Cuprins

[Introducere 2](#_Toc26930)

[Fundamente Teoretice 2](#_Toc7247)

[Descrierea jocului 2](#_Toc31710)

[Concept, executie, explicatii 3](#_Toc14061)

[Bibliografie 6](#_Toc27451)

## Introducere

Jocurile de tip platformer sunt un sub gen al jocurilor video in care principalul mod de a interactiune cu lumea din joc este prin miscare. In general provocarea principala din aceste tipuri de jocuri este de a traversa de la inceputul la sfarsitul unui nivel. Depinzand de detaliile jocului inamici pot exista, sau nu, diverse mecanici avansate de traversare pot exista, sau nu.

In urma simplicitatii mecanicilor de baza necesare si a stilului vizual am ales sa creez o aplicatie care poate fi considerata un joc de tip platformer. Stilul aplicatiei va fi 2D cu grafica pixelizata si rudimentara deoarece am dorit sa creez resurse originale. Principalul obiectiv al jocului va fi ca jucatorul sa completeze intregul nivel. Inspiratia de baza vine de la un joc similar care am dorit sa il recreez. [1]

Pentru indeplinirea acestui proiect am folosit un game engine library numit LibGDX precum si cateva unelte externe precum Blender, Krita, Tiled Editor, Texture Packer si Physics Editor.

## Fundamente Teoretice

In special, pentru a avea o intelegere robusta a sistemelor utilizate in creearea acestui joc, am avut nevoie de constinte legate de:

(In nici un fel de ordine arbitrara)

1. Atlase de texturi
2. Ciclul de baza al exitentei unui obiect in mediul creat de LibGDX
3. Clasele, obiectele si tipurile de date din Java
4. Interactionarea cu resurse externe de tip imagine, sunet, etc folosind sistemele din LibGDX
5. Modul in care functioneaza scenele in LibGDX
6. Biblioteca box2D pentru sistemul de miscare fizic
7. Collision Handling
8. Garbage Disposal

Toate acestea vor fi detaliate mai jos in explicatia codului si a rularii intregii aplicatii.

## Descrierea jocului

Modul in care functioneaza jocul meu (si prin extensie cel pe care ma bazez) este in felul urmator:

Jucatorul porneste la baza nivelului si trebuie sa urce pana sus unde se afla finalul nivelului. Nivelul este inpartit in segmente mai mici pe care camera pune accent in momentul in care jucatorul se afla in acel segment. Jucatorul se poate misca la stanga si dreapta si poate sari in stanga, dreapta sau direct in sus. Puterea saltului este determinata de cat de mult jucatorul tine tasta spatiu apasata inainte sa sara. Odata ce jucatorul este in aer el nu are posibilitatea sa mai schimbe traiectoria saltului. Daca jucatorul atinge un perete in timpul saltului atunci el isi va schimba directia de miscare orizontala si va continua cu aceeasi viteza orizontala. Dificultatea consta in faptul ca jucatorul nu are un indicator precis care sa specifice cat de mare va fi saltul care trebuie sa il faca. In acest sens nu exista un mod de a pierde propriu zis, deoarece nivelul este conceput ca jucatorul sa piarda doar progress in cazul unui salt rau. Nu exista nici un mod permanent de a salva progres de-a lungul nivelului. Singura salvare care are loc este atunci cand jucatorul iese din joc in meniul principal. Aceasta salvare faciliteaza doar continuarea jocului in cazul mai multor sesiuni de joc, nu salveaza jucatorul in cazul in care acesta cade inapoi la inceputul nivelului.

## Concept, executie, explicatii

In primul rand voi elabora intr-o maniera rudimentara structura unui program facut cu LibGDX. Jocurile facute cu LibGDX se bazeaza pe un sistem de ‘Scene’ sau cum le numesc ei ecrane. Fiecare dintre aceste ecrane extind clasa Screens pentru a avea acces la metodele necesare fiecarei scene interactibile din LibGDX. Putem folosi un constructor pentu initializarea majoritatii obiectelor necesare.

Metoda show() ruleaza de fiecare data cand acest ecran este cel ce va fi redat pe ecran. Aceasta metoda este in special folositoare pentru scene care nu au fost curatate folosind garbage collectorul specific acestui game engine.

Metoda render() este chemata in fiecare frame care aplicatia il reda pe ecran. Aici are loc 80-90% din activitatea programului fie ca sunt simulari fizice, input handling sau redarea obiectelor pe ecran.

Metoda resize() este chemata de fiecare data cand fereastra isi schimba dimensiunile. Aici avem grija ca elementele de pe ecran sa fie redate centrate, in pozitiile lor corespunzatoare sau relativ corespunzatoare in functie de ratia de aspect a ecranului.

Metoda dispose() este locul in care toate elementele dintr-o scena care au legatura cu biblioteca LibGDX sunt distruse. In cazul in care nu avem grija de ce asseturi pot fi sau nu distruse cu aceasta metoda este posibil sa avem parte de memory leakuri.

Folosim apoi aceste scene pentru a reda in primul rand meniurile cu care jucatorul va interactiona. Avem nevoie de butoanele bine cunoscute, Play, Resume, Options, Exit. Avem nevoie si de slidere si check box-uri in meniul de optiuni. Din fericire aceste obiecte exista in LibGDX si sunt create cu usurinta folosind skinuri pentru partea vizuala. Aceste skinuri sunt produse de echipa LibGDX si sunt incarcate pe github. [2]

Dupa ce au fost create elementele UI sunt introduse intr-un tabel si afisate pe ecran sub obiectul Stage care contine metoda de interactiune cu eventurile produse de fiecare buton. Eventuri care le putem initializa cu success in show().

Toate aceste meniuri au o referinta parent la clasa principala MyGdxGame care este creata la initializarea programului si utilizata pana la inchiderea acestuia. In aceasta clasa putem crea cu usurinta o metoda care contoleaza ce ecran este afisat la momentul dat. Aceasta metoda o vom accesa apoi cu toate butoanele de inerfata care navigheaza intre meniuri.

Ajungand la jocul propriu zis procesul de ansamblu este urmatorul:

- Instantiem toate obiectele necesare in constructor

- Facem o metoda pt. constructor care va avea grija de initializarea asseturilor bazat pe faptul ca jocul poate sa fie unul complet nou sau o continuare a unui joc antecedent.

- Pregatim mediul fizic cu care jucatorul va interactiona, incluzand harta

- In render() vom detecta inputul, apoi vom face simularea de miscare fizica si la final vom reda toate elementele vizuale care exista.

Voi descrie din acest punct elementele din instanta principala de joc in ordinea in care le-am implementat original, deoarece prezinta si cea mai organica modalitate de abordare a subiectului. EG: Dorim sa punem la punct partea vizuala a proiectului inainte de a vedea daca miscarea obiectelor se intampla cu adevarat si/sau corpul fizic din box2D are pozitia sincronizata cu iconita de la jucator.

In primul rand aceasta scena contine o camera ce isi va schimba pozitia. Aceasta camera este si in mod inseparabil legata de obiectul care reda grafica pe ecran precum si partea vizuala a hartii jucatorului.

Aici vorbim despre tileuri si texture atlas. Din moment ce este mult mai usor sa stocam iconite modulare care asambleaza harta vom folosi o textura mare care va fi inpartita apoi de program. Astfel la momentul redarii pe ecran fiecare iconita modulara este recombinata intr-un buffer care este apoi trimis catre placa video. Textura ca atare poate fi desenata in orice program video-photo (Krita, Photoshop) si apoi adaugata in Tiled Map Editor. Aici vom putea impartii textura originala si crea harta pe care jucatorul o va vedea. Aceste fisiere le vom utiliza apoi in LibGDX.

Din moment ce folosim acest TileMap (Iconita = Tile) vom avea nevoie de TiledMapRenderer pentru redarea ei eficienta pe ecran. Din cauza la acest obiect avem nevoie si de o camera de tip ortogonal, din moment ce jocul este 2D pentru procesarea imaginii. Un avanataj la acest renderer este faptul ca putem incarca si celelalte obiecte independente (in acest caz doar iconita jucatorului) inainte de a compila si trimite bufferul, ceea ce este mai eficient decat a folosi 2 rendere diferite (unul pt fundal si unul generic).

In al 2-lea rand avem grija sa initializam univesul box2D. Simularea va fi efectuata in render folosind metoda stepWorld(float delta) care folosete deltaTime sau timpul de la ultimul frame. Fiecare obstacol din harta creata va fi initializat apoi in lumea box2D cu clasa MapBodyBuilder. In aceasta clasa detectam poligoanele de pe suprafata Objects din Tiled Editor si le transformam in poligoane fizice, statice care vor functiona ca platforme pt. jucatorul nostru.

Tot lumea box2D se ocupa de collision handling folosind metoda CollisionHandler. In aceasta metoda determinam ce se intampla in urma colisiunilor intre jucator si mediul inconjurator. Aceste collision eventuri ruleaza codul necesar pentru ricosarea de pe pereti in timpul saltului sau detectarea pamantului sub picioarele jucatorului. Aceste coliziuni sunt filtrate folosind operatii la nivel de bit unde fiecare putere a lui 2 reprezinta un al layer de coliziune asociat cu fixture-ul sau segmentul de corp al elementului fizic. (Un corp fizic poate avea 2 su mai multe fixture-uri, de aceea datele de filtru sunt legate de fixture si nu de corp).

Miscarea playerului se face folosind univerul box2D, utilizand date arbitrare oferite de mine eg. Viteza de micare, puterea saltului etc. Si input handling-ul care este inplementat din libGDX. Tot ce trebuie sa fac este sa preiau inputul in fieacre frame si apoi sa procesez fortele pt. Corpul jucatorului. Aceasta parte este facuta in interiorul unei metode din clasa Player.

Din moment ce miscarea jucatorului este implementata in clasa lui, acolo utilizez un stateMachine. Folosind enum putem seta starea jucatorului si preluand viteza lui de miscare precum si inputuri sau daca atinge sau nu pamantul in momentul de fata, putem determina ce animatii ar trebui sa aiba jucatorul. Animatiile sunt incarcate sub forma unor texturi prelungite care apoi le putem taia in segmente pt. A produce fiecare cadru al animatiei. Dupa care vom folosi o copie a acelui enum pentru a determina daca starea jucatorului s-a schimbat in ultimul cadru. Animation controllerul primeste apoi textura animatiei curente in cadrul metodei PrepareTextures();

Dupa aceea folosim MyUtility.drawAnimatedSprite() pentru a reda acel cadru de animatie la pozitia jucatoruliui, cu marimea, rotatia si orientarea dorita.

Utilizand apoi elementele de collision handling, meniurile si eventuri asemanatoare cu cele ale butoanelor din celelalte meniuri putem incheia jocul odata ce castigam. In metoda dispose() avem grija sa salvam pozitia jucatorului in caz de nevoie folosind preferintele din LibGDX pt. Stocare. Acestea sunt specifice pentru fiecare sistem si sunt responsabile si pentru salvarea nivelului volumului.

## Bibliografie

1. [https://store.steampowered.com/app/1061090/Jump\_King/](https://store.steampowered.com/app/1061090/Jump_King/;)
2. <https://github.com/czyzby/gdx-skins>

Toata informatia pt. Scrierea codului a fost preluata de la https://libgdx.com/wiki/