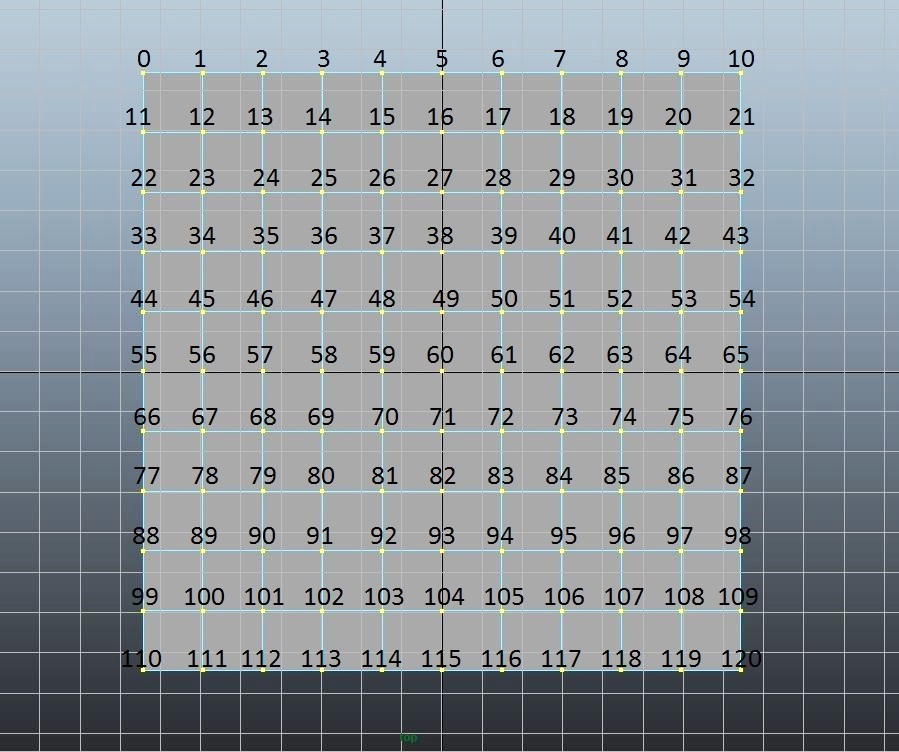
Pentru o înţelegere mai bună a modului cum se calculează poziţia fiecărui obiect din spaţiul de lucru se vor prezenta metodele din clasa *PositionClass()*:

1. *findNumberSelectedVertex()* - această metodă verifică dacă s-a selectat măcar un vertex în plan, iar dacă nu se returnează un mesaj de eroare.
2. *initializeVectors()* - această metodă iniţializează vectorii ce reţin poziţia punctelor în planul 3D şi vectorul de disponibilitate.
3. *findVertexPosition()* - această metodă memorează poziţia fiecărui punct în plan în vectorii de poziţie.
4. *calculateDistance(objectTX, objectTZ)* - acesta metodă calculează distanţa dintre 2 puncte. Formula distanţei este: distanţa .
5. *calculateXZ(radius)* - această metodă returnează două valori din cercul cu raza dată.
6. *meadowPosition(radius, object, numberGenerateMedow)* - această metodă calculează poziţia fiecărui obiect din câmpie şi mută obiectul pe poziţia respectivă. În Figura 3.9 este reprezentată o poiană. După cum se observă poiana are forma unui cerc datorită faptului că s-au generat obiecte numai pe poziţiile în care distante dintre puncte a fost mai mică decât raza cercului.
7. *treePosition(position, object)* - această metodă calculează poziţia fiecărui copac din pădurea creată.
8. *checkSuperimpose(object, i, numberGenerateMedow)* - această metodă verifică dacă punctele din jurul unui punct sunt disponibile pentru a putea muta un obiect în zona respectivă.
9. *makeUnAvailablePositionMeadow(radius, availableNumber)* - această metodă schimbă disponibilitatea punctelor din cercul în care s-au creat obiectele poienii dacă pe respectivele punţi nu s-a creat nimic.



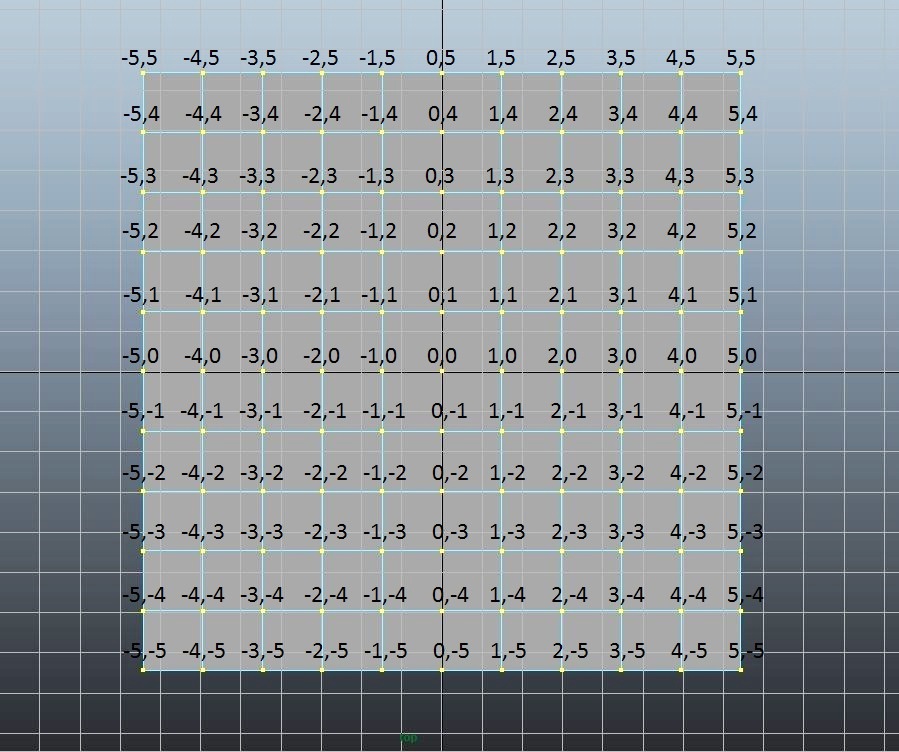
**Figura 3.9 - Reprezentare unei poieni**

(Sursa: imaginea autorului realizată cu programul Autodesk Maya)



**Figura 3.10 - Reprezentare grafică a punctelor unui plan 11x11**

(Sursa: imaginea autorului realizată cu programul Autodesk Maya)



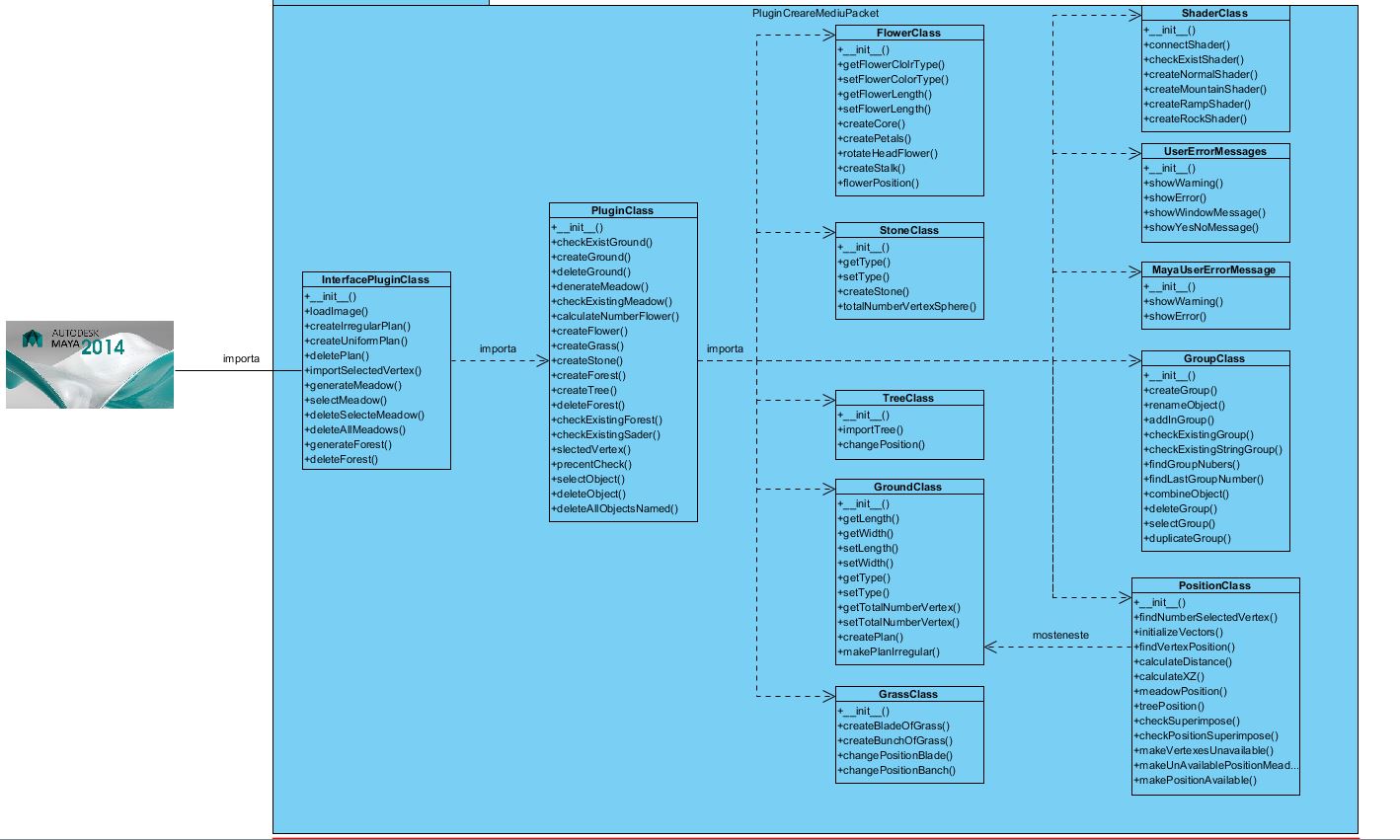
**Figura 3.11 - Reprezentare grafică a distanţelor punctelor unui plan 11x11 pe axele (x,y)**

(Sursa: imaginea autorului realizată cu programul Autodesk Maya)

**GroupPanelLayout.py -** În softul Autodesk Maya, ca şi în celelalte softuri, redenumirea şi gruparea obiectelor este foarte importantă! De aceea acest modul deţine o clasă numită *GroupClass()* cu următoarele metode:

1. *createGroup(groupName)* - această metodă crează un nou grup cu numele dat ca parametru;
2. *renameObject(object, objectName)* - această metodă redenumeşte un obiect sau un grup dat ca parametru cu noul nume dat ca parametrii;
3. *addInGroup(groupName, object)* - această metodă adaugă un obiect într-un grup, ambele, şi numele obiectului şi cel al grupului sunt dată ca parametru;
4. *checkExistingGroup(groupName)* - această metodă verifică dacă există deja vreun grup creat cu numele dat ca parametru;
5. *checkExistingStringGroup(stringName) -* această funcţie verifică dacă există vreun grup ce conţine în numele sau stringul dat ca parametru;
6. *findGroupNumbers(groupName)* - această metodă returnează numerele grupurilor existente ce conţin un anumit şir de personaje în numele grupului. Exemplu: avem create următoarele grupuri: *Poiana\_1*, *Poiana\_2*, *Poiana\_5*, *Poiana\_7*; se va returna o listă de formă: [1, 2, 5, 7];
7. *findLastGroupNumber(groupName)* - acesta metodă returnează numărul grupului cu numărul cel mai mare; Exemplu: avem grupurile: *Poiana\_2*, *Poiana\_5* şi *Poiana\_9*. Funcţia va returna 9.
8. *combineObjects(listOfObjects)* - această metodă ia ca parametru o listă de obiecte şi le combină, creând un singur obiect;
9. *deleteGroup(nameGroup)* - această metodă şterge grupul cu numele dat ca parametru;
10. *selectGroup(groupName)* - această metodă selectează în spaţiul de lucru grupul cu numele dat ca parametru;
11. *duplicateGroup(groupName)* - această metodă crează o dublură a grupului dat ca parametru.

Structura claselor este de formă: Autodesk Maya importă *InterfacePluginClass(), InterfacePluginClass()* importă *PluginClass()*, iar *PluginClass()* importă restul claselor. Un alt lucru important în structura claselor este faptul că *PositionClass()* moşteneşte *GroupdClass().* Pentru o înţelegere mai bună, în Figura 3.12 este reprezentată structura claselor.

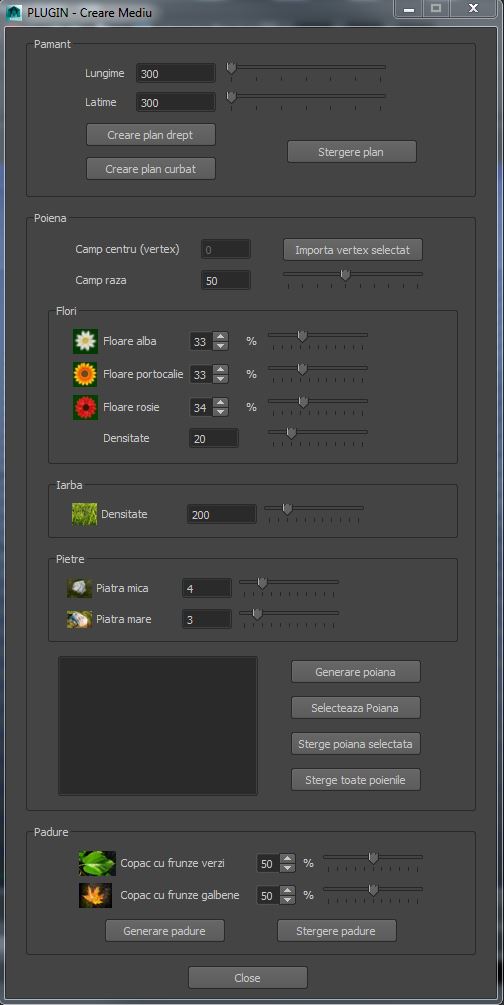


**3.12 - Structura claselor**

(Sursa: imaginea autorului realizată cu programul Visual Paradigm pentru UML 11.0)

# Capitolul 4. Ghid de utilizare

## Utilizarea Plug-inului



**Figura 4.1 - Interfaţa Plugin-ului - Creare Mediu**

(Sursa: imaginea autorului realizată cu programul Autodesk Maya)

Din cauza structurii Plugin-ului, utilizarea lui este foarte uşoară. În Figura 4.1 este prezentată interfaţa Plugin-ului. El este împărţit în 3 părţi mari:

1. **Pământ** În câmpurile *lungime* şi *lăţime* se introduce lungimea şi lăţimea planului ce va fi creat. Pentru a seta lungimea şi lăţimea planului ne putem ajuta şi de slide-urile din dreptul acestora.  
    La apăsarea butonului *Creare plan drept* se crează un plan drept.

La apăsarea butonului *Creare plan curbat* se crează un plan curbat.

La apăsarea butonului *Şterge plan* se şterge planul, fie curbat fie drept.

Reguli:

* 1. Dacă există un plan deja creat, fie drept fie curbat, şi se apasă butonul *Creare plan drept* sau *Creare plan curbat* se va afişa un mesaj de de tip *Yes/No* textul: *"Doriţi regenerarea unui nou plan?".*
  2. Dacă se apasă butonul *Ştergere plan* se va afişa un mesaj de avertizare de tip "Yes/No" cu textul: *"Sunteţi sigur că doriţi să ştergeţi planul existent?".*
  3. Dacă se apasă butonul *Ştergere plan*, dar nu există niciun plan se va afişa un mesaj de eroare cu textul: *"Nu există niciun obiect cu numele 'Pamant' creat!".*
  4. Dacă nu există niciun plan creat atunci groupBox-ul poieni şi cel al pădurii este setat "diseable" (nu se poate crea o poieniţă sau o pădure).
  5. valorile din câmpurile *Lungime şi Lăţime* se vor limita între 300 şi 600 pentru a nu produce erori.

1. **Poiană** La apăsarea butonului *Importă vertex selectat* se importă vertexul selectat din plan în *Câmp centru (vertex)*. Acesta reprezintă centrul cercului unde se va crea poiană.  
   În *Câmp rază* se introduce lungimea razei pe care se va genera pornită.  
    În câmpurile *Floare albă, Floare portocalie* şi *Floare roşie* se introduc procentele de flori ce se vor genera.

În câmpul *Densitate* din Layoutul florilor se va introduce numărul de flori ce vor fi generate.  
 În câmpul *Densitate* din Layoutul Iarbă se introduce densitatea ierbii ce va fi generată.  
 În câmpul *Piatră mică* se introduce densitatea pietrelor mici ce vor fi generate.  
 În câmpul *Piatră mare* se introduce densitatea pietrelor mari ce vor fi generate.   
 La apăsarea butonului *Generare poiană*  se va genera o poiană pe o rază introdusă în câmpul *Câmp rază* cu centrul în vertexul introdus în câmpul *Câmp centru (vertex)* şi se va adăuga numele poieni în listă.

La apăsarea butonului *Selectează Poiană* se va selecta în spaţiu de lucru poiană ce este selectată în listă.

La apăsarea butonului *Şterge poiană selectată* se va şterge din spaţiul de lucru şi din lista poiană selectată în listă.

La apăsarea butonului *Şterge toate poienile* se vor şterge din spaţiul de lucru şi din lista toate poienile create.

Reguli:

* + 1. Dacă suma introdusă în câmpurile *Floare albă, Floare portocalie* şi *Floare roşie* este diferită de 100 atunci se va recalcula, iar diferenţa se va adăuga la ultimul câmp *Floare roşie.*
    2. Dacă se apasă butonul *Importă vertex selectat* şi nu este selectat măcar un vertex din plan, atunci se va afişa un mesaj de eroare cu textul: *"Selectaţi un vertex pentru a fi importat!".*
    3. Dacă densitatea ierbii, densitatea florilor, numărul pietrelor mici şi cel al pietrelor mari este 0, atunci nu se va genera nimic.
    4. Dacă există cel puţin o poiană, atunci Layoutul Pământ se va seta indisponibil.
    5. Dacă se apăsă butonul *Şterge toate poienile* atunci se va afişa un mesaj de tip *Yes/No* cu textul: "*Sunteţi sigur că doriţi să ştergeţi toate poienile?".*

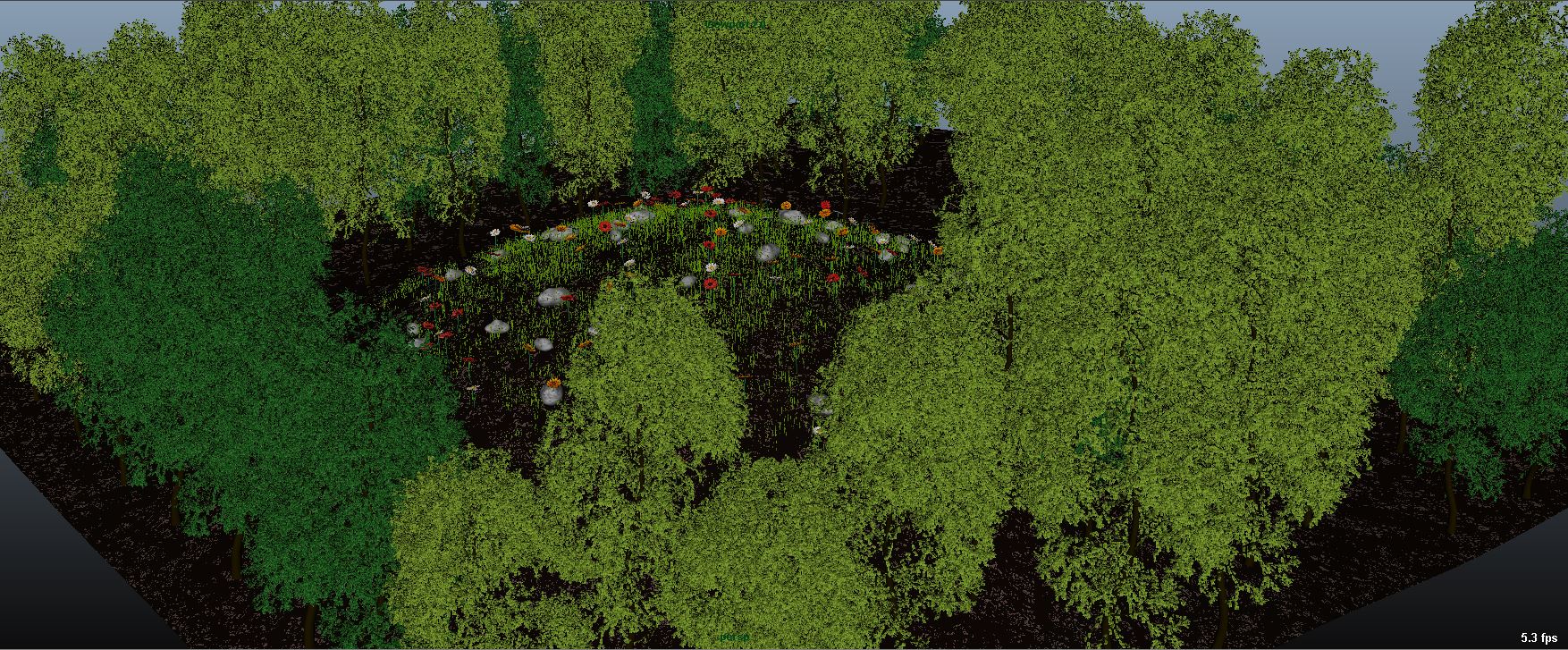
1. **Pădure** În câmpurile *Copac cu frunze verzi* şi *Copac cu frunze galbene* se introduc procentele copacilor ce vor fi generaţi.

La apăsarea butonului *Generare pădure* se va genera pădurea pe poziţiile libere din planul pământului, fără a se suprapune cu poiana sau poienile create.   
 La apăsarea butonului *Şterge Pădure* se va şterge pădurea creată dacă aceasta există.  
Reguli:

* 1. Dacă se apasă butonul *Generare Pădure* atunci se vor verifica condiţiile:
     + 1. Dacă Layoutul Pământ este activ atunci acesta va deveni inactiv.
       2. Dacă Layoutul Poiană este activ atunci şi acesta va deveni inactiv.
       3. Butonul *Generare Pădure* va deveni şi el la rândul lui inactiv.
     1. Dacă se apasă butonul *Şterge Pădure*, înainte să se şteargă pădurea se va afişa un mesaj de tip *Yes/No* cu textul: *"Sunteţi sigur că doriţi să ştergeţi pădurea existenta?".*

La apăsarea butonului *Cancel* se va închide Plugin-ul "Creare Mediu". Funcţia butonului este creată în PyQt4.

În Figura 4.2 este prezentat rezultatul obţinut cu ajutorul Plugin-ului. Crearea unei astfel de scene fără a folosi un script ar putea dura de la câteva zile la câteva săptămâni pentru un artist. Cu ajutorul acestui script generarea scenei poate dura de la câteva minute la câteva zeci de minute.



**Figura 4.2 - Rezultatul obtinut cu ajutorul Plugin-ului - "Creare Mediu"**

(Sursa: imaginea autorului realizată cu programul Autodesk Maya)

# Capitolul 5. Codul sursa

## InterfacePlugin.py

"""

# @path D:\Programare\Python\ProiectLicenta\trunk\apps\Plugin\app\python\PluginCreareMediu

# @file InterfacePlugin.py

# @brief Acest script creaza interfata Pluinu-lui "Creare Mediu".

# @author remus\_avram

# @date 12.2013

"""

# se imporata PyQt pentru a se putea citi fisierul cu extensie ".ui".

from PyQt4 import QtGui

from PyQt4 import QtCore

from PyQt4 import uic

from PyQt4 import Qt

import os

import sip

import Plugin\_v04 as Plugin

# se memoreza calea catre fisierul cu extensie ".ui"

uiFile = os.path.join(os.path.dirname(\_\_file\_\_), '../../ui/PluginCreareMediuUI\_v08.ui')

# se memoreaza caile catre imaginile importate în interfata

labelWhiteFlowerPath = os.path.join(os.path.dirname(\_\_file\_\_), '../../images/WhiteFlower.jpg')

labelOrangeFlowerPath = os.path.join(os.path.dirname(\_\_file\_\_), '../../images/OrangeFlower.jpg')

labelRedFlowerPath = os.path.join(os.path.dirname(\_\_file\_\_), '../../images/RedFlower.jpg')

labelGrassPath = os.path.join(os.path.dirname(\_\_file\_\_), '../../images/Grass.jpg')

labelSmallStonePath = os.path.join(os.path.dirname(\_\_file\_\_), '../../images/SmallStone.jpg')

labelBigStonePath = os.path.join(os.path.dirname(\_\_file\_\_), '../../images/BigStone.jpg')

labelGreenTreePath = os.path.join(os.path.dirname(\_\_file\_\_), '../../images/greenTree.jpg')

labelYellowTreePath = os.path.join(os.path.dirname(\_\_file\_\_), '../../images/yellowTree.jpg')

# se retine numele ferestrei intr-o variabila

windowObject = 'PluginWindow'

class InterfacePluginCass(QtGui.QMainWindow):

def \_\_init\_\_(self, parent=None):

super(InterfacePluginCass, self).\_\_init\_\_(parent)

# se creaza un obiect de tip Plugin

self.PluginObject = Plugin.PluginClass()

uic.loadUi(uiFile, self)

# se seteaza numele ferestrei

self.setObjectName(windowObject)

self.mainWidget = QtGui.QWidget()

self.setCentralWidget(self.mainWidget)

self.mainWidget.setLayout(self.mainGrid)

# se verifica daca pamantul exista în momentul în care fereastra este redeschisa

if self.PluginObject.checkExistGround() == 1:

self.meadowGroupBox.setEnabled(True)

self.forestGroupBox.setEnabled(True)

# se verifica daca padurea exista în momentul în care fereastra este redeschisa

if self.PluginObject.checkExistingForest():

self.groundGrupBox.setEnabled(False)

self.meadowGroupBox.setEnabled(False)

self.generateForestPushButton.setEnabled(False)

# se verifica daca exista vreo poiana în momentul în care fereastra este redeschisa si se importa în lista

for i în self.PluginObject.checkExistingMeadow():

self.listMeadowWidget.addItem('Poiana\_' + str(i))

if self.PluginObject.checkExistingMeadow():

self.groundGrupBox.setEnabled(False)

# signals/slots pentru pamant

self.creatIrregularPlanePushButton.clicked.connect(self.createIrregularPlan)

self.createUniformPlanPushButton.clicked.connect(self.createUniformPlan)

self.deletePlanPushButton.clicked.connect(self.deletePlan)

# signals/slots pentru poiana

self.importSelectedVertexPushButton.clicked.connect(self.importSelectedVertex)

self.generateMeadowPushButton.clicked.connect(self.generateMeadow)

self.selectMeadowPushButton.clicked.connect(self.selectMeadow)

self.deleteSelectedMeadowPushButton.clicked.connect(self.deleteSelectedMeadow)

self.deleteAllMeadowsPushButton.clicked.connect(self.deleteAllMeadows)

# signals/slots pentru padure

self.generateForestPushButton.clicked.connect(self.generateForest)

self.deleteForestPushButton.clicked.connect(self.deleteForest)

# incarca imaginile în interfata

self.loadImage(labelWhiteFlowerPath, self.whiteFlowerImage)

self.loadImage(labelOrangeFlowerPath, self.orangeFlowerImage)

self.loadImage(labelRedFlowerPath, self.redFlowerImage)

self.loadImage(labelGrassPath, self.grassImage)

self.loadImage(labelSmallStonePath, self.smallStoneImage)

self.loadImage(labelBigStonePath, self.bigStoneImage)

self.loadImage(labelGreenTreePath, self.greenTreeImage)

self.loadImage(labelYellowTreePath, self.yellowTreeImage)

def loadImage(self, labelPath, labelName):

""" Aceasta metoda incarca o imagine în fisierul cu extensie ".ui". """

tempImage = QtGui.QImage(labelPath)

Map = QtGui.QPixmap()

picture = Map.fromImage(tempImage)

labelName.setPixmap(picture)

def createIrregularPlan(self):

""" Acesta metoda cheama createGround() din PluginClass. """

self.PluginObject.createGround(self.lengthSpinBox.value(),

self.widthSpinBox.value(),

'Irregular')

# se initializeaza campul centru cu 0

self.centerFieldSpinBox.setValue(0)

# se schimba Layout-ul poienii si cel al padurii activ

self.meadowGroupBox.setEnabled(True)

self.forestGroupBox.setEnabled(True)

def createUniformPlan(self):

""" Aceasta metoda cheama createGround() din CreatePluginClass. """

self.PluginObject.createGround(self.lengthSpinBox.value(),

self.widthSpinBox.value(),

'Uniform')

# se initializeaza campul centru cu 0

self.centerFieldSpinBox.setValue(0)

# se schimba Layout-ul poienii si cel al padurii activ

self.meadowGroupBox.setEnabled(True)

self.forestGroupBox.setEnabled(True)

def deletePlan(self):

""" Aceasta metoda sterge pamantul chemand deleteGround() din CreatePluginClass. """

response = self.PluginObject.deleteGround()

# se schimba Layout-ul poienii si cel al padurii inactiv

if response:

self.meadowGroupBox.setEnabled(False)

self.forestGroupBox.setEnabled(False)

def importSelectedVertex(self):

""" Aceasta metoda importa în centerFieldSpinBox vertex-ul selectat. """

numberVertex = self.PluginObject.selectedVertex()

self.centerFieldSpinBox.setValue(numberVertex)

def generateMeadow(self):

""" Aceasta metoda cheama generateMeadow() din CreatePluginClass. """

# se verifica daca a fost intordus procentul florilor corect

# daca nu se recalculeaza

newValueWhiteFlower, newValueOrangeFlower, newValueRedFlower = self.PluginObject.procentCheck(self.whiteFlowerSpinBox.value(),

self.orangeFlowerSpinBox.value(),

self.redFlowerSpinBox.value())

self.orangeFlowerSpinBox.setValue(newValueOrangeFlower)

self.redFlowerSpinBox.setValue(newValueRedFlower)

respons = self.PluginObject.generateMeadow(self.radiusSpinBox.value(),

self.whiteFlowerSpinBox.value(),

self.orangeFlowerSpinBox.value(),

self.redFlowerSpinBox.value(),

self.flowersDensitySpinBox.value(),

self.grassDensitySpinBox.value(),

self.smallStoneDensitySpinBox.value(),

self.bigStoneDensitySpinBox.value())

# daca poiana a fost creata atunci se va adauga si numele ei în lista

if respons:

self.listMeadowWidget.addItem(respons)

# daca exista cel putin o poiana creata atunci Kayout-ul pamantului se schimba în inactiv

if self.PluginObject.checkExistingMeadow():

self.groundGrupBox.setEnabled(False)

def selectMeadow(self):

""" Aceasta metoda selecteaza obiectul ce a fost selectat în lista. """

toSelect = []

for item în self.listMeadowWidget.selectedItems():

toSelect.append(item.text())

if toSelect:

self.PluginObject.selectObject('MeadowGroup\_' + toSelect[0][7:])

def deleteSelectedMeadow(self):

""" Aceasta metoda sterge obiectul selectat în lista. """

toSelect = []

for item în self.listMeadowWidget.selectedItems():

toSelect.append(item.text())

self.listMeadowWidget.takeItem(self.listMeadowWidget.row(item))

if toSelect:

self.PluginObject.deleteObject('MeadowGroup\_' + toSelect[0][7:])

# se verifica daca poiana stearsa a fost ultima

# si daca da face transforma Layout-ul pamantului în activ

if self.PluginObject.checkExistingMeadow() == []:

self.groundGrupBox.setEnabled(True)

def deleteAllMeadows(self):

""" Aceasta metoda sterge toate poienile. """

self.PluginObject.deleteAllObjectsNamed('MeadowGroup')

# seteaza Layout-ul Pamntului activ

self.groundGrupBox.setEnabled(True)

def generateForest(self):

""" Aceasta metoda apeleaza createForest() din CreatePluginClass. """

# se verifica daca a fost corect introdus procentul copacilor

# daca nu se recalculeaza

newValueGreenTree, newValueYellowTree, tempValue = self.PluginObject.procentCheck(self.greenTreeSpinBox.value(),

self.yellowTreeSpinBox.value())

self.yellowTreeSpinBox.setValue(newValueYellowTree+tempValue)

self.PluginObject.createForest(self.greenTreeSpinBox.value(), self.yellowTreeSpinBox.value())

# Layout-ul pamantului se seteaza inactiv, Layout-ul poieni se seteaza inactiv si generateForestPushButton tot inactiv

self.groundGrupBox.setEnabled(False)

self.meadowGroupBox.setEnabled(False)

self.generateForestPushButton.setEnabled(False)

def deleteForest(self):

""" Aceasta metoda sterge padurea. """

# se seteaza Layout-ul poieni activ daca padurea se sterge

if self.PluginObject.deleteForest():

self.meadowGroupBox.setEnabled(True)

self.generateForestPushButton.setEnabled(True)

# se verifica daca cel putin o poiana exista si daca nu Layout-ul Pamantului se seteaza activ altfel inactiv.

if self.PluginObject.checkExistingMeadow():

self.groundGrupBox.setEnabled(False)

else:

self.groundGrupBox.setEnabled(True)

# Bibliografie şi Webografie generală

1. Adam Mechtley, Ryan Trowbridge, *Maya Python for Games and Film. A complete Reference for Maya Python and the Maya Python API,* Morgan Kaufmann, Unite State of America, 2012
2. Mark Lutz, *Python Pocket Reference,* Editura O'Reilly, editia 4, Statele Unite ale Americii, 2010
3. Dariush Derakhshani, *Introduction Maya 2009,* Editura Wiley Publishing, Inc., Canada 2009
4. Allen B. Downey, *Python for Software Design - How to Think Like a Coputer Scientist*, Editura Cambridge University Press, United States of America, 2009
5. docs.python.org/
6. www.autodesk.com/
7. download.autodesk.com/global/docs/maya2013/en\_us/CommandsPython/
8. www.digitaltutors.com/software/Maya-tutorials
9. thenewboston.org/tutorials.php
10. www.fxphd.com/
11. www.3dbuzz.com/
12. cmivfx.com/
13. zurbrigg.com/maya-tutorials
14. www.lynda.com/
15. en.wikibooks.org/wiki/Computer\_Animation/Computer\_Generated\_Imagery
16. en.wikipedia.org/wiki/Computer-generated\_imagery
17. www.mindactive.com/cgi-computer-generated-imagery
18. en.wikipedia.org/wiki/Autodesk\_Maya
19. ro.wikipedia.org/wiki/Plugin
20. en.wikipedia.org/wiki/Plug-în\_(computing)
21. code.google.com/p/pymel/
22. www.autodesk.com/products/autodesk-maya/overview

Cuprins

[Introducere 1](#_Toc377983382)

[Capitolul 1. Ce înseamnă CGI? 3](#_Toc377983383)

[Computer-generated imagery (CGI) [7][8][9] 3](#_Toc377983384)

[Autodesk Maya - soft de creare CGI [10] 11](#_Toc377983385)

[Plugin - cheia dezvoltării softurilor de grafică [11][12] 14](#_Toc377983386)

[Capitolul 2. Programarea orientată pe obiecte în Python. Interacţiunea Python cu Maya 18](#_Toc377983387)

[Interacţionări cu Maya [1] 18](#_Toc377983388)

[Python versus MEL în Maya [1] 20](#_Toc377983389)

[Executarea limbajului de programare Python în Maya [1] 20](#_Toc377983390)

[Diferenţele principale dintre Python şi alte limbaje de programare 22](#_Toc377983391)

[Ce este un modul în Python? [1] 24](#_Toc377983392)

[Calea în Python [1] 29](#_Toc377983393)

[Programarea obiect-orientat în Maya [1] 31](#_Toc377983394)

[Programarea obiect-orientata versus programarea procedurală [1] 32](#_Toc377983395)

[Noţiuni de baza ale implementării unei clase în Python [1] 33](#_Toc377983396)

[Moştenirea [1] 35](#_Toc377983397)

[PyMEL [13] 36](#_Toc377983398)

[Capitolul 3. Plugin - "Creare Mediu". Descriere si funcţionalitate 39](#_Toc377983399)

[Descrierea şi funcţionalitatea Plug-inului "Creare Mediu" 39](#_Toc377983400)

[Descrierea instrucţiunilor de bază 40](#_Toc377983401)

[Capitolul 4. Ghid de utilizare. 55](#_Toc377983402)

[Utilizarea Plug-inului. 55](#_Toc377983403)

[Capitolul 5. Codul sursă 59](#_Toc377983404)

[InterfacePlugin.py 59](#_Toc377983405)

[Bibliografie şi Webografie generală 64](#_Toc377983406)

1. en.wikipedia.org/wiki/Autodesk\_Maya, *Maya (software)*, articol din data de 20.01.2014 [↑](#footnote-ref-1)
2. ro.wikipedia.org/wiki/Plugin, *Plugin,* articol din data de 20.01.2014 [↑](#footnote-ref-2)
3. en.wikipedia.org/wiki/Plug-in\_(computing), *Plug-in (computing),* articol din data de 20.01.2014 [↑](#footnote-ref-3)
4. en.wikipedia.org/wiki/Plug-in\_(computing), Plug-in (computing), articol din data de 20.01.2014 [↑](#footnote-ref-4)