**HeapUpBox Game**

**(Proiect IP)**

Coordonator, Student,

Prof. Cristian Sandu Luca Sergiu

ANUL III CTI

GRUPA 22C32 B

AN 2022-2023

Cuprins

[Introducere 4](#_Toc125497617)

[Ce este programarea orientată pe obiect? – „Object oriented” sau „object-oriented”? 4](#_Toc125497618)

[Avantajele programării orientată pe obiect: 4](#_Toc125497619)

[Cele mai populare limbaje orientate pe obiect: 5](#_Toc125497620)

[Obiecte vs clase 5](#_Toc125497621)

[Manualul jocului 6](#_Toc125497622)

[Lansarea Aplicatiei 6](#_Toc125497623)

[Selectarea nivelului 7](#_Toc125497624)

[Jucatul nivelului 8](#_Toc125497625)

[Resursele jocului 10](#_Toc125497626)

[Logo-ul 11](#_Toc125497627)

[Fundalul de ecran 11](#_Toc125497628)

[Tehnologiile folosite 13](#_Toc125497629)

[Java 13](#_Toc125497630)

[JavaFX 13](#_Toc125497631)

[Fisierele .fxml 13](#_Toc125497632)

[Vizualizarea codului sursa 14](#_Toc125497633)

[Sabloane de proiectare 16](#_Toc125497634)

[Adapter 16](#_Toc125497635)

[Arhitectura si Structura sistemului 17](#_Toc125497636)

[Calculul indicelui de stabilitate 19](#_Toc125497637)

[Planul de testare 20](#_Toc125497638)

[Lista de cazuri testate 21](#_Toc125497639)

[Rezultatele testelor 22](#_Toc125497640)

[Bibliografie resurse si librarii 23](#_Toc125497641)

# Introducere

## Ce este programarea orientată pe obiect? – „Object oriented” sau „object-oriented”?

Programarea orientată pe obiect este una dintre cele mai importante paradigme ale programării. Este un sistem de principii care definesc ce logică de bază folosim pentru a rezolva o sarcină dată și pentru a ne structura programele și cum elementele sunt conectate unele cu altele.

Un limbaj de programare poate urma multe paradigme, iar o bucată de cod poate conține mai multe paradigme. Există paradigme de programare procedurale, orientate pe obiect, funcționale și logice.

## Avantajele programării orientată pe obiect:

* E mai ușor să modelezi lucruri complexe ca structuri mai simple și ușor de reprodus
* Codul e ușor de reutilizat
* Putem crea programre care rulează mai rapid cu acesta
* Structura codului e mai transparentă
* Dezvoltarea e mai rapidă deoarece clase paralele pot fi dezvoltate în același timp
* Programul poate fi modifica, depanat și menținut mai ușor
* E mai sigură (encapsulearea și abstractizarea fac datele mai sigure)

## Cele mai populare limbaje orientate pe obiect:

* C++
* C#
* Java (limbaj folosit si in acest proiect alaturi de libraria JavaFX)
* JavaScript
* Python

## Obiecte vs clase

Conceptul principal al programării orientate pe obiect este obiectul. La fel ca și obiectele din viața reală, obiectele din programare au proprietăți și metode.

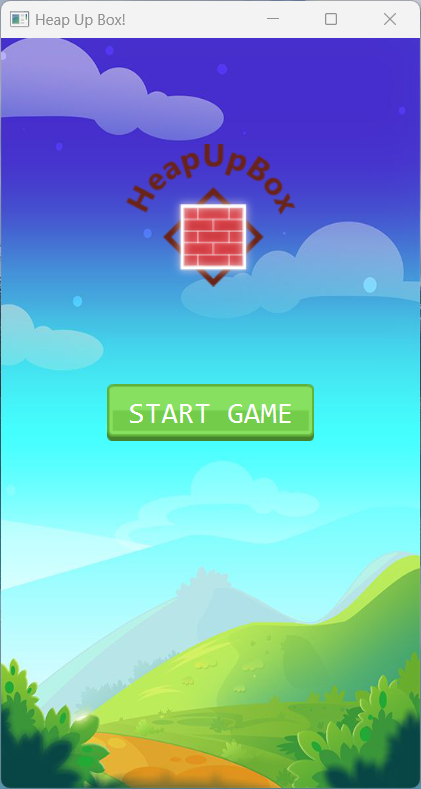
Structura datelor și a funcțiilor stocate în obiecte nu trebuie reintroduse iar și iar în timpul programării. Pentru a face procesul de codare și codul mai simple și mai economice, creăm planuri numite clase. Acestea conțin informații de baze despre structura obiectelor, iar obiectele conțin mai multe bucăți de date și metode.

# Manualul jocului

În Heap Up Box, tot ce ai de făcut este să clădești toate cutiile prezente în fiecare nivel. Trebuie să clădești până ce ajungi la linia punctată în limita de timp pe care o ai la dispoziție. Pare simplu, nu-i așa?

Succes!

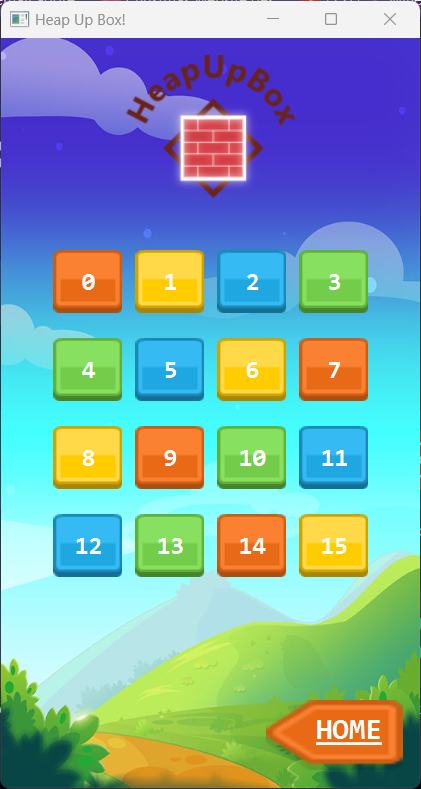
## Lansarea Aplicatiei



Odata lansat jocul, jucatorul este intampinat de ecranul „Acasa” ce contine Logo ul jocului si butonul „START GAME”.

Odata apasat butonul „START GAME” ne va trimite pe ecranul „Selectarea nivelului”.

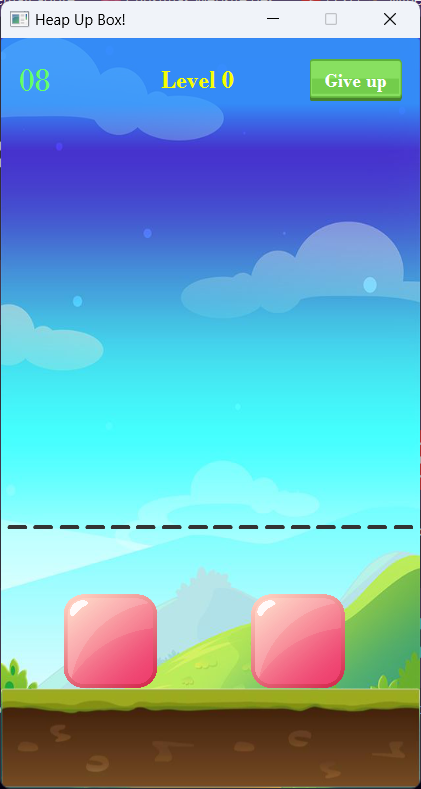
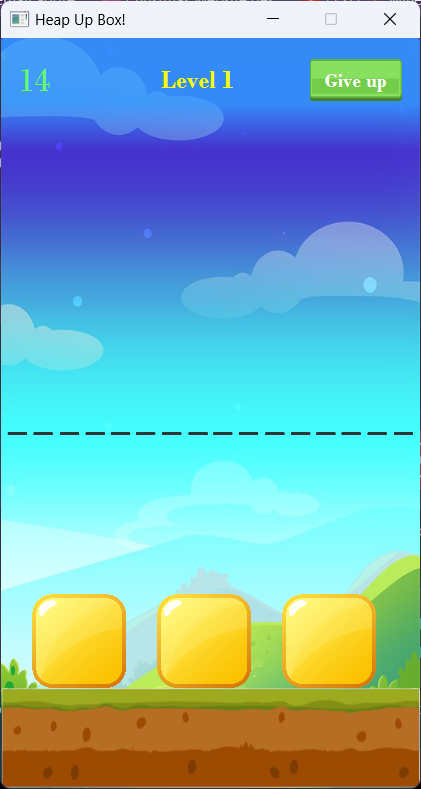
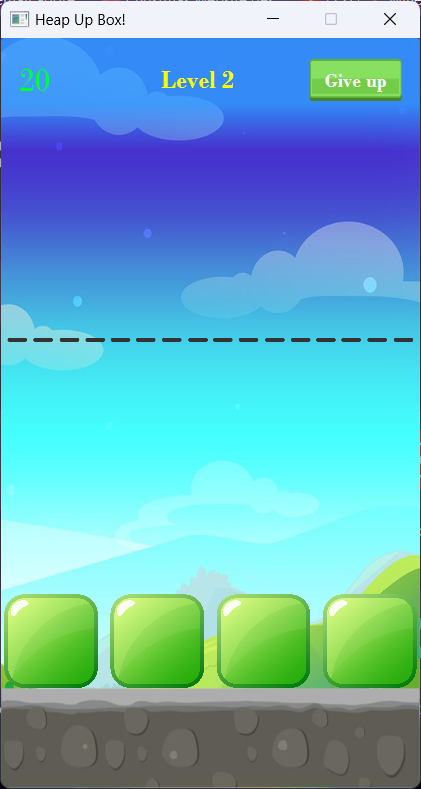
## Selectarea nivelului



Pe ecranul „Selectarea nivelului” este diponibil inca o data logo-ul jocului alaturi de butoanele specifice pentru selectarea nivelului dorit si butonul „Home” ce ne va redirectiona catre ecranul „Acasa” odata apasat.

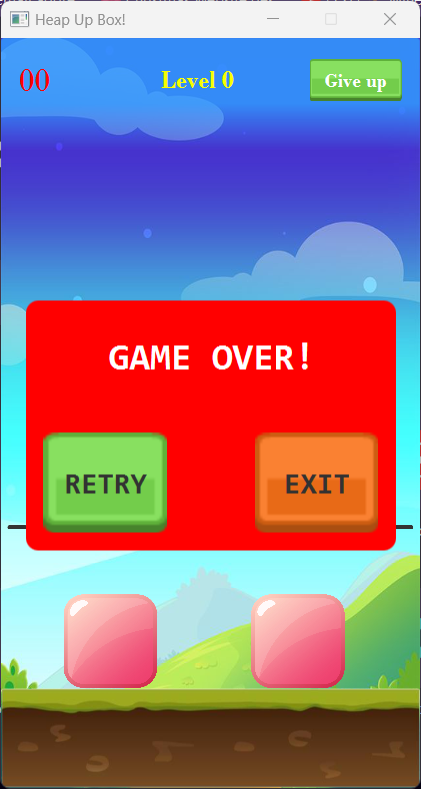
Fiecare buton de nivel apasat reprezinta un set nou de cutii si platforma cu o linie de Win total noua generata in functie de dificultatea nivelului.

## Jucatul nivelului

Fiecare nivel poate avea un tip diferit de cutii si un tip diferit de platforma.

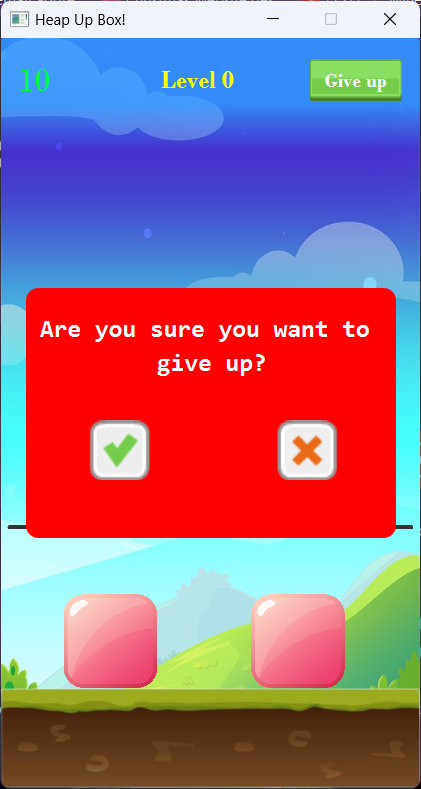
In functie de dificultatea nivelului vor exista mai multe sau mai putine cutii iar timpul disponibil va fi ajustat conform cutiilor existente.



Odata ce timpul a ajuns la 0 iar jucatorul nu a reusit sa faca turnul din cutii se afiseaza fereastra „Sfarsit joc” si sunt afisate doua optiuni.

Butonul „Retry” – Acest buton poate fi folosit pentru a reincarca nivelul.

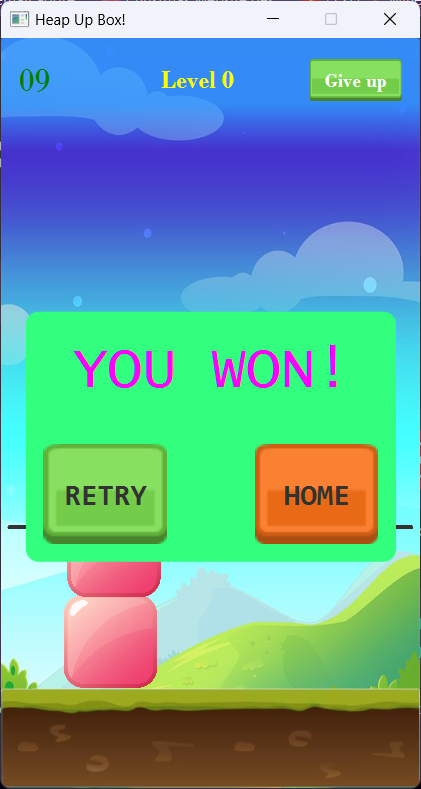
Butonul „Exit” – Acest buton poate fi folosit pentru a reintoarce jucatorul la ecranul „Selectarea nivelului”.



Daca jucatorul considera ca nu poate realiza nivelul, acesta poate apasa butonul „Give Up” pentru a renunta la incercarea curenta. Odata apasat jucatorului ii este afisat fereasta de confirmare a renuntarii la joc impreuna cu butoanele aferenta.

Butonul Bifa – Acest buton poate fi folosit pentru a confirma renuntarea si a reveni la ecranul „Selectarea nivelului”.

Butonul X – Acest buton poate fi folosit de catre jucator pentru a anula incercarea de a renunta la nivelul curent si a continua de unde a ramas.



Daca jucatorul a reusit sa faca turnul din cutii in timp util fereastra de castig va fi afisata.

Aceasta prezinta tot 2 butoane pentru a reincerca nivelul si a obtine un timp mai bun si pentru a redirectiona jucatorul catre ecranul „Acasa”.

Butonul „Retry” – Acest buton poate fi folosit pentru a reincarca nivelul.

Butonul „Home” – Acest buton poate fi folosit pentru a reintoarce jucatorul la ecranul „Selectarea nivelului”.

# Resursele jocului

Pentru resursele jocului am folosit kenney assets.

De pe acest website am descarcat si folosit:

* Cutiile:

* Platformele:

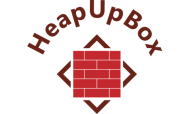




* Butoanele:



Logo-ul a fost creat cu ajutorul aplicatiei Canva.com folosind un template predefinit:



Fundalul de ecran a fost descarcat de pe aplicatia Pinterest:



Jocului ofera de asemenea sunete pentru timer, eroare, start joc, renuntare nivel, joc pierdut si joc castigat.

# Tehnologiile folosite

Java este un limbaj de programare puternic și multiplatformă, care permite scrierea codului care poate fi executat pe orice sistem care are instalat un runtime Java. Acest lucru face ca Java să fie o alegere populară pentru dezvoltarea de aplicații desktop și web.

JavaFX este o platformă software construită pe Java, care permite crearea și livrarea aplicațiilor desktop, precum și a aplicațiilor web bogate (RIAs). JavaFX oferă o serie de instrumente și biblioteci pentru crearea de interfețe utilizator grafice și gestionarea evenimentelor, precum și suport pentru animații, efecte grafice și suport pentru multimedia.

In acest joc, JavaFX este utilizat pentru a construi interfața grafică a aplicației, precum și gestionarea evenimentelor și a interacțiunilor utilizatorului. Pe langa acestea, in acest proiect s-au folosit unele biblioteci Java pentru a gestiona logica jocului si a obiectelor din el.

Fisierele .fxml sunt utilizate în aplicațiile JavaFX pentru a defini interfața grafică a utilizatorului (GUI) prin intermediul unui format de fișier XML. Acest format permite dezvoltatorilor să creeze și să organizeze elementele grafice din interfața utilizatorului într-un mod declarativ, în loc să fie nevoiți să scrie cod Java pentru a construi interfața manual.

Fisierele fxml pot include elemente precum butoane, etichete, câmpuri de text, tabele, imagini, etc. și pot fi utilizate pentru a crea diferite pagini sau ferestre din aplicație. Acestea sunt de obicei asociate cu clase Java care se ocupa de logica și comportamentul acestor elemente grafice.

In concluzie, acest joc utilizeaza tehnologiile Java și JavaFX pentru a construi un joc desktop amuzant și interactiv numit "HeapUpBoxGame".

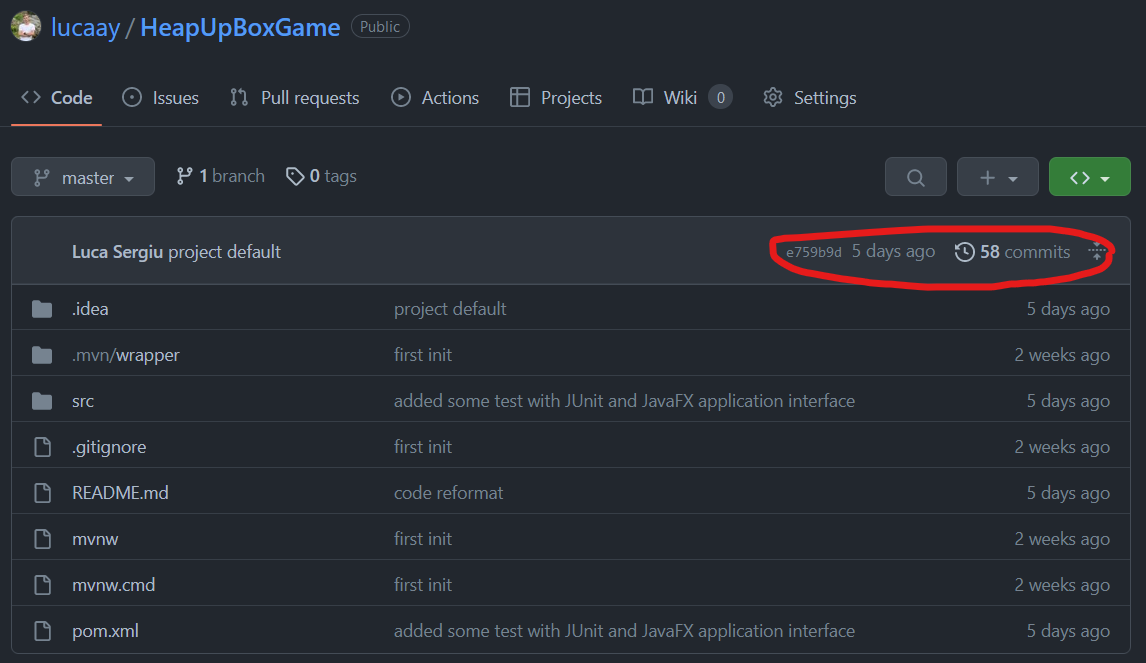
# Vizualizarea codului sursa

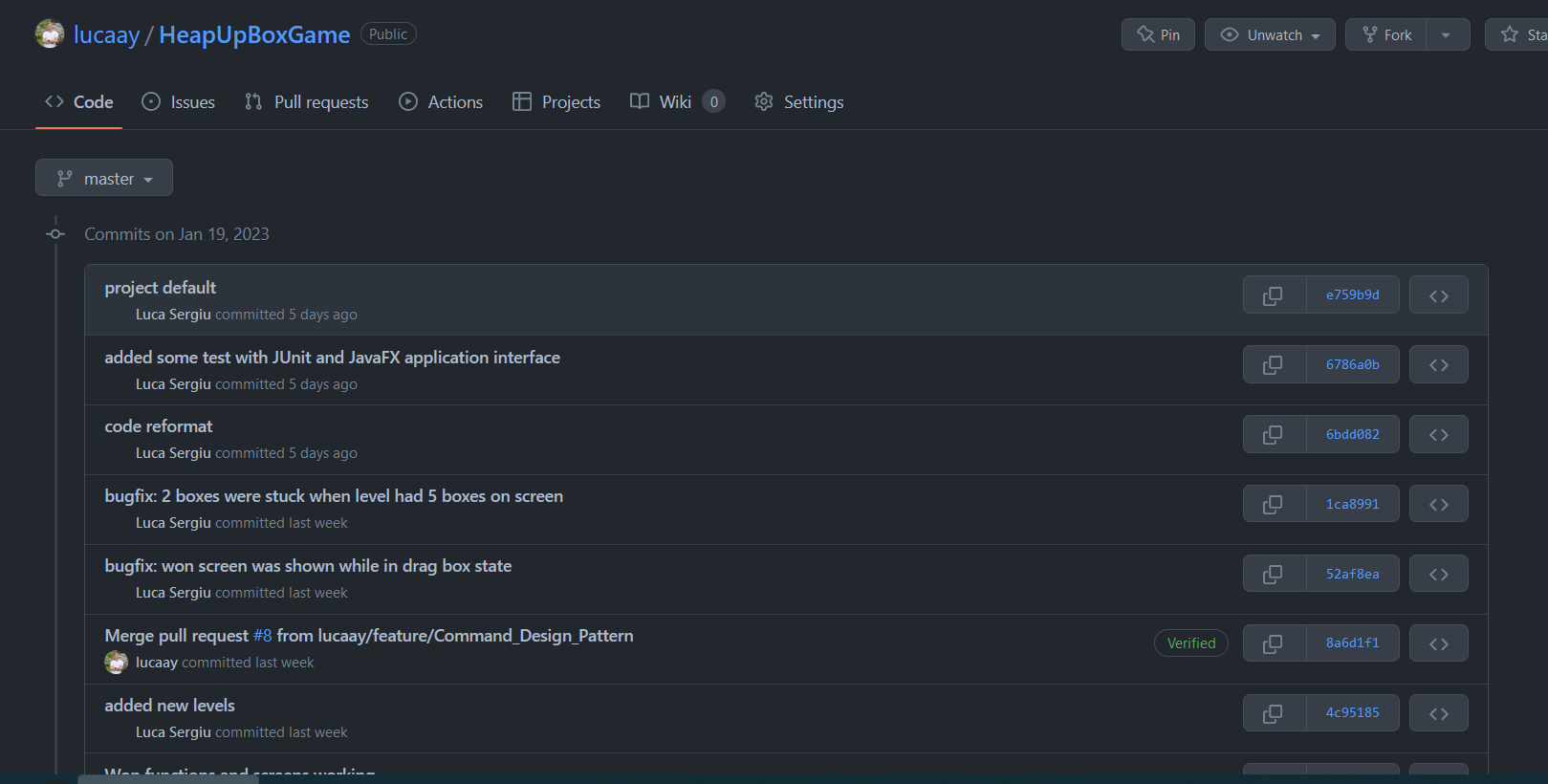
Codul sursa poate fi vizualizat pe linkul urmator:

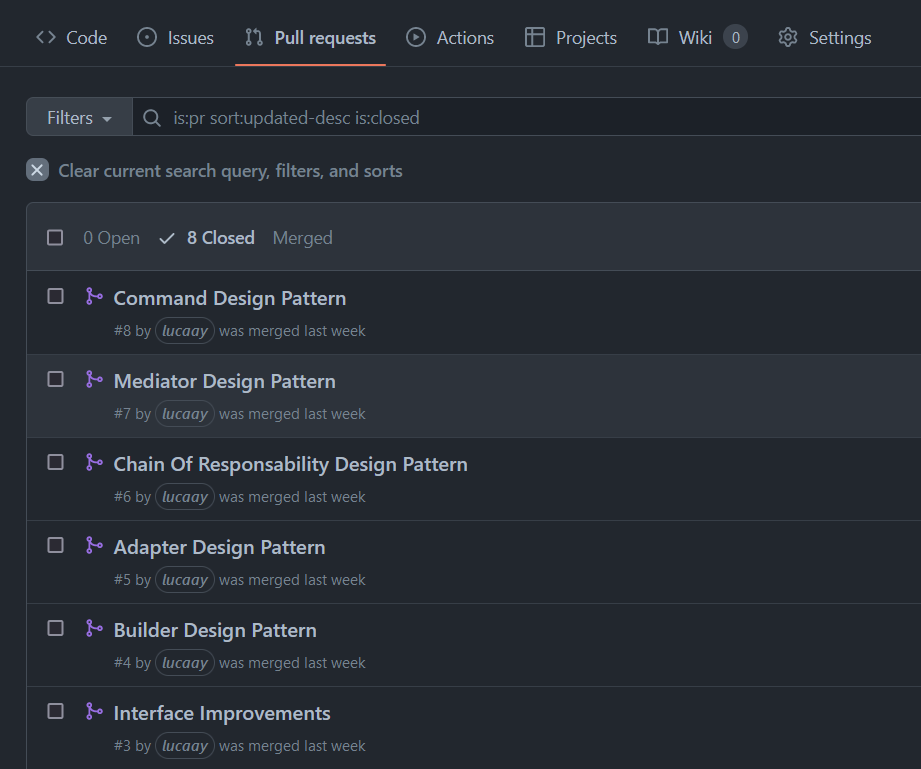
<https://github.com/lucaay/HeapUpBoxGame>

Aici pot fi vizualizate urmatoarele:

* Tot codul jocului
* Mai multe commituri ce arata modificarile realizate de-a lungul dezvoltarii jocului
* Codul impartit pe Pull Request uri reprezentand Etapele majore din dezvoltarea jocului.
* Cate un mesaj pentru fiecare commit ce explica modificarile realizate







# Sabloane de proiectare

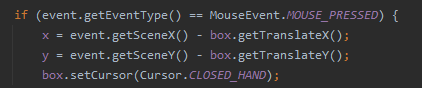
## Adapter

Acest design pattern permite adaptarea unei clase sau a unei interfete la alta, astfel încât să poată fi utilizată împreună cu alte componente care au o interfață diferită. Acest pattern este folosit pentru a permite comunicarea între componente care nu ar fi putut comunica în mod normal din cauza unor diferențe de interfață.

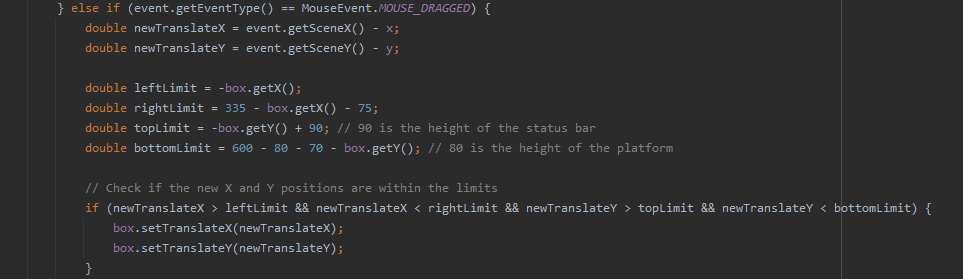
In acest joc am folosit Adapter pentru a realiza legatura dintre mouse ul jucatorului si obiectele din nivel.



Odata apasata o cutie se preiau coordonatele x si y ale acesteia si sunt stocate in variabilele private locale:



In momentul in care jucatorul tine apasat click ul si muta cutiile pe ecran se stocheaza noile coordonate constant in timpul mutarii si se asigura faptul ca jucatorul nu poate scoate cutiile din ecran.



La sfarsitul mutarii cutiei selectate aceasta va fi actualizata cu noua sa pozitie.

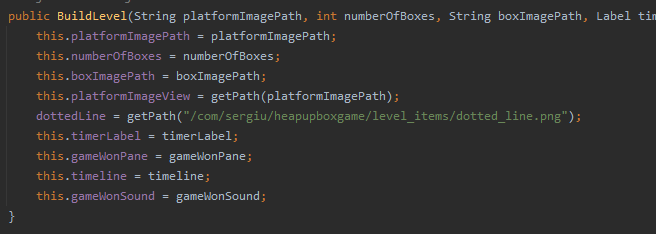
## Builder

Acest design pattern permite construirea unui obiect complex prin intermediul unui proces pas cu pas. Acest pattern este util atunci când există o varietate de opțiuni pentru construirea unui obiect sau atunci când construirea unui obiect este un proces complicat.

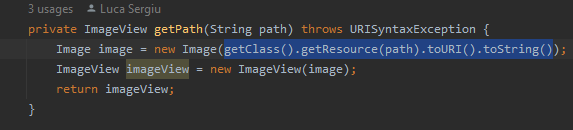
In acest joc am folosit Builder pentru a creea nivelele jocului cat mai usor si modular.



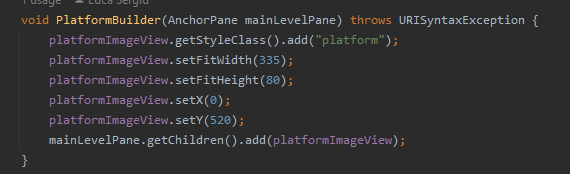
Am creat clasa BuildLevel ce contine toate variabilele necesare initializarii unui nivel:



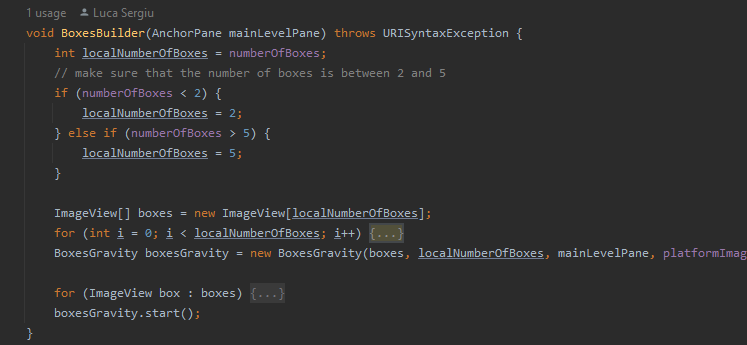
Am creat functia getPath ce schimba caile imaginilor in Obiecte ImageView pentru a putea fi afisate in joc:



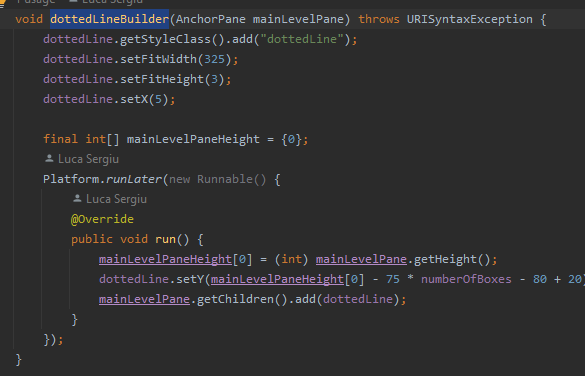
Am creat functia PlatformBuilder ce are ca rol initializarea, dimensionarea si pozitionarea platformei alese:



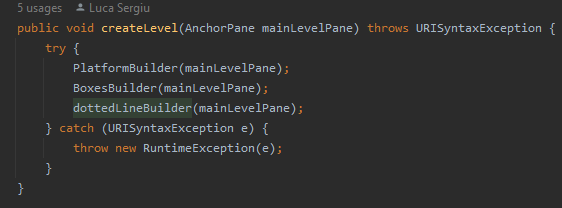
Am creat functia BoxesBuilder ce are ca rol initializarea cutiilor alese cat si a fortei gravitationale asupra lor (folosind sablonul de proiectare Chain Of Responsability prezentat mai jos):



Am creat functia dottedLineBuilder ce are ca rol initializarea liniei de Win alese cat si a pozitiei acesteia in functie de bara de status si platforma existenta:



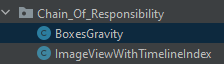
Iar la final am creat functia createLevel ce apeleaza cele 3 functii de mai sus pentru initializarea unui nou nivel:



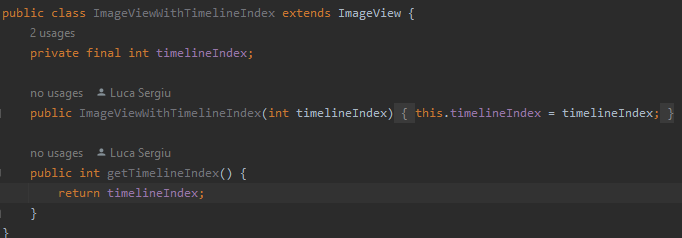
## Chain of Responsability

Acest design pattern permite transmisia unei cereri printr-o șiră de obiecte, până când unul dintre obiecte este capabil să o gestioneze. Acest pattern este util pentru a evita să se lege un obiect la mai mulți alte obiecte care ar putea să gestioneze cererea.

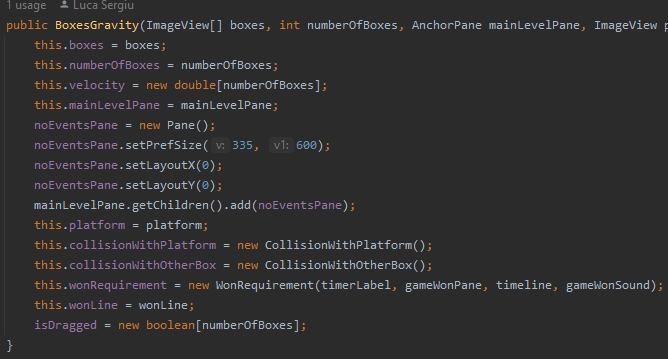
In acest joc am folosit Chain of Responsability pentru a realiza pulsul gravitational al cutiilor:



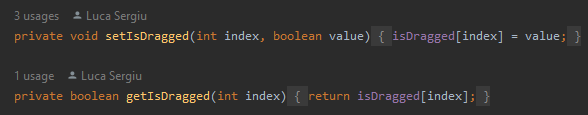
Am creat o clasa noua ce initializeaza un ImageView cu index propriu pentru a fi mai usor urmarirea cutiilor ce cad si a celei selectate:



Am creat clasa BoxesGravity ce contine toti parametrii necesari initializarii pulsului gravitational si detectarii platformei si a liniei de Win:



Am creat o variabila isDragged in care se stocheaza daca o cutie este sau nu in mutare de catre jucator:



Functia moveBoxes realizeaza pulsul gravitational si detecteaza daca cutia este in coliziune cu platforma, alta cutie sau este mutata de catre jucator. Daca cutia este in coliziune sau este mutata de jucator pulsul gravitational al acesteia se va opri pana cand coliziunea dispare sau jucatorul decide sa elibereze cutia:



Totodata, atunci cand cutiile sunt asezate corect una peste alta, iar turnul format din acestea ajunge si atinge linia de Win, fereastra You Won va fi afisata si va anunta castigare nivelului.

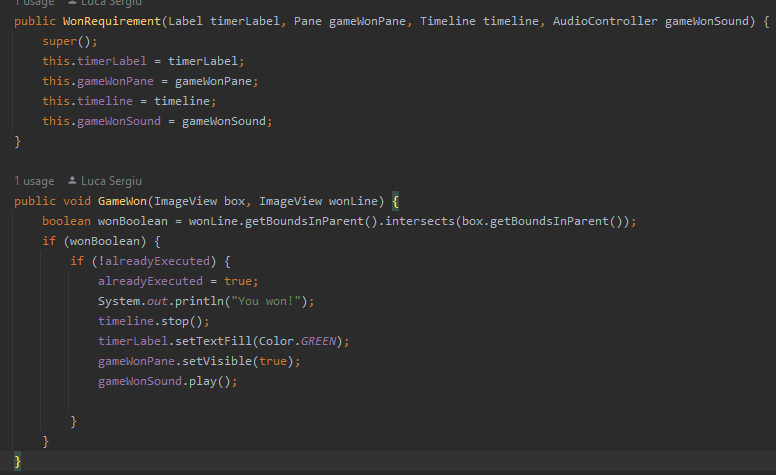
## Command

Acest design pattern permite encapsularea unei cereri într-un obiect, astfel încât să poată fi parametrizată diferite clase de cereri, care pot fi stocate și executate în diferite momente.

In acest joc am folosit Command pentru a realiza verificarea castigarii jocului curent.



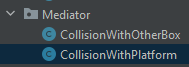
Functia GameWon din aceasta clasa controleaza daca trebuie si cand este afisata fereastra de castig joc:



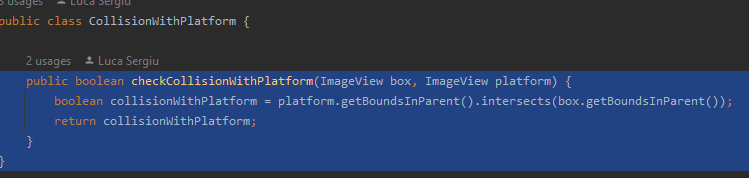
## Mediator

Acest design pattern permite centralizarea comunicării dintre mai multe obiecte, astfel încât obiectele nu mai trebuie să comunice direct între ele, ci prin intermediul unui obiect mediator. Acest pattern este util pentru a reduce dependența dintre obiecte și pentru a face sistemul mai flexibil.

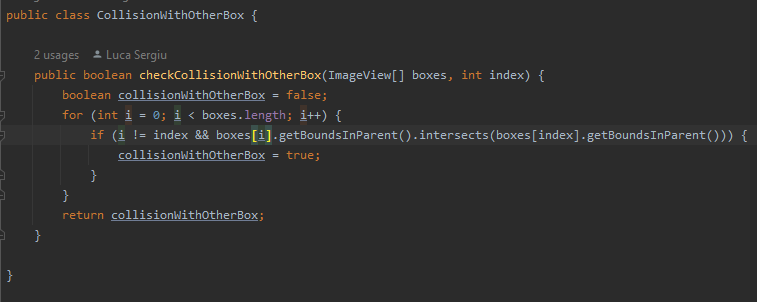
In acest joc am folosit Mediator pentru a realiza detectarea coliziunii unei cutii cu platform sau cu alta cutie.



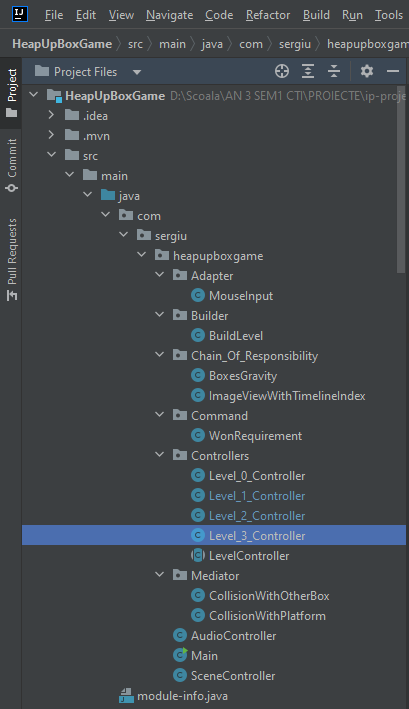
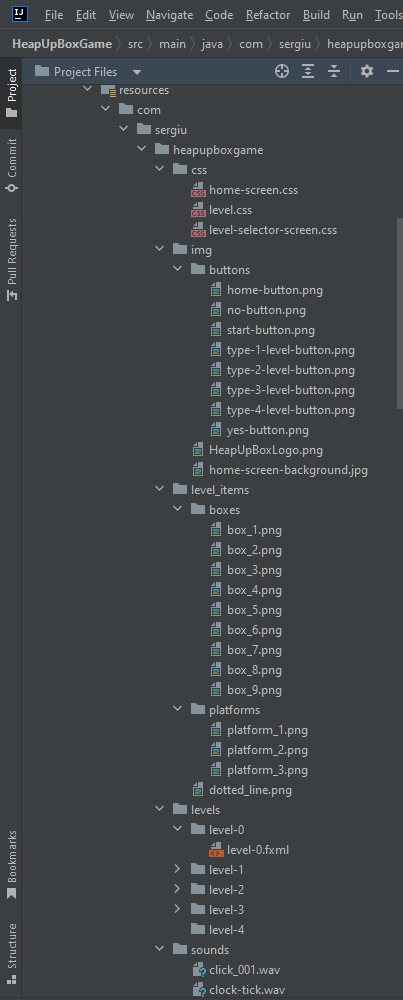
Functia ce detecteaza coliziunea cu platforma va returna o variabila de tip boolean si va instiinta restul obiectelor ce depinde de aceasta coliziune ca aceasta a avut loc:

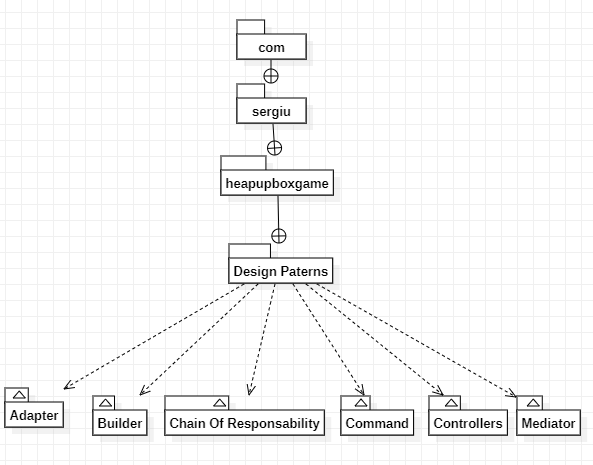


Functia ce detecteaza coliziunea cu alta cutie va returna o variabila de tip boolean si va instiinta restul cutiilor ce depinde de aceasta coliziune ca aceasta a avut loc:



# Arhitectura si Structura sistemului



# Calculul indicelui de stabilitate

Pentru a calcula indicele de stabilitate pentru fiecare pachet al unui proiect:

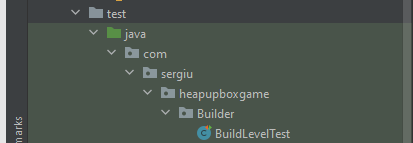
1. Identificați toate pachetele proiectului și definiți dependențele dintre acestea

Dependentele dintre pachetele unui proiect se referă la modul în care pachetele diferite din proiectul respectiv se afectează unul pe celălalt. Pachetele depind unul de celălalt atunci când un pachet necesită funcționalitate sau date oferite de un alt pachet pentru a funcționa corect.

1. Folosiți o metodă de analiză a dependențelor, cum ar fi diagrama de dependențe sau matricea de dependențe, pentru a identifica gradul de interdependență dintre pachete.
2. Calculați indicele de stabilitate pentru fiecare pachet utilizând o metodă adecvată.
3. Pentru fiecare pachet, calculati metodele McCabe sau Halstead pentru a masura complexitatea funțiilor și metodelor din pachet.
4. Utilizati aceasta masura de complexitate pentru a calcula indicele de stabilitate pentru fiecare pachet.
5. Interpretați rezultatele pentru a determina care pachete sunt cele mai stabile și care sunt cele mai vulnerabile la modificări.
6. Utilizați aceste informații pentru a lua decizii privind arhitectura și proiectarea proiectului și pentru a gestiona riscurile asociate cu modificările viitoare.

# Planul de testare

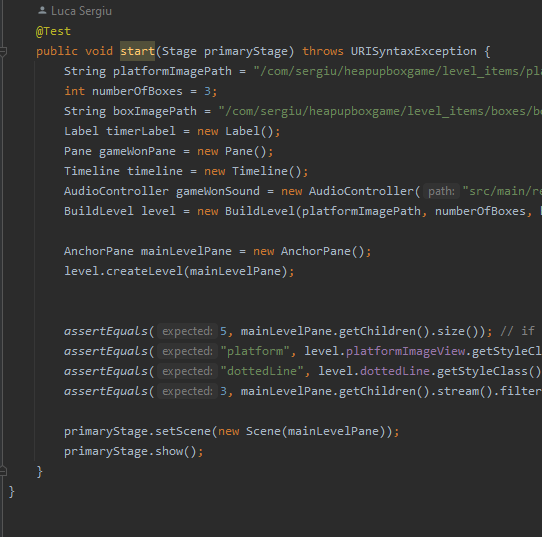
In acest exemplu, se creeaza o metoda de test numita BuildLevelTest pentru a testa daca elementele adaugat in scena in timpul initializarii nivelului sunt cele corecte.



Metoda de test creeaza un obiect de tip BuildLevel si ii atribuite parametri predeviniti in variabilele platformImagePath, numberOfBoxes, boxImagePath, timerLabel, gameWonPane, timeline, gemWonSound. Testul apoi foloseste metoda assertEquals din JUnit pentru a verifica daca aceste elemente sunt au fost create corespunsator si sunt afisare in scena.

Daca testele trec cu succes nu va fi afisata nici o eroare iar interfata va fi rulata si va putea fi jucata.

Daca testele nu trec cu success erorile ce au fost cauzate vor fi afisate in consola editorului.

\

# Lista de cazuri testate

Cazurile testate pentru aceasta aplicatie sunt:

* Cazul 1:

Se testeaza si se verifica daca toate elementele au fost adaugate in scena.

* Cazul 2:

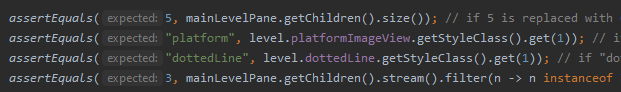
Se testeaza si se verifica daca platforma creata are clasa corecta pentru stiluri CSS.

* Cazul 3:

Se testeaza si se verifica daca linia punctata creata are clasa corecta pentru stiluri CSS.

* Cazul 4:

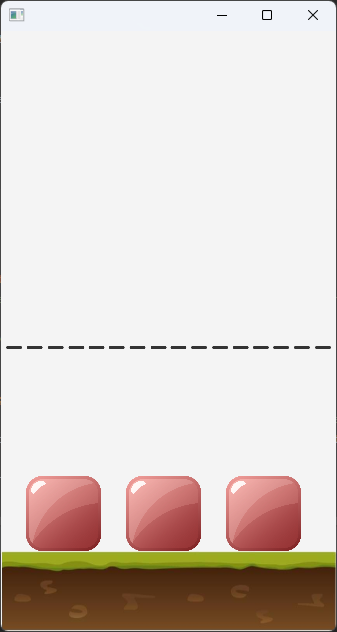
Se testeaza si se verifica daca cutiile create au clasa corecta pentru stiluri CSS.



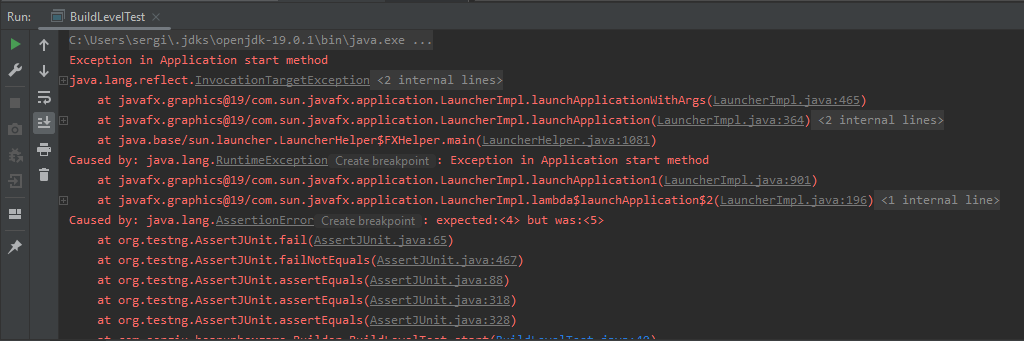


# Rezultatele testelor

Daca toate testele au fost realizate cu succes nu va fi afisata nici o eroare in consola si se va crea un nivel de test:



Daca testele NU au fost realizate cu succes va fi afisata o eroare in consola si NU se va crea un nivel de test:



# Bibliografie resurse si librarii

1. <https://www.java.com/en/>
2. <https://openjfx.io/>
3. <https://gluonhq.com/products/scene-builder/>
4. <https://kenney.nl/assets>
5. <https://opengameart.org/>
6. <https://staruml.io/>
7. IDE: <https://www.jetbrains.com/idea/>
8. GIT Version Control: <https://github.com/lucaay/>
9. GIT Client: <https://www.syntevo.com/smartgit/>