Cuprins

[1. Introducere 4](#_Toc170921067)

[1.1. Motivație 4](#_Toc170921068)

[1.2. Obiective 4](#_Toc170921069)

[2. Fundamente teoretice 6](#_Toc170921070)

[2.1. Contextul jocurilor și evoluția lor 6](#_Toc170921071)

[2.2. Exemple de jocuri similare și influențe 9](#_Toc170921072)

[**2.2.1.** Super Mario Bros 9](#_Toc170921073)

[**2.2.2.** Celeste 9](#_Toc170921074)

[**2.2.3.** Ori and the Blind Forest 10](#_Toc170921075)

[**2.2.4.** Analiza comparativă și inovațiile aduse de „În căutarea lui Pikachu” 11](#_Toc170921076)

[3. Tehnologii și instrumente utilizate 12](#_Toc170921077)

[3.1. Godot 4 12](#_Toc170921078)

[3.2. GDScript 12](#_Toc170921079)

[3.3. Paint.NET 13](#_Toc170921080)

[3.4. Adobe Illustrator 13](#_Toc170921081)

[4. Dezvoltarea Jocului "În Căutarea lui Pikachu" în Godot 4 14](#_Toc170921082)

[4.1. Analiza cerințelor 14](#_Toc170921083)

[4.2. Pregătirea mediului de lucru 14](#_Toc170921084)

[4.3. Pagina principală 18](#_Toc170921085)

[4.4. Gestionarea tranzițiilor 18](#_Toc170921086)

[4.5. Grafica și script-urile personajelor din cutscene 20](#_Toc170921087)

[4.6. Personajul “Anusk” și mecanismul lui de mișcare 22](#_Toc170921088)

[4.7. Gestionarea dialogurilor 25](#_Toc170921089)

[4.8. Momente cinematice 27](#_Toc170921090)

[4.8.1. Momentul cinematic “Intro” 27](#_Toc170921091)

[4.8.2. Momentul cinematic “lvl1\_cutscene” 28](#_Toc170921092)

[4.9. Sistemul de viață 30](#_Toc170921093)

[4.9.1. Grafica inimilor și actualizarea stării lor 30](#_Toc170921094)

[4.9.2. Alinierea inimilor 31](#_Toc170921095)

[4.9.3. Implementarea sistemului în nivele 31](#_Toc170921096)

[4.9.4. Pierderea vieților 32](#_Toc170921097)

[4.10. Checkpoint-uri 33](#_Toc170921098)

[4.10.1. Implementarea checkpoint-urilor 33](#_Toc170921099)

[4.10.2. Reapariția și resetarea poziției jucătorului 34](#_Toc170921100)

[4.11. Zona periculoasă 35](#_Toc170921101)

[4.12. Crearea și configurarea inamicilor 36](#_Toc170921102)

[4.12.1. Structura nodurilor inamicilor 36](#_Toc170921103)

[4.12.2. Implementarea comportamentului inamicilor 37](#_Toc170921104)

[4.12.3. Grafica inamicilor 39](#_Toc170921105)

[4.13. Implementarea steagului final 40](#_Toc170921106)

[4.14. Grafica nivelelor 41](#_Toc170921107)

[4.14.1. Nivelul 1 41](#_Toc170921108)

[4.14.2. Nivelul 2 42](#_Toc170921109)

[5. Ghidul utilizatorului 44](#_Toc170921110)

[5.1. Lansarea jocului 44](#_Toc170921111)

[5.2. Controalele jocului 45](#_Toc170921112)

[5.3. Activarea checkpoint-urilor 46](#_Toc170921113)

[5.4. Inamicii și pericolele 46](#_Toc170921114)

[5.5. Viața 47](#_Toc170921115)

[6. Concluzii 48](#_Toc170921116)

[Bibliografie 50](#_Toc170921117)

**Lista de figuri**

[Fig. 1-Tennis for two[17] 6](#_Toc170897715)

[Fig. 2-Pong cu mașina arcade [ 13] 7](#_Toc170897716)

[Fig. 3-Space Invaders[24] 7](#_Toc170897717)

[Fig. 4-Pac-Man[] 7](#_Toc170897718)

[Fig. 5-Street Fighter[6] 7](#_Toc170897719)

[Fig. 6-PlayStation[12] 8](#_Toc170897720)

[Fig. 7- Counter-Strike[5] 8](#_Toc170897721)

[Fig. 8- Grand Theft Auto V[1] 9](#_Toc170897722)

[Fig. 9- Red Dead Redemption II[16] 9](#_Toc170897723)

[Fig. 10- Super Mario Bros[22] 9](#_Toc170897724)

[Fig. 11-Celeste[3] 10](#_Toc170897725)

[Fig. 12- Ori and the Blind Forest[17] 10](#_Toc170897726)

[Fig. 13- Creare proiect 15](#_Toc170897727)

[Fig. 14-Dimensiuni Ecran 15](#_Toc170897728)

[Fig. 15-Configurare Taste 15](#_Toc170897729)

[Fig. 16- Poza personalizată cu Pictograma jocului 16](#_Toc170897730)

[Fig. 17-Resurse 16](#_Toc170897731)

[Fig. 18-Sectiunea „Autoload” din godot 17](#_Toc170897732)

[Fig. 19-Configurare layer-uri 17](#_Toc170897733)

[Fig. 20-Meniu Principal 18](#_Toc170897734)

[Fig. 21-Animația „incepe\_jocul” 19](#_Toc170897735)

[Fig. 22-Editarea Personajului "Ana\_pers" 20](#_Toc170897736)

[Fig. 23-Ana\_pers 20](#_Toc170897737)

[Fig. 24-Bogdan\_pers 20](#_Toc170897738)

[Fig. 25-Marius\_pers 21](#_Toc170897739)

[Fig. 26-Petru\_pers 21](#_Toc170897740)

[Fig. 27-Spanu\_pers 21](#_Toc170897741)

[Fig. 28-Cosmin\_pers 21](#_Toc170897742)

[Fig. 29-Noduri Jucător 22](#_Toc170897743)

[Fig. 30-Grafica "Anusk" 22](#_Toc170897744)

[Fig. 31-Configurare straturi jucător 23](#_Toc170897745)

[Fig. 32-Componentele balonului pentru dialog 25](#_Toc170897746)

[Fig. 33-Configurarea fundalului (Panel) 26](#_Toc170897747)

[Fig. 34-Configurarea Ancorelor (Panel) 26](#_Toc170897748)

[Fig. 35-Configurarea DialogueLabel 26](#_Toc170897749)

[Fig. 36-Dialogurile jocului 26](#_Toc170897750)

[Fig. 37-Animația "Scena1" 28](#_Toc170897751)

[Fig. 38-Animația "Plecare" 28](#_Toc170897752)

[Fig. 39-Structura scenei "lvl1\_cutscene" 29](#_Toc170897753)

[Fig. 40-Animația "animatie1" 30](#_Toc170897754)

[Fig. 41- Grafica inimilor 31](#_Toc170897755)

[Fig. 42- Structura Checkpoint-urilor 34](#_Toc170897756)

[Fig. 43-Grafica Checkpoint 34](#_Toc170897757)

[Fig. 44- Zona pericol a nivelului 1 36](#_Toc170897758)

[Fig. 45-Zona pericol a nivelului 2 36](#_Toc170897759)

[Fig. 46-Noduri inamic 37](#_Toc170897760)

[Fig. 47-Sprite-ul "Paianjen" 40](#_Toc170897761)

[Fig. 48-Sprite-ul "Sobo" 40](#_Toc170897762)

[Fig. 49-Grafica steagului final 41](#_Toc170897763)

[Fig. 50-Structura nivelului 1 42](#_Toc170897764)

[Fig. 51-Harta nivelului 1 43](#_Toc170897765)

[Fig. 52-Structura nivelului 2 43](#_Toc170897766)

[Fig. 53-Harta nivelului 2 44](#_Toc170897767)

[Fig. 54-pictograma jocului 45](#_Toc170897768)

[Fig. 55-Meniu principal 45](#_Toc170897769)

[Fig. 56-Trecerea peste dialoguri 46](#_Toc170897770)

[Fig. 57- tastele pentru control 46](#_Toc170897771)

[Fig. 58-Pisica kira 47](#_Toc170897772)

[Fig. 59- Păianjen 47](#_Toc170897773)

[Fig. 60-șobolan 47](#_Toc170897774)

[Fig. 61-Viată inițială a nivelului 2 48](#_Toc170897775)

[Fig. 62- Viată după ce s-au pierdut inimi 48](#_Toc170897776)

# Introducere

Dorința centrală a proiectului "În Căutarea lui Pikachu" este de a crea un joc captivant, construit pe baza platformei Godot 4 și dezvoltat în GDScript. Această lucrare este organizată în șase capitole ce explica pe larg procesul de dezvoltare a jocului "În Căutarea lui Pikachu", acoperind motivația ce a stat la baza proiectului, obiectivele propuse, tehnologiile utilizate, mecanismele de joc, elementele de grafică și muzică, precum și interfața cu utilizatorul și documentația necesară pentru a experimenta și a savura jocul în întregime.

## Motivație

Povestea din spatele jocului "În Căutarea lui Pikachu" este una plină de aventură, ce se învârte în jurul pasiunii pentru motociclism și descoperirea unor noi prietenii. Pentru mine, motociclismul nu este doar o activitate, ci o adevărată pasiune care m-a introdus într-un cerc de oameni fascinanți și m-a condus pe trasee pline de surprize și amintiri. Într-o zi însorită la cafeneaua noastră obișnuită, întâlnirea cu Marius, proeclit și Pikachu, a fost momentul care a dat naștere unei aventuri incredibile.

Cu Marius lipsind fără veste și cu telefonul său răsunând fără răspuns, am pornit în căutarea lui pe traseul pe care știam că îl parcurgea în fiecare weekend - drumul spre Poiana Brașov. Această călătorie neașteptată și plină de suspans ne-a adus împreună într-o căutare care a legat și mai strâns prietenia noastră. Această aventură de neuitat a devenit izvorul de inspirație pentru crearea jocului "În Căutarea lui Pikachu în care pasiunea pentru motociclism se îmbină armonios cu aventură și spiritul prieteniei.

## Obiective

Scopul esențial al proiectului "În Căutarea lui Pikachu" este de a crea un joc de succes, atrăgător și accesibil pentru o gamă diversă de jucători. Această dorință este ghidată de următoarele obiective specifice:

* Captivarea jucătorilor prîntr-un joc 2D plin de viață, ancorat în universul motociclismului, oferind o experiență distractivă și memorabilă pentru toți cei pasionați de două roți.
* Construirea unor nivele și povestiri unice, inspirate din locații reale din România, pentru a oferi jucătorilor o călătorie autentică și captivantă.
* Crearea unui meniu simplu și intuitiv, împreună cu o interfață de utilizator prietenoasă, pentru a facilita accesul și navigarea în joc, fără obstacole sau dificultăți.
* Dezvoltarea de grafică și muzică originală, contribuind astfel la crearea unei atmosfere vibrante și captivante în cadrul jocului.
* Testarea și optimizarea jocului pentru a asigură o experiență de joc netedă și plăcută pentru toți jucătorii, indiferent de platformă sau dispozitivul utilizat.
* Implementarea unor mecanici de joc inovatoare și captivante, menite să ofere jucătorilor o experiență memorabilă și provocatoare, care să îi țină captivați și implicați în aventura lor.
* Crearea unei povești originale și captivante, menită să fie o sursă de inspirație și amuzament pentru jucători pe întreaga durată a călătoriei lor în căutarea lui Pikachu.

Cele 6 capitole ale acestei lucrări sunt structurate astfel:

Capitolul 1, INTRODUCERE, prezintă direcția lucrării și oferă o perspectivă asupra temei alese, evidențiind-se motivația care a stat la baza alegerii acesteia și stabilindu-se obiectivele specifice ce urmează să fie atinse în lucrarea de licență.

Capitolul 2, FUNDAMENTE TEORETICE, explorează evoluția jocurilor 2D și subliniază aceast progres în industria jocurilor video. De asemenea, se analizează exemple de jocuri similare și influențe în dezvoltarea jocului propus.

Capitolul 3, TEHNOLOGII ȘI INSTRUMENTE UTILIZATE, prezintă platformă Godot 4, limbajul de scripting GDScript și instrumentele de grafică în procesul de dezvoltare a jocului.

Capitolul 4, DEZVOLTAREA JOCULUI "ÎN CĂUTAREA LUI PIKACHU" ÎN GODOT 4, asigură o analiză detaliată a procesului de creare a jocului. Acesta începe cu definirea conceptului și analiza cerințelor, apoi continuă prin pregătirea mediului de lucru și implementarea mecanicilor și interacțiunilor jocului. Capitolul se încheie prin discutarea design-ului, programarea nivelelor, a scenelor, inclusiv integrarea graficii, animațiilor și sunetului în joc.

Capitolul 5, DOCUMENTAȚIE ȘI GHID DE UTILIZARE, oferă instrucțiuni de pornire a aplicației, descrie interfața utilizatorului, controalele jocului și prezintă sfaturi pentru jucători.

Capitolul 6, CONCLUZII, prezintă rezultatele obținute în cadrul lucrării, analizând aici procesul de dezvoltare a jocului și atingerea obiectivelor propuse.

# Fundamente teoretice

Jocurile video au devenit, pentru majoritatea oamenilor, un stil de a-și petrece timpul liber în scopul de a se relaxa sau de a pătrunde într-o lume fictivă unde totul este posibil. Privind evoluția acestora, de la niște simple curiozități, la o adevărată industrie, jocurile video au adus un impact major în spațiul divertismentului.

Pentru a înțelege de ce “În căutarea lui Pikachu” este special, este nevoie de o privire în ansamblu asupra primelor apariții a jocurilor video, a rădăcinilor acestora, dar și a evoluției lor în toți acești ani.

## Contextul jocurilor și evoluția lor

Calculatoarele și jocurile video au mers mână în mână pentru prima dată în anul 1958, la institutul de cercetare al guvernului American, când, pentru ziua porților deschise, inginerul Willy Hihinbotham a dorit să le prezinte vizitatorilor ceva mai captivant. Astfel, el a convertit un osciloscop într-un joc asemănător celui de pinball. El și-a considerat creația că fiind o simulare a jocului de tenis, denumindu-l *“Tennis for Two”(*Fig. 1*)*. Jocul este considerat pionerul jocurilor video [17].

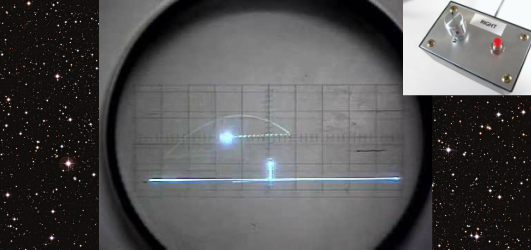


Fig. 1-Tennis for two[12]

În anii ‘70 compania Atari a dominat industria jocurilor video, dezvoltand cele mai de succes jocuri, cel mai important fiind “Pong”(Fig. 2), primul joc care a avut un triumf global. Ideea jocului nu era nouă, fiind foarte similară cu predecesorul lui “Tennis for Two”, însă Atari a integrat computerul împreună cu un ecran într-o cutie cu un spațiu pentru monede, inventand astfel mașina arcade (Fig. 2). De asemenea, “Space Invaders”(Fig. 3) a marcat începutul epocii de aur a sălilor de jocuri, unde adolescenții anilor 1980 își cheltuiau banii de buzunar pe mașinile de jocuri [12].

|  |  |
| --- | --- |
|  |  |
| Fig. 2-Pong cu mașina arcade [ 13] | Fig. 3-Space Invaders[24] |
|  |  |

În 1980 jocurile arcade au rămas în urma pc-urilor și a consolelor, însă au supraviețuit datorită audienței lor specifice și dedicate. În această perioadă au apărut jocurile clasice care dețin popularitate și în prezent, cum ar fi “Pac-Man”(Fig. 4), “Tetris”, “Mario Bros”, “SimCity”, “The legend of Zelda”, “Street Fighter”(Fig. 5). Totodată, au apărut îngrijorări din partea părinților, a politicienilor și a mass-mediei, critici bazate pe modul “sumbru” în care aceste jocuri au abordat violența[12,17].

|  |  |
| --- | --- |
|  |  |
| Fig. 4-Pac-Man[] | Fig. 5-Street Fighter[6] |

Anii 1990 au adus o creștere masivă a industriei, jocurile intrând într-o nouă dimensiune, grafica devenind tridimensională. Sony a lansat PlayStation (Fig. 6) în 1994, o consolă cu un salt imens din punct de vedere tehnic și grafic în comparație cu celelalte console existente. De asemenea, în această perioadă jocurile au început să aiba idei inovatoare, adesea inspirate din istorie [12].



Fig. 6-PlayStation[12]

La începutul anilor 2000, internetul nu era înca răspândit, jucătorii întâlnindu-se la petreceri în LAN pentru a se juca împreună până târziu în noapte. Cel mai popular joc la aceste petreceri era Counter-Strike(2000) (Fig. 7). Odata cu creșterea utilizării internetului, jocurile au început să se mute în online și să se dezvolte în așa fel încât să simuleze viată reală. Inteligența artificială și graficile mai puternice au reușit să creeze lumile jocurilor mult mai realiste [12].



Fig. 7- Counter-Strike[5]

În prezent, jocurile video au devenit o afacere de miliarde de dolari, cu profituri mult mai mari decat cele ale industriei filmului. Studiourile independente au început să apară în număr cât mai mare și să dezvolte jocuri pentru orice tip de platformă: computere, console,tablete, telefoane mobile. „ Grand Theft Auto V”(Fig. 8) și „Red Dead Redemption II”(Fig. 9) sunt doar câteva exemple de jocuri complexe, care evoluează diferit în funcție de deciziile jucătorului și pot fi rejucate de mai multe ori [12].

|  |  |
| --- | --- |
|  |  |
| Fig. 8- Grand Theft Auto V[1] | Fig. 9- Red Dead Redemption II[16] |

## Exemple de jocuri similare și influențe

În acest subcapitol se vor analiza câteva jocuri de tip platformer care au adus influențe semnificative în ceea ce privește dezvoltarea jocului “În căutarea lui Pikachu”. Fiecare dintre acestea dispun de mecanici de joc ingenioase, dar și de povești unice, emoționante și profunde.

### Super Mario Bros

În “Super Mario Bros” (Fig. 10) jucătorul îl controlează pe Mario sau pe Luigi și trebuie să ajungă la sfarșitul nivelului într-o limită de timp, pe parcurs învingând inamici și colectând power-up-uri. Jucătorul trebuie să sară pe inamici că să îi învingă, iar power-up-urile, cum ar fi Super Ciuperca și Floarea de foc sunt ascunse în cutii cu monede și oferă abilități speciale. Scopul final este de a ajunge la steagul lui Browser și de a-l înfrunta în etapele finale ale jocului. După înfrângerea acestuia, jucătorul o salvează pe Prințesa Toadstool [22].



Fig. 10- Super Mario Bros[22]

### Celeste

“Celeste” (Fig. 11) este un joc de aventură în care jucătorul o controlează pe Madeline, o fată care pornește într-o călătorie pentru a escalada misteriosul munte Celeste. Pe parcursul aventurii, ea se descoperă pe sine. Jocul cuprinde 9 capitole de gameplay, fiecare cu mecanici și provocări diferite. Primul capitol introduce jucătorilor controalele de baza, precum săritul și abilitatea de a se cățăra pe pereți, iar la final se introduce abilitatea de dash, un element care oferă un impuls în direcția aleasă. El poate fi folosit de un număr limitat de ori și trebuie reîncărcat prin atingerea solului, colectarea cristalelor de stamină sau prin tranziții de ecran [3].



Fig. 11-Celeste[3]

### Ori and the Blind Forest

“Ori and the Blind Forest”(Fig. 12) este un joc de aventură în care jucătorii preiau controlul asupra lui Ori, un spirit al unui iepure alb, și Sein, “Lumina și ochii” Copacului Spiritelor din pădure. Scopul jocului este de a naviga între platforme și de a rezolva puzzle-uri. Jocul include un sistem de salvare numit “Soul Links” care permite jucătorilor să-și salveze progresul doar dacă jucătorii au suficientă “Energie Spirituală”, o resursă limitată obținută prin trecerea de nivele și prin eliminarea inamicilor. De asemenea, există și un sistem de upgrade care permite jucătorilor să îmbunătățească abilitățile și competențele lui Ori [26].



Fig. 12- Ori and the Blind Forest[17]

### Analiza comparativă și inovațiile aduse de „În căutarea lui Pikachu”

Comparând jocul “În căutarea lui Pikachu” cu cele menționate anterior, se observă că există anumite influențe și analogii. De exemplu, jocurile “Super Mario Bros”, “Celeste” și “Ori and the blind forest” împartășesc cu “În căutarea lui Pikachu” elementele de platformă și explorare. Fiecare dintre acestea pune accentul pe navigarea prin nivele, colectarea de obiecte și interacțiunea cu mediul înconjurător.

“Super Mario Bros” este cunoscut pentru mecanica sa de eliminare a inamicilor, unde jucătorul trebuie să sară pe aceștia pentru a-i doborî . Similar, “În căutarea lui Pikachu” utilizează aceeași abordare în interacțiunea cu inamicii.

Față de “ Ori and the Blind Forest”, unde povestea este una emoționantă, iar explorarea se face într-un univers fantastico-mistic, “În căutarea lui Pikachu” explorează o poveste mai usoară, cu un ton mai aventuros. Jocul își concentrează atenția pe explorare și trecerea peste anumite provocări, oferind astfel o experiență de joc mai umoristică și relaxantă.

# Tehnologii și instrumente utilizate

Tehnologiile folosite în dezvoltarea jocului „În căutarea lui Pikachu” au un rol important, iar în acest capitol se vor prezența aplicații precum Godot 4, Paint.net, Adobe Illustrator, dar și limbajul de programare folosit GDScript.

## Godot 4

Godot 4 este un motor conceput pentru a crea jocuri sau aplicații pe care să le putem lansa pe desktop, mobil sau pe web. Pentru jocul “În căutarea lui Pikachu”, această platformă a oferit un mediu de dezvoltare prietenos, flexibil, cu o ușurință mare de utilizare datorită editoarelor integrate, precum editorul de cod, editorul de animații, editorul de tilemap, debugger-ul și multe altele. Caracterul sau open-source adaugă un plus de valoare deoarece se pot introduce și folosi direct module și funcționalități suplimentare [8].

Versiunea 4 a motorului beneficiază de nenumărate îmbunătațiri aduse performanței, funcționalității și suportului pentru diverse platforme.

Câteva caracteristici principale ale versiunii Godot 4 sunt:

* Îmbunătățiri ale iluminării în mediul 2D: prin utilizarea câmpurilor de distanță din shadere se pot obține efecte vizuale avansate, precum umbre lungi, contururi clare și halouri.
* Opțiuni noi de randare: se permite amestecarea complexă a mai multor CanvasItems care să se suprapună, că de exemplu adăugarea mai multor sprite-uri și amestecarea lor cu fundalul că un singur element.
* Instrumente pentru editarea nivelelor: s-a îmbunătățit editorul de tilemap, oferind mai multa flexibilitate în organizarea și amplasarea plăcilor. Coliziunile, navigarea, punctele de pivotare se pot regla mult mai eficient [8].

## GDScript

Limbajul de programare GdScript este cel ce a fost utilizat pentru dezvoltarea jocului video “În căutarea lui Pikachu”.

Caracteristicile fundamentale ale acestui limbaj sunt:

* Sintaxa simplă: Limbaj de programare orientat pe obiecte cu o sintaxă bazată pe indentare, similară cu limbajul Python.
* Incorporat cu Godot: GDScript este proiectat și integrat cu Godot, asigurând o legătură între codul scris și motorul de joc.
* Gestionarea memoriei: În locul unui colector de gunoi (garbage collector), GDScript folosește un sistem automat de numărare a referințelor pentru a gestiona eliberarea resurselor care nu mai sunt utilizate.

## Paint.NET

Paint.NET este un program de editare gratuit și ușor de folosit, care are că scop crearea și corectarea fotografiilor și a desenelor. Interfața lui prietenoasă și gama largă de instrumente de editare oferă o experiență solidă în modificarea imaginilor fără a avea nevoie de experiență în domeniu [21]. Pentru crearea aplicației “În căutarea lui Pikachu”, Paint.NET a contribuit la modelarea și editarea personajelor, inamicilor și a altor elemente grafice folosite în joc. Acest software a furnizat toate instrumentele necesare pentru a transpune ideea principală și a crea atmosfera dorită.

## Adobe Illustrator

Adobe Illustrator este un editor de grafică vectorială și un software de design dezvoltat și comercializat de Adobe. El oferă unelte avansate pentru desen, ilustrație, design de logo-uri și alte proiecte grafice. De asemenea, fiind bazat pe grafică vectorială, imaginile create sunt întemeiate pe formule matematice și nu pe pixeli, ceea ce le face scalabile, fără pierderea clarității [25]. În dezvoltarea “În căutarea lui Pikachu”Adobe Illustrator a fost folosit pentru a genera imagini pixelate care să fie folosite drept fundal pentru harțile nivelelor jocului.

# Dezvoltarea Jocului "În Căutarea lui Pikachu" în Godot 4

## Analiza cerințelor

În dezvoltarea unui joc video analiza cerintelor reprezintă un pas esențial deoarece în acest mod se pun bazele jocului, dar se stabilesc și direcțiile în care acesta va evolua. Subcapitolul detaliază cerintele funcționale și nefuncționale necesare pentru dezvoltarea jocului “În căutarea lui Pikachu”. Scopul principal este de a crea un joc captivant care intrigă jucătorul să treacă peste toate nivelele.

Gameplay:

* Obiectivul jocului: Jucătorul trebuie să controleze personajul “Anusk” pentru a evita obstacole și inamici, a colecta obiecte și a sari de pe o platformă pe altă.
* Niveluri: Jocul va fi împărțit în mai multe niveluri, fiecare cu dificultăți și obiective diferite.
* Control: Personajul se va controla folosind tastele A,D sau săgețile stânga,dreapta pentru mișcare și Space/W/ săgeată în sus pentru sărituri. Va exista și săritura dublă activată la două acțiuni a butonului pentru sărit.
* Inamici: Fiecare nivel conține inamici diferiți care pot fi evitați sau eliminați

Interfața utilizatorului:

* Ecranul de start: Conține optiuni pentru a începe jocul sau a ieși din aplicatie.
* Ecran de joc: Se afișează viețile rămase.

## Pregătirea mediului de lucru

Prin pregătirea mediului de lucru s-a asigurăt un cadru necesar pentru implementarea funcționalităților și a designului dorit. Astfel, în acest subcapitol, se vor detalia etapele prin care s-a inițializat proiectul în Godot 4.

Așadar, primul pas a constat în crearea unui nou proiect, alegând locația și atribuind numele jocului- “În căutarea lui Pikachu” (Fig. 13).

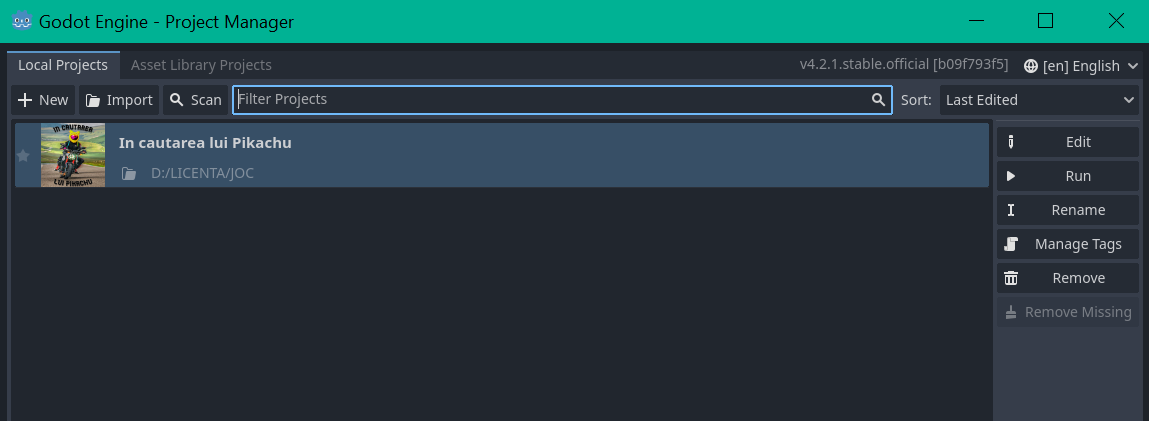


Fig. 13- Creare proiect

Pentru a asigură o experiență de joc placută, rezoluția a fost setată la 1920x1080 din meniul “Project”->”Project Settings”. Această dimensiune a ecranului oferă o claritate a graficii și a detaliilor din joc. Modul „windowed” permite jocul să ruleze într-o fereastră pe desktop, iar scalarea la 4 ajută la optimizarea performanței grafice și asigură că aplicația ramane fluidă și receptivă chiar și pe sisteme mai puțin puternice. De asemenea, am setat modul la “canvas\_items” și aspectul la “keep” pentru ca elementele jocului să se potrivească pe întreaga suprafață a ferestrei și să nu se distorsioneze(Fig. 14).

În ceea ce privește setările din “Input Map”, am creat 3 acțiuni specifice pentru controlul personajului- sus,stânga,dreapta, la care s-au atribuit tastele specifice (Fig. 15).

|  |  |
| --- | --- |
|  |  |
| Fig. 14-Dimensiuni Ecran | Fig. 15-Configurare Taste |

Din sectiunea “Application”-> “Config” ce aparține aceluiași meniu, “Project Settings”, am atribuit proiectului un icon (Fig. 16)- o poza personalizată, creată cu ajutorul aplicației Adobe Illustrator. Simbolul vizual al jocului este important deoarece contribuie la o prima impresie, acesta intrând primul în contact cu utilizatorul.



Fig. 16- Poza personalizată cu Pictograma jocului

Resursele grafice necesare pentru joc, precum sprite-uri pentru personaje, obiecte și fundaluri au fost importate în proiectul Godot. Mai mult, pentru implementarea dialogurilor a fost necesar un asset specific din „AssetLib” denumit „Dialogue Manager”, adăugat ulterior în proiect(Fig. 17).

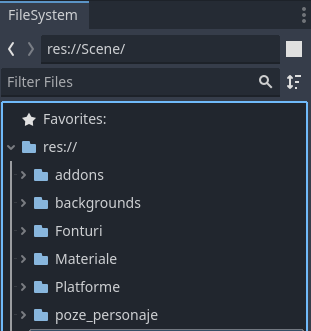


Fig. 17-Resurse

O componentă foarte importantă în pregătirea mediului de lucru este utilizarea încărcărilor automate (“autoload-uri”) (Fig. 18). Acestea pot fi noduri sau scripturi care sunt încărcate automat la pornirea aplicației, asigurând astfel accesul global pentru gestionarea anumitor funcționalități. Pentru proiectul “În căutarea lui Pikachu” s-au utilizat trei încărcări atomate, și anume:

* DialogueManager: asigură afișarea și gestionarea interacțiunilor textuale între personaje;
* Tranzitie: un nod care controleaza tranzițiile între scene;
* ManagerJoc: un script responsabil pentru sistemul de checkpoint-uri al jucătorului.

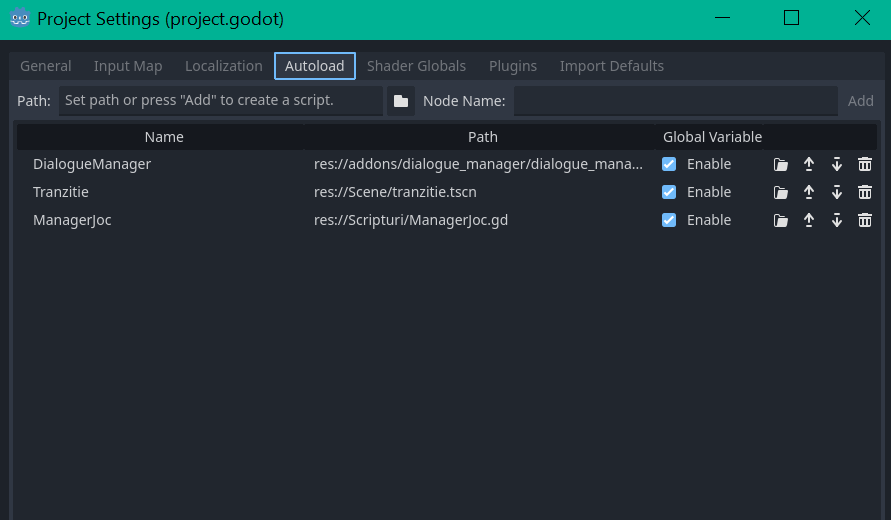


Fig. 18-Sectiunea „Autoload” din godot

De asemenea, pentru a asigura o interacțiune corectă între diversele elemente ale jocului, s-au configurat din meniul „Layers Names”-> „2D Physics”(Fig. 19) următoarele straturi:

* stratul 1- harta: destinat tuturor elementelor ce tin de fundal și de terenurile pe care se desfăsoară acțiunea;
* stratul 2- inamici: aici este plasat tot ceea ce poate rani jucătorul;
* stratul 3- pericol: elementele de pericol, cum ar fi apa sau țepii;
* stratul 4- jucător: este dedicat jucătorului și cuprinde personajul principal și elementele asociate acestuia, precum animațiile și coliziunile. Acest strat permite un control precis al interacțiunilor dintre jucător și mediul înconjurător lui;
* stratul 5- checkpoint: reprezintă punctele de control unde progresul este salvat.

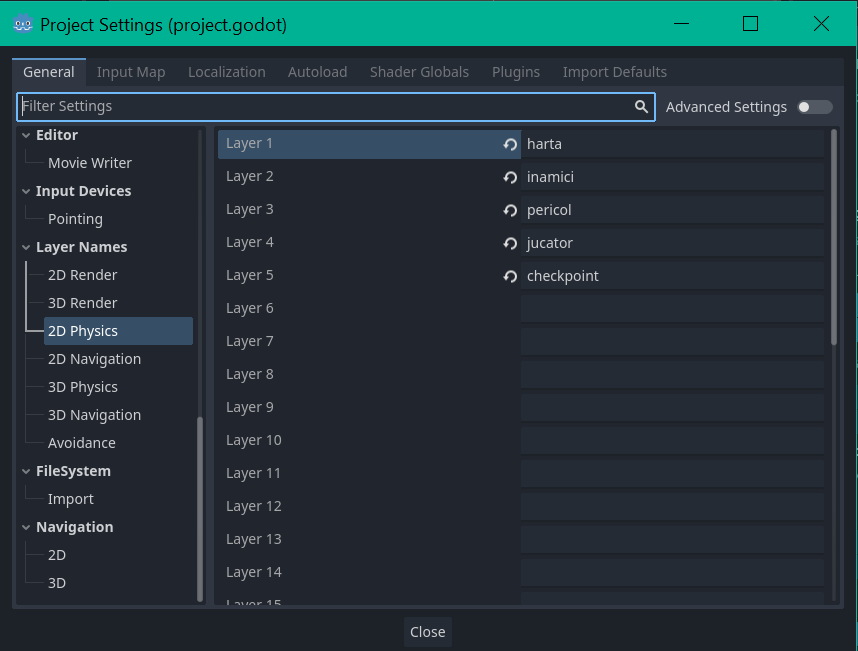


Fig. 19-Configurare layer-uri

## Pagina principală

Scena “main.tscn” reprezintă pagina principală a jocului “În căutarea lui Pikachu”(Fig. 20). Aceasta funcționează că punct central de control, conținând următoarele elemente: un buton “Play” pentru începerea jocului, un buton “Quit game” pentru a închide aplicația și nu în ultimul rand, o imagine de fundal creată în AdobeIllustrator ce reprezintă tematica jocului.

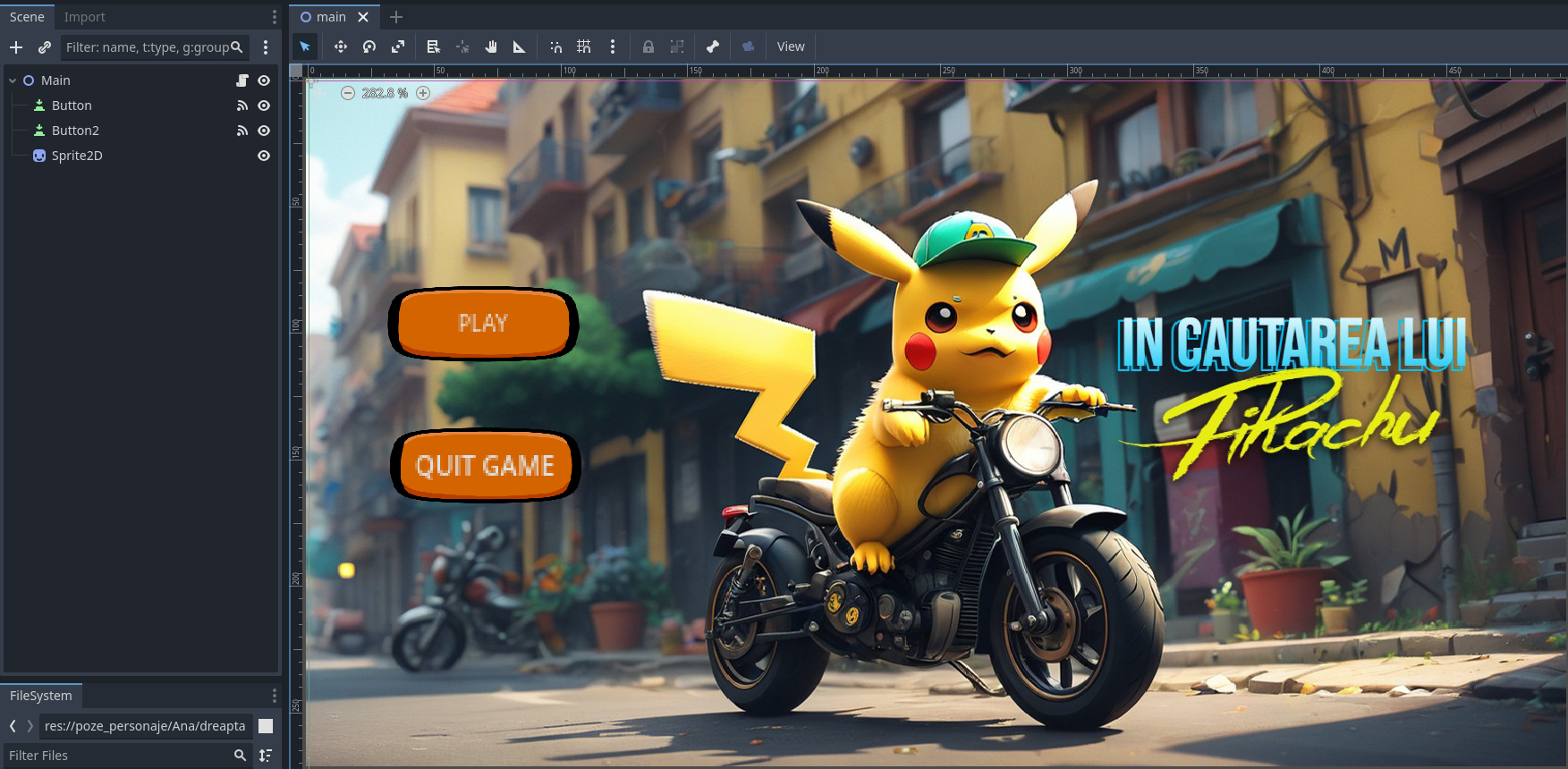


Fig. 20-Meniu Principal

Codul atașat scenei principale cuprinde semnale pentru cele două butoane, atunci când sunt apăsate. Funcția “\_on\_button\_pressed()” asociată butonului “Play” folosește metoda personalizată “Tranzitie” pentru a schimba scena curentă la scena de introducere a jocului.

func \_on\_button\_pressed() -> void:

Tranzitie.schimba\_scena1("res://Scene/CutScenes/intro.tscn")

Cea de-a două funcție “\_on\_button\_2\_pressed()” este apelată atunci când butonul de iesire “Quit Game” este apăsat și se folosește de metoda get\_tree().quit() pentru a închide aplicația.

func \_on\_button\_2\_pressed():

get\_tree().quit()

## Gestionarea tranzițiilor

Pentru a face trecerea de la o scenă la altă mai fluidă, s-a creat nodul “tranzitie.tscn”. În esenta, acesta utilizează un “AnimationPlayer” ce conține patru animații (“RESET”, “incepe\_jocul”, “tranzitie”, “tranzitie\_moarte”), fiecare fiind apelată la un moment specific în joc.

Efectele de apariție treptată și estompare s-au realizat folosind “ColorReact” , iar pentru a afișa texte, precum numele nivelelor, s-a utilizat nodul “Label”.

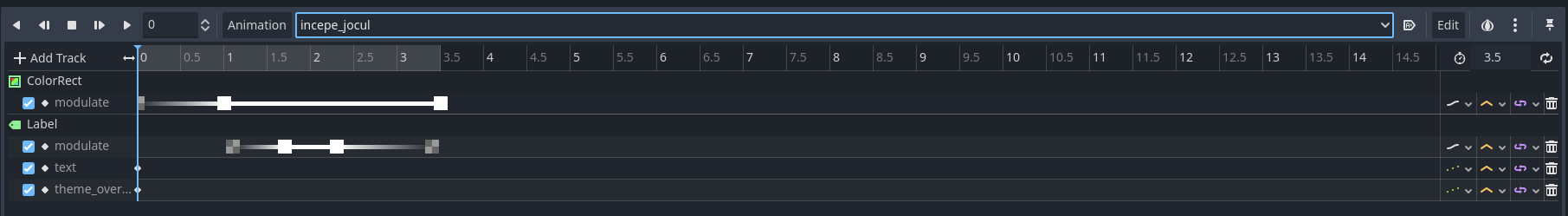


Fig. 21-Animația „incepe\_jocul”

Un exemplu de animație este “incepe\_jocul” (Fig. 21) cu o durata de 3.5 secunde, unde s-au introdus proprietăți ale celor două elemente “ColorReact” și “Label”. S-a creat un efect de apariție treptată atat pentru fundal, cât și pentru text, sporind astfel experiența de joc.

Pe de altă parte, pentru a putea apela ușor și global orice trecere de la o scenă la alta, nodul “tranzitie.tscn” a fost declarat ca fiind încărcat automat (“autoload”) cu numele sugestiv “Tranzitie”.

În ceea ce privește codul ce stă la baza tranzițiilor, acesta conține trei funcții, și anume:

func schimba\_scena\_basic( nume\_scena:String):

animatie.play('tranzitie')

await animatie.animation\_finished

get\_tree().change\_scene\_to\_file(nume\_scena)

animatie.play\_backwards('tranzitie')

func schimba\_scena1( nume\_scena:String):

animatie.play('incepe\_jocul')

await animatie.animation\_finished

get\_tree().change\_scene\_to\_file(nume\_scena)

animatie.play\_backwards('tranzitie')

func dacă\_moare(nume\_scena: String):

animatie.play('tranzitie\_moarte')

await animatie.animation\_finished

get\_tree().change\_scene\_to\_file(nume\_scena)

animatie.play\_backwards('tranzitie')

Fiecare funcție începe cu o animație specifică , asteaptă ca aceasta să se termine,ca mai apoi să se schimbe scena cu cea care urmează, specificată cu ajutorul parametrul “nume\_scena” și să se ruleze animația “tranzitie” în sens invers.

## Grafica și script-urile personajelor din cutscene

Pentru ca personajele să se integreze perfect în povestea jocului, s-au utilizat grafici din pachetul “[Motorcycles Asset Pack](https://pixel-salvaje.itch.io/motorcycles-asset-pack)” achiziționate de pe platforma itch.io. Acestea au fost ulterior editate individual în programul Paint.NET pentru mici schimbări de culoare (Fig. 22).

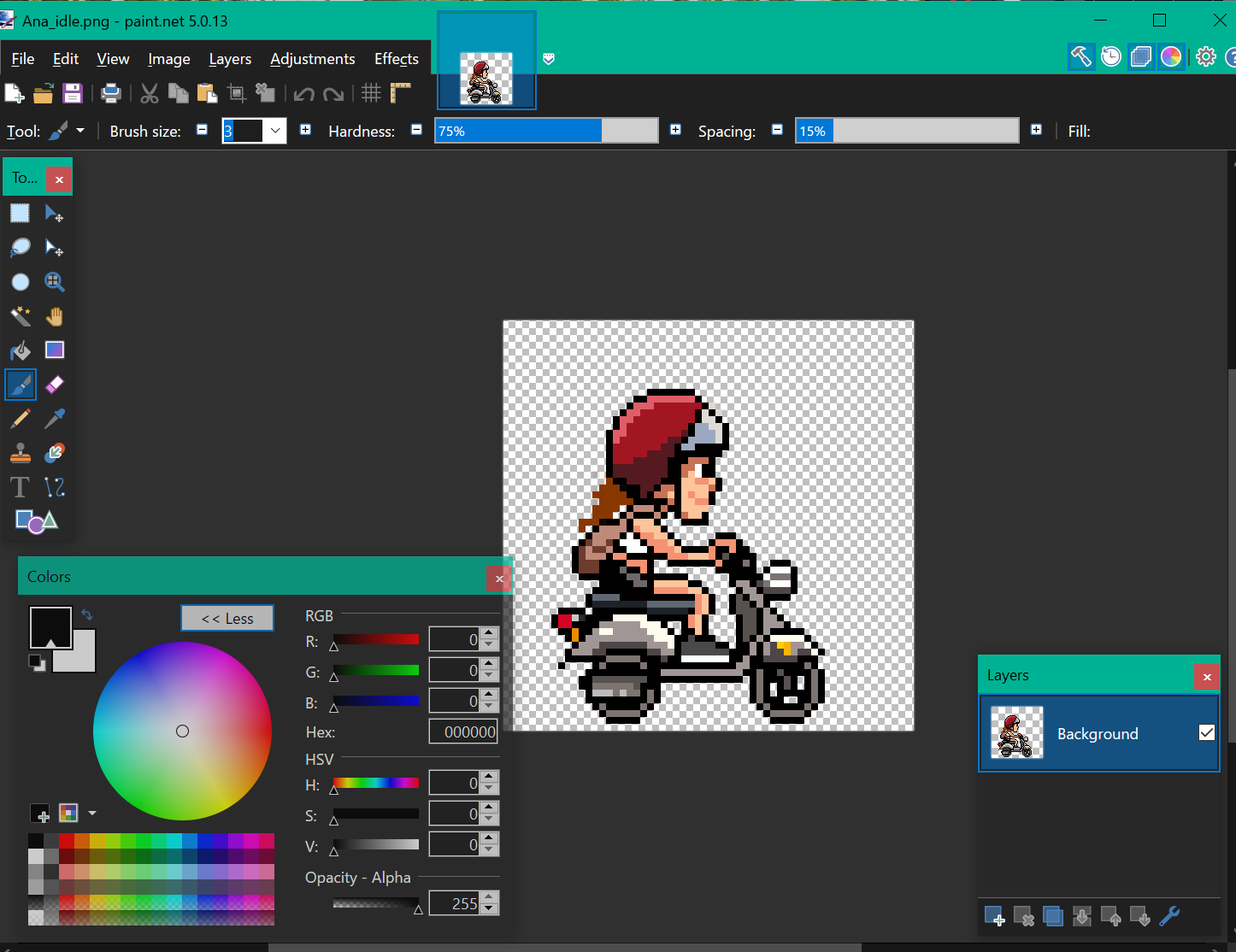


Fig. 22-Editarea Personajului "Ana\_pers"

Mai jos sunt prezentate graficele pentru personajele din cutscene: “Ana\_pers”(Fig. 23), “Bogdan\_pers”(Fig. 24), “Spanu\_pers”(Fig. 27), “Marius\_pers”(Fig. 25), “Petru\_pers”(Fig. 26), “Cosmin\_pers”(Fig. 28):



Fig. 23-Ana\_pers



Fig. 24-Bogdan\_pers

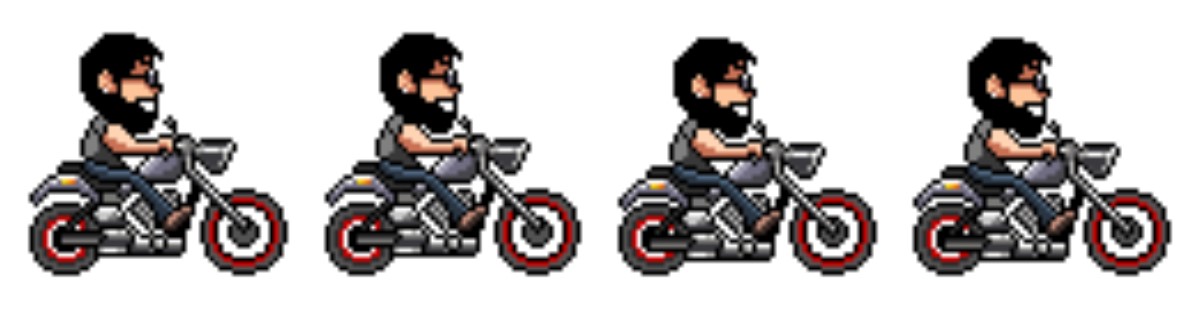


Fig. 25-Marius\_pers



Fig. 26-Petru\_pers



Fig. 27-Spanu\_pers



Fig. 28-Cosmin\_pers

Toate personajele dispun de un “AnimationPlayer” care ajută la gestionarea mișcărilor acestora, cele de bază fiind “stai” și “mergi”. De asemenea, fiecare personaj dispune de două funcții care ajută la schimbarea stărilor lor în cutscene-uri.

func \_ready()-> void:

anim.play("stai")

func play\_anim(nume\_animatie:String) -> void:

anim.play(nume\_animatie)

* funcția \_ready( )- când personajul este plasat în joc, se pornește automat animația “stai”, asigurăndu-se astfel că se pleacă dintr-o stare inițială statică
* funcția play\_anim(nume\_animație:String) – permite să se treacă într-o altă stare prin specificarea numelui animației dorite ca argument.

## Personajul “Anusk” și mecanismul lui de mișcare

“Anusk” este personajul principal pe care jucătorul îl va controla pe tot parcursul firului narativ. El detine o grafică creată cu ajutorul programului Paint.NET, aspectul său fiind inspirat de personajul “Ana\_pers”, însa adaptat pentru a fi transpus fără motocicletă (Fig. 29).



Fig. 29- GRAFICA "ANUSK"

Din punct de vedere structural, el este alcătuit din următoarele noduri: “AnimatedSprite2D” pentru dinamica mișcărilor, “CollisionShape2D” pentru detectarea coleziunii cu alte elemente și “Camera2D” pentru o urmărire mai fluidă a personajului pe ecran (Fig. 30).

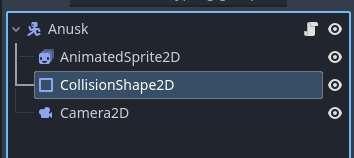


Fig. 30-Noduri Jucător

În ceea ce privește mișcările din nodul “AnimatedSprite2D”, personajul dispune de trei animații, și anume “stai”, “mergi” și “sari”.

Setările pentru coliziune sunt extrem de importante, așadar personajul „Anusk” se va afla pe stratul cu numărul 4 și va putea interacționa prin intermediul măștii sale cu harta, inamicii, pericolele și checkpoint-urile (Fig. 31).

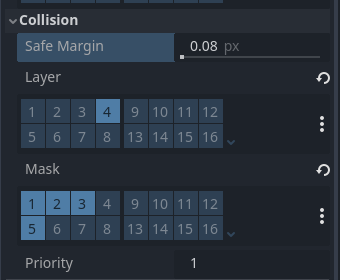


Fig. 31-Configurare straturi jucător

Pentru a implementa mecanismul de mișcare al personajului “Anusk” am creat un script denumit “Anusk.gd” unde am format logica necesară pentru mișcarea și animarea jucătorului în două direcții: stânga și dreapta.

Funcția “\_physics\_process(delta)” este apelată în fiecare cadru pentru a actualiza starea jucătorului astfel:

* inițial se verifică dacă jucătorul se poate controla; dacă nu, procesul se încheie

func \_physics\_process(delta):

if not putem\_controla: return

* se adaugă gravitația dacă jucătorul nu se afla pe sol și se anunță că el este în aer (“in\_aer=true”), în caz contrar, se resetează variabila pentru săritura dublă, iar dacă jucătorul a fost în aer, la atingerea solului acesta va avea animația “stai”.

if not is\_on\_floor():

velocity.y += gravity \* delta

in\_aer=true

else:

a\_sarit\_dublu=false

if in\_aer==true:

anusk.play("stai")

blocheaza\_anim=false

in\_aer=false

* pentru gestionarea săriturii simple și duble: dacă jucătorul este pe sol se aplează funcția specifică “sari()”, în caz contrar, dacă nu s-a efectuat o săritura dublă înainte, se actualizează viteza pe verticală pentru săritura dublă.

if Input.is\_action\_just\_pressed("sus"):

if is\_on\_floor():

sari()

elif not a\_sarit\_dublu:

velocity.y=salt\_dublu

a\_sarit\_dublu=true

* pentru mișcarea pe orizontală: direcția de mișcare se obtine de la intrarile tastaturii-stânga,dreapta,sus,jos, configurate la începerea proiectului. Dacă există o directie, viteza se va actualiza în funcție de intrarea data de tastatură și viteza jucătorului. În caz contrar, dacă nu s-a apăsat nicio tastă, se reduce viteza personajului pentru a oferi o tranziție liniară de la mișcare la oprire.
* la final, se aplează trei funcții: prima pentru a aplică mișcarea jucătorului, a două pentru a actualiza animația în funcție de mișcarea curentă și a treia pentru ajustarea direcției sprite-ului în funcție de direcția de mișcare .

move\_and\_slide()

actualizează\_animatia()

directie\_mers()

directie = Input.get\_vector("stanga", "dreapta","sus","jos")

if directie:

velocity.x = directie.x \* viteza

else:

velocity.x = move\_toward(velocity.x, 0, viteza)

Funcția “actualizeaza\_animatia()” se ocupă cu schimbarea animației personajului în funcție de starea sa curentă: dacă personajul se mișcă pe orizontală, este redata animația “mergi”, iar dacă stă pe loc, se aplică animația “stai”.

func actualizeaza\_animatia():

if not blocheaza\_anim:

if directie.x!=0:

anusk.play("mergi")

else:

anusk.play("stai")

Funcția “directie\_mers()” verifică în ce directie să se orienteze sprite-ul atunci când personajul se mișcă la stânga sau la dreapta. În această metodă se folosește proprietatea “flip\_h” cu valorile sale “true/false”.

func directie\_mers():

if directie.x>0:

anusk.flip\_h=false

elif directie.x<0:

anusk.flip\_h=true

## Gestionarea dialogurilor

Dialogue Manager

DialogueManager este un asset creat de Nathan Hoad, disponibil în sectiunea “AssetLib” de pe platformă Godot 4. El facilitează implementarea și gestionarea dialogurilor, ajutând astfel creatorii de aplicații să realizeze interacțiuni mai complexe, fiind și ușor de utilizat.

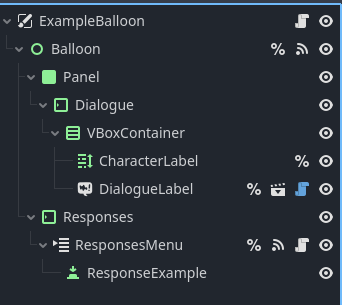


Fig. 32-Componentele balonului pentru dialog

În cadrul jocului “În căutarea lui Pikachu”, Dialogue Manager a fost utilizat în momentele cinematice pentru a crea dialogurile dintre personaje. Asupra componentelor balonului de dialog (Fig. 32) s-au adus următoarele modificări:

* pentru nodul “Panel”- s-a schimbat culoarea de fundal (Fig. 33) și s-a modificat poziția acestuia pentru a fi pe centrul ecranului, utilizând ancore customizate (Fig. 34).
* pentru “DialogueLabel”- s-a adăugat fontul pentru text numit “poco” și s-a redimensionat fontul la 15 px pentru a fi mai vizibil (Fig. 35).

|  |  |  |
| --- | --- | --- |
|  |  |  |
| Fig. 33-Configurarea fundalului (Panel) | Fig. 34-Configurarea Ancorelor (Panel) | Fig. 35-Configurarea DialogueLabel |

Replicile s-au preluat din fișierul “dialog\_intro.dialogue” cu ajutorul managerului de dialog global “dialogue\_manager.gd” ce conține funcții specifice pentru acestea. Un dialog în Godot poate conține mai multe elemente, precum numele dialogului, replicile personajelor sau apeluri de funcții (Fig. 36).



Fig. 36-Dialogurile jocului

În aplicația “În căutarea lui Pikachu”, funcția “show\_exemple\_dialogue\_balloon” din scriptul “dialogue-manager.gd” a fost folosită pentru a gestiona afișarea corectă a dialogurile dintre personaje. Codul din GDScript al acestei funcții este:

func show\_example\_dialogue\_balloon(resource: DialogueResource, title: String = "", extra\_game\_states: Array = []) -> CanvasLayer:

var balloon: Node = load(\_get\_example\_balloon\_path()).instantiate()

get\_current\_scene.call().add\_child(balloon)

balloon.start(resource, title, extra\_game\_states)

return balloon

Aici, se încarcă și instanțiază un balon utilizând resursa “DialogueResource” (cea care colectează și organizează liniile de dialog). Acest balon de dialog este adăugat apoi la scena curentă și inițializat prin intermediul apelului liniei “*balloon.start(resource, title,extra\_game\_states)”* care specifică resursa, titlul dialogului care se dorește a fi afișat și niște stări suplimentare de joc. În final, funcția returnează obiectul “balloon”, permițând în continuare manipularea și gestionarea acestuia.

## Momente cinematice

Momentele cinematice (cutscene-urile) din jocul “În căutarea lui Pikachu” au un rol foarte important deoarece ajută la introducerea utilizatorului în lumea poveștii, creionând astfel contextele pornirii în această aventură. Ele sunt implementate cu ajutorul nodurilor de tip “AnimationPlayer” care ajută la crearea dinamicii personajelor, dar se folosește și de assetul “Dialogue Manager” pentru gestionarea dialogurilor dintre personaje. În acest subcapitol, se vor detalia cele două cutscene-uri din joc, și anume “intro.tscn” și “lvl1\_cutscene.tscn”.

### Momentul cinematic “Intro”

În această scenă se prezintă întâlnirea personajelor la cafeneaua din locația “Sala Sporturilor Olimpia”, înainte de a pleca într-o excursie cu motocicletele. În structura sa, pe lângă elemente precum ”fundal”, “pamant” și personaje, se găsește un “AnimationPlayer” cu 2 animații: “scena1”(Fig. 37) și “Plecare”(Fig. 38).

Fiecare animație este construită folosind chei pentru pozițiile personajelor (“position”). În plus, funcțiile “play\_anim()” s-au folosit pentru a gestiona execuția animațiilor specifice, precum “stanga\_stai”, “stanga\_mergi”, “mergi”. Cele două elemente, “position” și “functions”, au contribuit la sincornizarea și controlul precis al animațiilor personajelor din joc, asigurând astfel o acțiune fluentă.

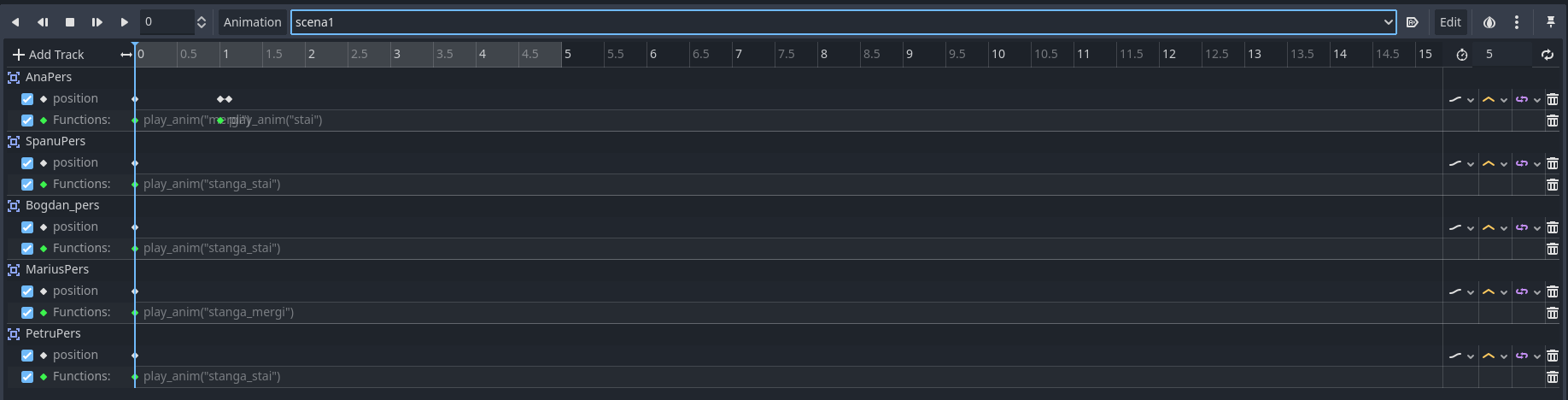


Fig. 37-Animația "Scena1"

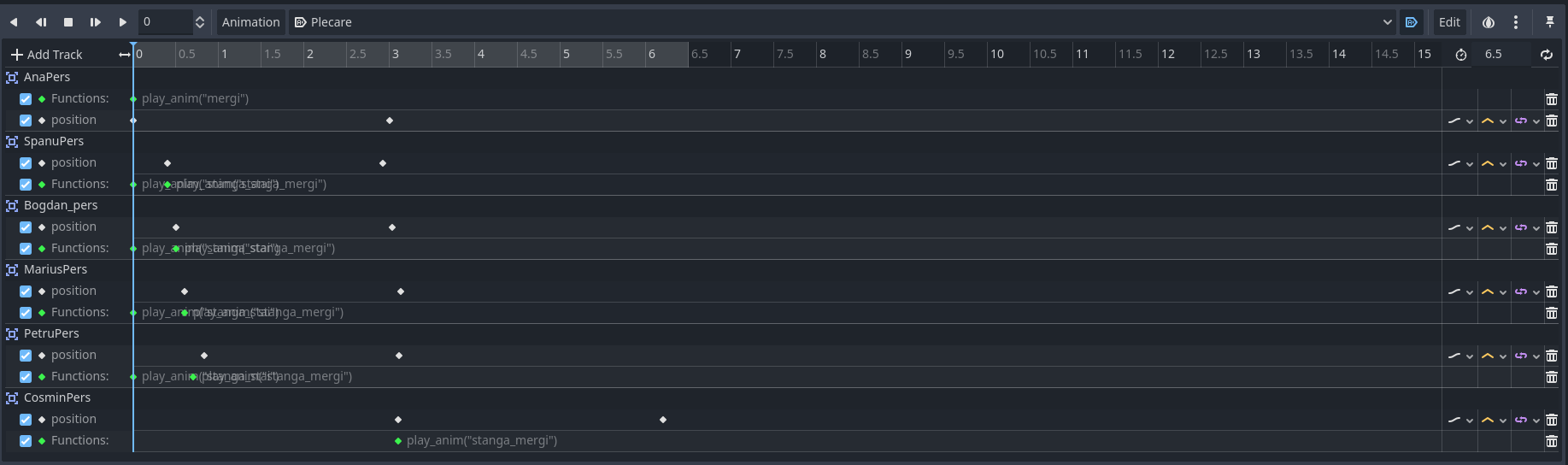


Fig. 38-Animația "Plecare"

Pentru a redă animațiile, nodului “intro.tscn” i s-a atașat scriptul “intro.gd”, prezentat mai jos:

func \_ready():

$AnimationPlayer.play("scena1")

await get\_tree().create\_timer(1.2).timeout

DialogueManager.show\_example\_dialogue\_balloon(load("res://Materiale/dialog/dialog\_intro.dialogue"), "inceput")

func dialog\_term():

$AnimationPlayer.play("Plecare")

await get\_tree().create\_timer(3.1).timeout

DialogueManager.show\_example\_dialogue\_balloon(load("res://Materiale/dialog/dialog\_intro.dialogue"), "Cosmin")

await get\_tree().create\_timer(3.4).timeout

Tranzitie.schimba\_scena\_basic("res://Scene/CutScenes/lvl1\_cutscene.tscn")

* în funcția “\_ready()” se redă animația “scena1”. După un timp specificat (1.2 secunde) este afișat și dialogul cu numele “început” cu ajutorul managerului de dialog ce conține funcția “show\_exemple\_dialogue\_balloon”.
* după terminarea dialogului, se aplează funcția “dialog\_term()” în care personajele se pregătesc de plecare ( animația “Plecare”), se afișează dialogul cu numele “Cosmin” iar apoi se trece la următorul moment cinematic (“lvl1\_cutscene.tscn”) prin intermediul unei tranziții.

### Momentul cinematic “lvl1\_cutscene”

În momentul cinematic “lvl1\_cutscene”, personajele au plecat în căutarea lui Pikachu, însă dau de un drum forestier, imposibil de pătruns cu vehiculele. Motorul personajului Petru se strică, astfel că Ana decide să plece în căutarea de elemente ajutătoare, lăsându-i pe ceilalți să păzească motorul ei.

În cadrul structurii acestei scene (Fig. 39) se regăsesc componente precum: un nod “AnimationPlayer”, două noduri de tip “TileMap” pentru fundal și drum, personajele “AnaPers”, “BogdanPers”, “MariusPers”, “SpanuPers”, ”PetruPers”, dar și un nod “Camera2D” adăugat pentru a urmări mișcarea personajului “AnaPers” în mod dinamic.

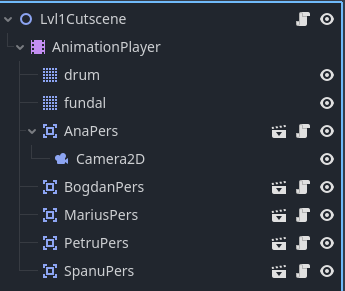


Fig. 39-Structura scenei "lvl1\_cutscene"

“AnimationPlayer”-ul din această scenă conține animația cu numele “animatie1” (Fig. 40) care are o durata de 18 secunde. Aici, dinamica este realizată cu ajutorul cheilor de animație și a funcțiilor fiecărui personaj.

Cheile de animație au fost folosite pentru a defini stări ale elementelor în anumite momente ale acțiunii. Astfel, s-au adăugat poziții ale personajelor pentru a crea mișcarea lor dintr-un punct în altul. Prin intermediul funcției “play\_anim()” personajele execută animațiile corespunzătoare date ca parametru, animații precum “mergi”, “stai”, “oprit\_motor”.

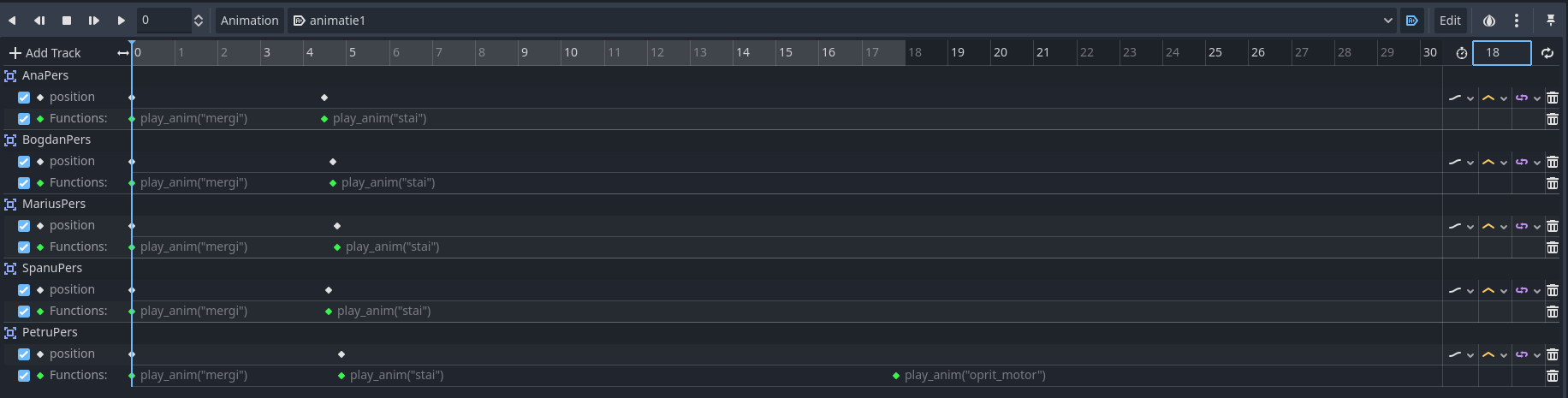


Fig. 40-Animația "animatie1"

În ceea ce privește codul din spatele scenei “lvl1\_cutscene”, el începe prin pornirea animației “animatie1”. Pentru a sincroniza oprirea mișcării personajelor cu apariția dialogului cu titlul “lvl1\_cutscene”, s-a utilizat un temporizator cu o întârziere de 5 secunde. Apoi, prin funcția “lvl1()” s-a creat o tranziție către primul nivel al jocului.

func \_ready():

$AnimationPlayer.play("animatie1")

await get\_tree().create\_timer(5).timeout

DialogueManager.show\_example\_dialogue\_balloon(load("res://Materiale/dialog/dialog\_intro.dialogue"), "lvl1\_cutscene")

func lvl1():

Tranzitie.schimba\_scena\_basic("res://Scene/levels/lvl1.tscn")

## Sistemul de viață

În jocul “În căutarea lui Pikachu” sistemul de viață este reprezentat printr-un număr de inimi afișate în colțul din stânga al ecranului la începutul fiecărui nivel. Acest sistem este esențial pentru a oferi o fluiditate jocului, astfel încât, atunci când jucătorul pierde toate viețile, el va fi nevoit să reia nivelul de la început, pierzând progresul.

### Grafica inimilor și actualizarea stării lor

Grafica inimilor a fost realizată în Paint.NET. Fiecare inimă are două stări: plină sau goală. Pentru a gestiona dispariția unei inimi atunci când aceasta se pierde, s-a folosit un nod “Panel” cu un “Sprite2D” care conține o textură cu trei cadre (frames) (Fig. 41). În implementarea actuală, s-au folosit doar cadrul al doilea pentru inimă plină și cadrul al treilea pentru inimă goală.



Fig. 41- Grafica inimilor

Cu ajutorul funcției “update(tip\_inima:bool)” din scriptul atașat nodului “Panel”, se actualizează grafica unei inimi individuale, setând starea acesteia în funcție de parametrul “tip\_inima”:

func update(tip\_inima:bool):

if tip\_inima: inima.frame=2

else: inima.frame=1

### Alinierea inimilor

Pentru a alinia inimile într-un mod ordonat pe orizontală, s-a creat un nod de tip “HBoxContainer”. Acesta are capacitatea de a gestiona automat spațiul dintre inimi, fără a avea nevoie de ajustări manuale.

Codul din spatele containerului se ocupă de afișarea și actualizarea setului de inimi, astfel:

* funcția “seteaza\_max\_inimi(max\_inim:int)” are rolul de a inițializa și afișa numărul maxim de inimi. Se parcurge un interval ( de la 0 la “max\_inim”), iar pentru fiecare iterație se instanțiază o noua inimă și se afișează pe ecran.

func seteaza\_max\_inimi(max\_inim:int):

for i in range(max\_inim):

var inimi=nrInimi.instantiate()

add\_child(inimi)

* funcția “updateInimi(viata\_curenta:int)” se ocupă de actualizarea grafică a vieții. Parametrul “viata\_curenta” reprezintă numărul de inimi pline pe care le are jucătorul în acel moment. Pentru a seta starea acestora, funcția parcurge mai întâi lista de inimi până la “viata\_curenta”, unde fiecare este afișată ca fiind plină. Apoi, de la valoarea variabilei “viata\_curenta” până la numărul total de inimi, se setează fiecare ca fiind goală. Ambele acțiuni aplează funcția “update” din nodul “Panel”, funcție menționată anterior.

func updateInimi( viata\_curenta:int):

var inimi= get\_children()

for i in range(viata\_curenta):

inimi[i].update(true)

for i in range(viata\_curenta, inimi.size()):

inimi[i].update(false)

### Implementarea sistemului în nivele

Pentru fiecare nivel al jocului, containerul de inimi este integrat ca un nod subordonat al tipului “CanvasLayer”. Cu ajutorul acestui părinte, se garantează o afișare corectă a elementelor grafice în cadrul scenei.

Codul de mai jos este responsabil pentru gestionarea sistemului de viață al jucătorului și este aplicat în funcția “\_ready()” a fiecărui nod nivel:

func \_ready():

container\_inimi.seteaza\_max\_inimi(jucator.max\_inimi)

container\_inimi.updateInimi(jucator.inimi)

jucător.viată\_schimbata.connect(container\_inimi.updateInimi)

În aceasta funcție există trei acțiuni importante:

* setarea maximului de inimi: se folosește funcția “seteaza\_max\_inimi()” a nodului “container\_inimi” pentru a inițializa viata maxima a jucătorului
* actualizarea starii inimilor: se actualizează starea inimilor în funcție de variabila “jucator.inimi”
* conectarea semnalului: se conectează semnalul “viata\_schimbata” al nodului “jucator” la metoda “container\_inimi.updateInimi”. Acest lucru asigură că orice modificare a vieții jucătorului va actualiza și afișajul inimilor în “container\_inimi”

De asemenea, pentru a permite accesul și configurarea usoară a valorilor legate de numărul de vieți a jucătorului, s-au inițializat în scriptul “anusk.gd” două variabile și anume:

@export var max\_inimi: int

@onready var inimi: int= max\_inimi

Prin utilizarea directivei “@export” pentru variabila “max\_inimi” s-a simplificat configurare numărului de vieți deoarece aceasta a devenit accesibilă din afara scriptului.

### Pierderea vieților

În jocul “În căutarea lui Pikachu”, pierderea vieților este tratată în funcția “pericol()” ce aparține scriptului “Anusk.gd”. Aceasta secvență de cod este apelată atunci când jucătorul este atacat de inamici sau cade de pe platforme.

Logica din spatele funcției “pericol()” este:

* numărul de vieți scade cu 1 atunci când jucătorul intră în pericol;
* dacă jucătorul nu mai dispune de nicio viață, se resetează numărul de inimi la cel maxim și se redă o tranzitie specifică, ca mai apoi să se reia nivelul;
* în caz contrar, se emite un semnal “viata\_schimbata”, se opreste controlul asupra personajului, se asteaptă un timp (0.5 secunde), iar mai apoi se resetează poziția personajului la ultimul checkpoint atins.

func pericol():

inimi -= 1

if inimi<=0:

inimi=max\_inimi

Tranzitie.daca\_moare("res://Scene/levels/lvl1.tscn")

else:

viata\_schimbata.emit(inimi)

putem\_controla=false

await get\_tree().create\_timer(0.5).timeout

reseteaza\_poz()

## Checkpoint-uri

În jocurile video, checkpoint-ul reprezintă o unealtă foarte des utilizată. Ele marchează locații specifice, salvând astfel progresul jucătorului și permițându-le să revină în aceste puncte în caz de pierdere a vieților sau în alte situații critice.

### Implementarea checkpoint-urilor

În ceea ce privește jocul ”În căutarea lui Pikachu”, checkpoint-urile au următoarea structură: un nod “AnimatedSprite2D” pentru grafică și un “CollisionShape2D” pentru detectarea coliziunilor (Fig. 42).

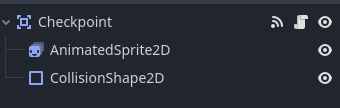


Fig. 42- Structura Checkpoint-urilor

Grafica pentru aceste puncte de control face parte din pachetul [“Cats || Pixel Asset Pack”](https://pop-shop-packs.itch.io/cats-pixel-asset-pack) ce este disponibil gratuit pe platforma itch.io. Nodul „AnimatedSprite2D” conține două animații: “activ” și “inactiv”. Aceste animații sunt folosite pentru a indica starea checkpoint-ului, oferind astfel și un feedback jucătorului cu privire la activarea punctului de control: când este inactiv, pisica stă jos și se uită în stânga, iar când se activează, pisica se ridică în picioare (Fig. 43).

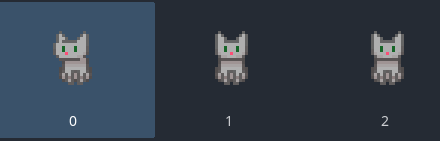


Fig. 43-Grafica Checkpoint

Pentru a gestiona funcționalitățile checkpoint-urilor a fost creat scriptul “checkpoint.gd”. În acest script, se afla clasa “Checkpoint” care cuprinde două funcții:

Funcția activează\_check() setează punctul de control curent în “ManagerJoc”, pornește animația de activare a pisicii și marchează checkpoint-ul ca fiind activat.

func activeaza\_check():

ManagerJoc.checkpoint\_curent= self

pisica.play('activ')

check\_activat= true

Funcția “on\_body\_entered()” este un semnal care se declanșează atunci când un corp intră în zona de coliziune atasată punctului de control, astfel, dacă acel corp este jucătorul și checkpoint-ul nu este deja activat, se aplează funcția “activeaza\_check”.

func \_on\_body\_entered(body):

if body is Jucator && !check\_activat:

activeaza\_check()

### Reapariția și resetarea poziției jucătorului

Pentru a permite accesul și actualizarea punctelor de control de oriunde din joc, acestea sunt gestionate cu ajutorul scriptului de încărcare automată “ManagerJoc.gd”. Codul din spatele scriptului asigură funcționalitatea de reapariție în joc, adică reîncărcarea jucătorului de la ultimul checkpoint activat.

Pentru început s-au declarat două variabile “checkpoint\_curent” și “jucator”, fiecare făcând referire la câte o clasă: “Checkpoint”, respectiv “Jucator”.

var checkpoint\_curent: Checkpoint

var jucator: Jucator

Funcția “respawneaza()” este responsabilă pentru reapariția jucătorului în ultimul punct de control activat. Ea verifică dacă există un checkpoint curent activat, iar în caz afirmativ, se setează poziția jucătorului la poziția globală a punctului de control curent.

func respawneaza():

if checkpoint\_curent != null:

jucator.position=checkpoint\_curent.global\_position

De asemenea, pentru ca punctele de control să funcționeze, a trebuit să se creeze în “Anusk.gd” o funcție care să reseteze poziția jucătorului atunci când acesta pierde o viață. Astfel, funcția “resetează\_poz()” este esențială pentru mecanica jocului deoarece prin intermediul ei se aplează metoda “respawneaza()”, menționată anterior, și se preia din nou controlul asupra personajului.

func reseteaza\_poz():

ManagerJoc.respawneaza()

putem\_controla=true

## Zona periculoasă

Conceptul de “zonă periculoasă” reprezintă o caracteristică de design a nivelului, ce impune riscuri și provocări suplimentare pentru jucător. Aceste zone sunt marcate de un nod de tip “Area2D” și numeroase “CollisionShape2D” care definesc regiuni ce determină pierderea unei vieți atunci când jucătorul intră în contact cu acestea.

Codul asociat zonei periculoase utilizează un semnal pentru a detecta dacă un corp intră în zona de coliziune:

func \_on\_body\_entered(body):

if body is Jucator:

body.pericol()

Dacă acel corp face parte din clasa “Jucător”, semnalul activează funcția “pericol()” care gestionează logica pierderii unei vieți și reapariția jucătorului.

|  |  |
| --- | --- |
| Fig. 44- Zona pericol a nivelului 1 | Fig. 45-Zona pericol a nivelului 2 |
|  |  |

În scena “lvl1.tscn”, zona periculoasă este reprezentată de nodul “ZonaPericol” ce conține nouă “CollisionShape2D”(Fig. 44).Acestea sunt folosite pentru a defini zonele de apă în care personajul nu poate supraviețui, fiecare având o forma rectangulară pentru reprezentarea coliziunii.

În scena “lvl2.tscn”, zona periculoasă conține cinci noduri “CollissionShape2D” (Fig. 45) și este configurată astfel: trei pentru zonele cu apă, unul pentru țepi și unul pentru zonele de înalțime( zona clădirilor pe care personajul trebuie să le traverseze).

## Crearea și configurarea inamicilor

Inamicii sunt niște elemente esențiale care adaugă fiecărui nivel o dinamică aparte cu noi provocări. În jocul “În căutarea lui Pikachu” există două tipuri de inamici: păianjeni (prezenți în nivelul 1) și șobolani (prezenți în nivelul 2).

### Structura nodurilor inamicilor

În Fig. 46, structura nodurilor pentru fiecare inamic (Fig. 46) este organizată într-o ierarhie clară și funcțională. Nodul principal, denumit în funcție de tipul inamicului, include următoarele noduri esențiale pentru o funcționare corectă a adversarului:

* “AnimatedSprite2D” care gestionează cinci animații specifice mișcărilor inamicului ( atac\_dreapta, atac\_stânga, mers\_dreapta, mers\_stânga, moare)
* “CollisionShape2D” definește zona de coliziune. Forma și dimensiunea acestui nod au fost configurate în așa fel încât să corespundă cu conturul vizual al inamicului, alegându-se o formă rectangulară.
* “Area2D” pentru detectarea zidului: o zona specială pentru identificarea coliziunii cu pereții
* “Area2D” pentru detectarea jucătorului și anuntarea eliminării inamicului
* “Area2D” pentru zona care, atunci când este pătrunsă, declanșează acțiuni de atac asupra jucătorului.

Toate aceste noduri asigură o interacțiune dinamică și precisă a inamicilor, permițându-le astfel să reacționeze la mediu și la prezența jucătorului în scena.

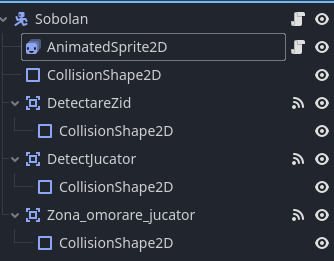


Fig. 46-Noduri inamic

### Implementarea comportamentului inamicilor

În ceea ce privește implementarea concretă a acțiunilor inamicilor, acestea au fost realizate cu ajutorul scriptului “comportament\_inamic.gd”.

Funcția “\_process(delta)” se ocupă de gestionarea comportamentului adversarilor astfel: inițial se verifică dacă inamicul este eliminat sau nu. Dacă da, se așteaptă 1.6 secunde pentru a se realiza animația “moare” și apoi se elimina din joc.

func \_process(delta):

if este\_mort:

await get\_tree().create\_timer(1.6).timeout

queue\_free()

Pe ramura “else” se verifică dacă inamicul atacă, în caz afirmativ, se asteaptă 0.8 secunde pentru a permite derularea animației “ataca”. Apoi se adaugă gravitația dacă acesta nu este pe pământ, se aplică viteza pe orizontală în funcție de parametrul “delta” și se activează animația specifică pentru direcția de deplasare a inamicului (“mers\_stanga” sau “mers\_dreapta”). La final, pentru a aplica mișcarea s-a utilizat metoda “move\_and\_slide()”.

else: if ataca\_jucator:

await get\_tree().create\_timer(0.8).timeout

ataca\_jucator=false

if not is\_on\_floor():

velocity.y += gravity \* delta

velocity.x= viteza\_inamic\*delta

if viteza\_inamic>0:

animatie.play("mers\_dreapta")

elif viteza\_inamic<0:

animatie.play("mers\_stanga")

move\_and\_slide()

Funcția “\_on\_detectare\_zid\_body\_entered(\_body)” se declanșează atunci când inamicul intră în coliziune cu un perete. Aici, viteza sa pe orizontală se schimba în direcția opusa celei în care se mișca anterior.

func \_on\_detectare\_zid\_body\_entered(\_body):

viteza\_inamic=-viteza\_inamic

Funcțiile “\_on\_zona\_omorare\_jucător\_body\_entered(body)” și “ataca()” fac parte din logica de atac a inamicilor, iar mai jos vor fi explicate:

* “\_on\_zona\_omorare\_jucător\_body\_entered(body)” : atunci când corpul jucătorului intră în zona de detecție a inamicului, funcția dezactivează capacitatea personajului de a mai putea fi controlat, apoi inamicul inițiaza atacul, apelând funcția “ataca()”. După o scurta pauză de 0.5 secunde, personajul se poate controla și se aplează funcția “pericol()” despre care s-a vorbit în subcapitolul “Sistemul de viată”.

func \_on\_zona\_omorare\_jucator\_body\_entered(body):

if body is Jucător:

jucator=body

jucator.putem\_controla=false

ataca()

await get\_tree().create\_timer(0.5).timeout

jucator.putem\_controla=true

body.pericol()

* “ataca()”: aici se gestionează efectiv mecanica de atac a inamicului. Funcția semnalizează faptul că adversarul atacă și inițiază animația corespunzatoare în funcție de poziția jucătorului (dacă acesta se află în partea stânga sau partea dreapta a inamicului)

func ataca():

ataca\_jucator=true

velocity.x=0

if jucator.position.x<position.x:

animatie.play("atac\_stanga")

else:

animatie.play("atac\_dreapta")

În ceea ce privește gestionarea eliminării inamicului, funcțiile “\_on\_detect\_jucator\_body\_entered(body)” și “moare()” sunt fundamentale pentru aceasta parte.

* funcția “\_on\_detect\_jucător\_body\_entered(body)” este activată ca un semnal atunci când jucătorul intră în zona de detecție și aplează funcția “moare()”
* Funcția denumită “moare()” preia controlul total asupra inamicului, oprindu-i mișcarea și eliminând toate obiectele asociate coliziunilor prin metoda “queue\_free()”. În timp ce adversarul este eliminat, se activează animația “moare” care oferă o reprezentare vizuală a momentului eliminării lui. De asemenea, variabila “este\_mort” este setată la “true”, marcându-i inamicului incapabilitatea de a interacționa ulterior în scenă.

func \_on\_detect\_jucator\_body\_entered(body):

if body is Jucator:

moare()

func moare():

velocity.x=0

coleziune\_inamic.queue\_free()

animatie.play("moare")

este\_mort=true

coleziune\_detectJuc.queue\_free()

coleziune\_atac.queue\_free()

### Grafica inamicilor

Inamicii sunt reprezentați printr-un design în stil “pixel art”, utilizând sprite-uri detaliate ce conduc la animații fluide. În acest mod se adaugă un strop de realism și expresivitate mișcărilor.

Fiecare inamic este dotat cu un nod “AnimatedSprite2D” care se ocupă cu gestionarea diferitelor animații ale acestora, inclusiv mișcări precum mersul în stânga și în dreapta, atacuri specifice și animații de eliminare.

Pentru nivelul 1 al jocului, am integrat inamicul “Paianjen” care este reprezentat vizual cu ajutorul sprite-urilor din pachetul “2D Pixel Art Spider Sprites” procurate de pe site-ul itch.io. Acestea au fost utilizate pentru a oferi o prezență adecvată pentru contextul mediul de joc al nivelului 1 (pădurea).

Mai jos este prezentat spritesheet-ul folosit pentru “Paianjen”:

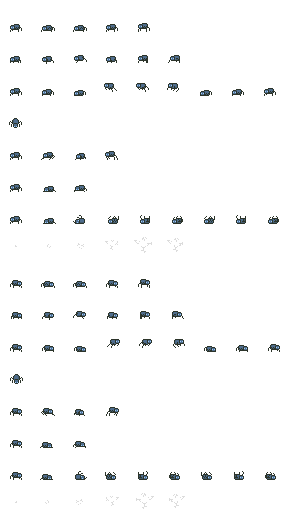
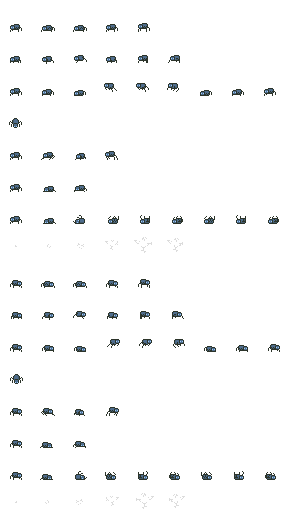


Fig. 47-Sprite-ul "Paianjen"

În nivelul 2 al jocului “În căutarea lui Pikachu”, am optat pentru inamicul “Sobolan”, utilizând grafica din pachetul “2D Pixel Art Rat Sprites” achiziționat de pe platformă itch.io. Datorita temei nivelului 2, aceea a orașului, am considerat că acest animal este un element caracteristic al acestor spații urbane. Fig. 48 reprezintă sprite-urile folosite pentru animarea inamicilor șobolani:

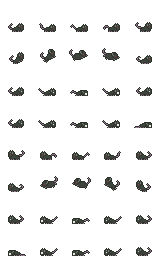
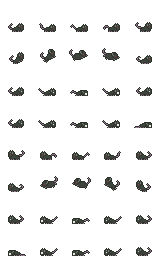


Fig. 48-Sprite-ul "Sobo"

## Implementarea steagului final

Steagul final marchează încheierea fiecărui nivel și semnalizează trecerea către următoarea scena a jocului. Grafica steagului (Fig. 49) a fost creată simplu cu ajutorul programului Paint.NET.



Fig. 49-Grafica steagului final

Structura nodului “steag\_final.tscn” include 2 componente, și anume:

* “AnimatedSprite2D” folosit pentru a gestiona animația steagului. Ea este setată să ruleze automat și în buclă, oferind o mișcare continuă care atrage atenția jucătorului către punctul de sfarșit al nivelului.
* “CollisionShape2D” care definește zona de coliziune. Forma coliziunii a fost aleasă ca fiind de tip “CapsuleShape2D” deoarece se potrivește bine cu forma steagului în mișcare, oferind astfel o detectare a coliziunii mai precisă și mai naturală.

Scriptul “steag\_final.gd” ce este atașat scenei “steag\_final.tscn” este următorul:

@export var scena\_urmatoare: String= ""

func \_on\_body\_entered(\_body):

Tranzitie.schimba\_scena\_basic(scena\_urmatoare)

În acest cod, variabila “scena\_urmatoare” este exportată pentru a putea configura ulterior, pentru fiecare nivel, scena ce urmează a fi încărcată după ce jucătorul atinge steagul. Funcția “\_on\_body\_entered(\_body)” este declanșată atunci când corpul jucătorului intră în zona de coliziune a steagului și ea apelează metoda pentru schimbarea scenei curente cu cea specificată în variabila “scena\_urmatoare”.

## Grafica nivelelor

Grafica reprezintă o componentă fundamentală pentru a crea o experiență de joc captivantă și pentru a imersa jucătorul în lumea poveștii. Fiecare nivel din jocul “În căutarea lui Pikachu” este construit cu grijă și atenție la detalii, elementele vizuale fiind distincte, iar resursele grafice adaptate tematicii. De la peisaje urbane la decoruri rurale, fiecare scenă ajută la crearea unei povești fluide. Acest subcapitol explorează detaliile grafice ale fiecărui nivel, evidențiind resursele folosite și deciziile grafice ale fiecărui nivel.

### Nivelul 1

În nivelul 1, acțiunea se petrece într-un peisaj rural și forestier, unde predomină elementele de vegetație. Grafica a fost realizată cu ajutorul mai multor pachete, principalele fiind:

* “[FREE SUMMER PIXEL ART BACKGROUNDS](https://craftpix.net/freebies/free-summer-pixel-art-backgrounds/)”- pentru fundal
* “[FREE NATURE BACKGROUNDS PIXEL ART](https://craftpix.net/freebies/free-nature-backgrounds-pixel-art/)”- pentru fundal
* “ [Unique Free Tileset](https://marvyra.itch.io/unique-free-tileset-16x16)”- pentru pământ și elemente înconjurătoare

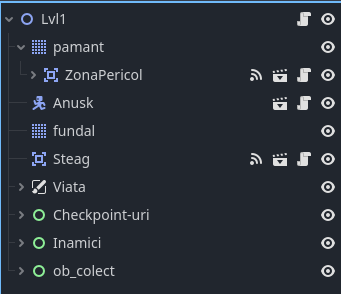


Fig. 50-Structura nivelului 1

Fig. 50 prezintă structura ierarhică a nivelului 1 din jocul “În căutarea lui Pikachu”. Elementele principale care contribuie la aceasta experiență de joc sunt:

* “Tileset” pentru pământ: acesta este utilizat pentru a defini suprafață pe care personajul va putea merge și interacționa. Tilesetul conține 5 straturi: “pamant”, “apa”, “elemente”, “iarba\_flori”.
* “Tileset” pentru fundal: este compus din diverse elemente grafice, grupate pe straturi, pentru a oferi un peisaj bogat și detaliat. Aceste elemente includ: “cer”, “munti”, “copaci”, “munte\_platformă”.
* Personajul “Anusk”: este protagonista jocului prin care utilizatorii se vor putea conecta la întreaga aventură.
* Steagul final: indică punctul de finalizare a nivelului.
* Checkpoint-uri: aici sunt toate punctele în care progresul jucătorului este salvat
* Inamici: aici se găsesc adversarii care încearcă să împiedice jucătorul. Inamicii acestui nivel sunt păianjenii.
* Containerul de viată: un element care indică starea de sanatate a personajului.

Harta finală, prezentată în Fig. 51, oferă o imagine detaliată despre traseul complet.

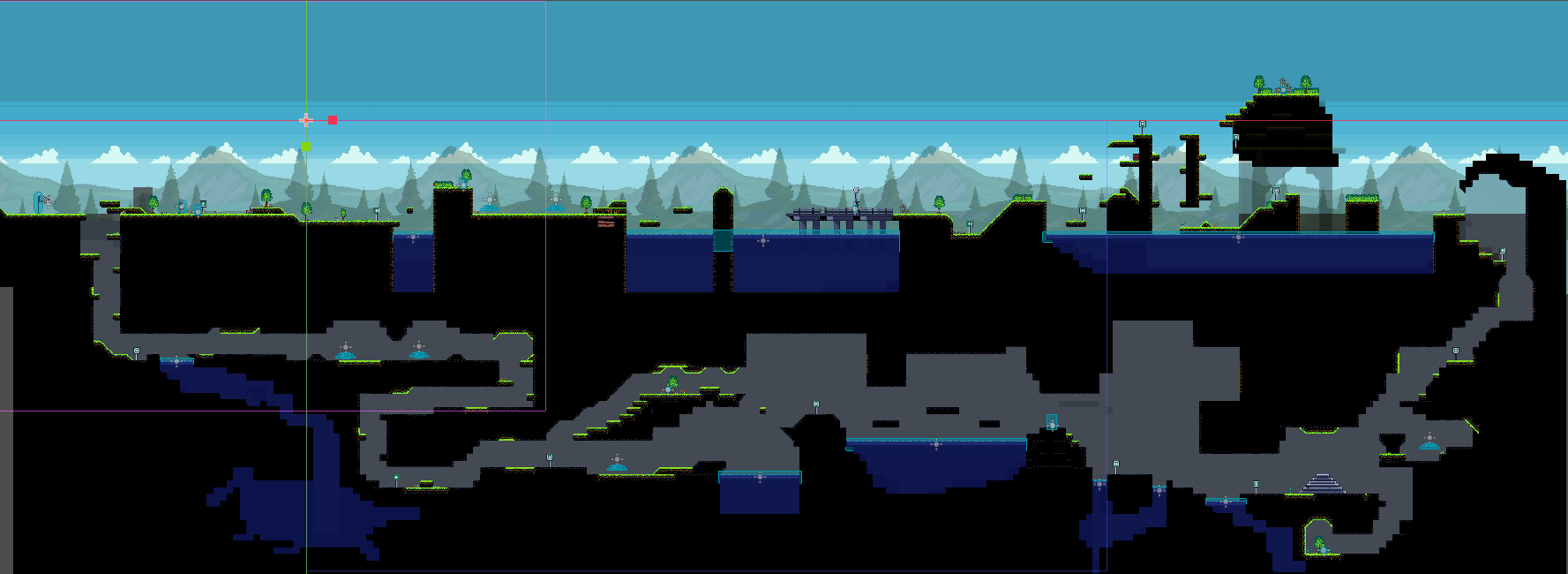


Fig. 51-Harta nivelului 1

### Nivelul 2

Nivelul 2 se desfășoară într-un oraș de munte, unde predomină structurile urbane. Grafica pentru acest nivel a fost achiziționată de pe platforma itch.io, iar numele pachetului este [“City Street: Tileset Pack”](https://muchopixels.itch.io/city-street-tileset-pack).



Fig. 52-Structura nivelului 2

Structura nivelului 2, prezentata în Fig. 52, include următoarele elemente:

* Pământ: reprezintă terenul de bază pe care se desfășoară acțiunea . Acesta este alcătuit dintr-un nod “Tileset” ce conține trei straturi: pământ, elemente, apa.
* Fundal: include mai multe straturi grafice care au ajutat la construirea peisajului urban din aproape în aproape: cer, munte, copaci\_aproape, copaci\_departe, case, decor\_ext, gard.
* Personajul Anusk: la fel ca în primul nivel, este protagonista jocului
* Steag: marcatorul final al nivelului
* ZonaPericol: include diverse zone periculoase unde personajul poate pierde din vieți dacă intră în contact cu ele
* Platforme\_miscatoare: acestea adaugă o dimensiune dinamică nivelului, obligând jucătorul să-și sincronizeze mișcările pentru a-și planifica salturile
* Tepi: sunt amplasați strategic în nivel pentru a spori dificultatea
* Viață: containerul de viață ce arată numărul de vieți rămase ale jucătorului
* Checkpoint-uri: punctele de salvare situate la intervale regulate în nivel
* Inamici: în acest nivel, jucătorul se confruntă cu șobolani, inamici adaptați pentru mediul urban.

Ca o imagine de ansamblu, harta finală a nivelului 2 este reprezentată mai jos, ilustrându-se toate elementele integrate și structura acestuia în Fig. 53.

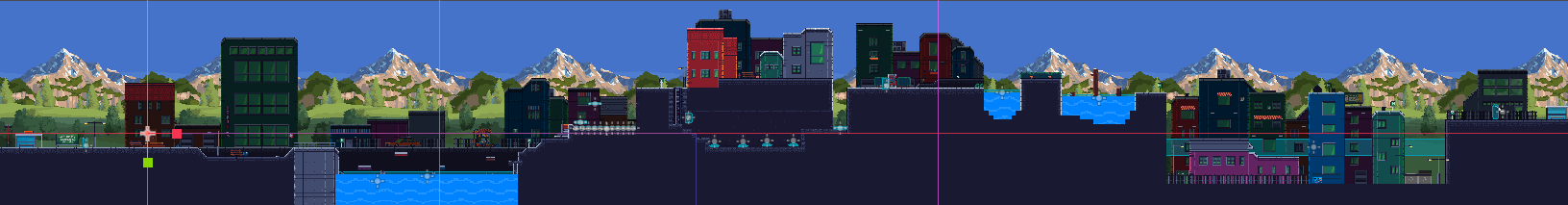


Fig. 53-Harta nivelului 2

# Ghidul utilizatorului

“În căutarea lui Pikachu” este un joc de acțiune de tip “Platformer”, unde personajul Anusk împreuna cu prietenii săi pleacă în căutarea lui Pikachu după ce acesta nu mai este de găsit. Scopul jocului este de a trece peste toate nivelele, astfel încât, la final să se reunească toată echipa.

## Lansarea jocului

Pentru a porni în aventura secolelor, descărcăm jocul “În căutarea lui Pikachu”, iar apoi se dă dublu click pe pictogramă (Fig. 54) pentru a deschide aplicația. Din meniul principal, alegem opțiunea “Play” (Fig. 55).



Fig. 54-pictograma jocului



Fig. 55-Meniu principal

## Controalele jocului

Jocul începe inițial cu niște momente cinematice însoțite de un dialog. Pentru a trece peste liniile dialogului trebuie să apesi click stânga.

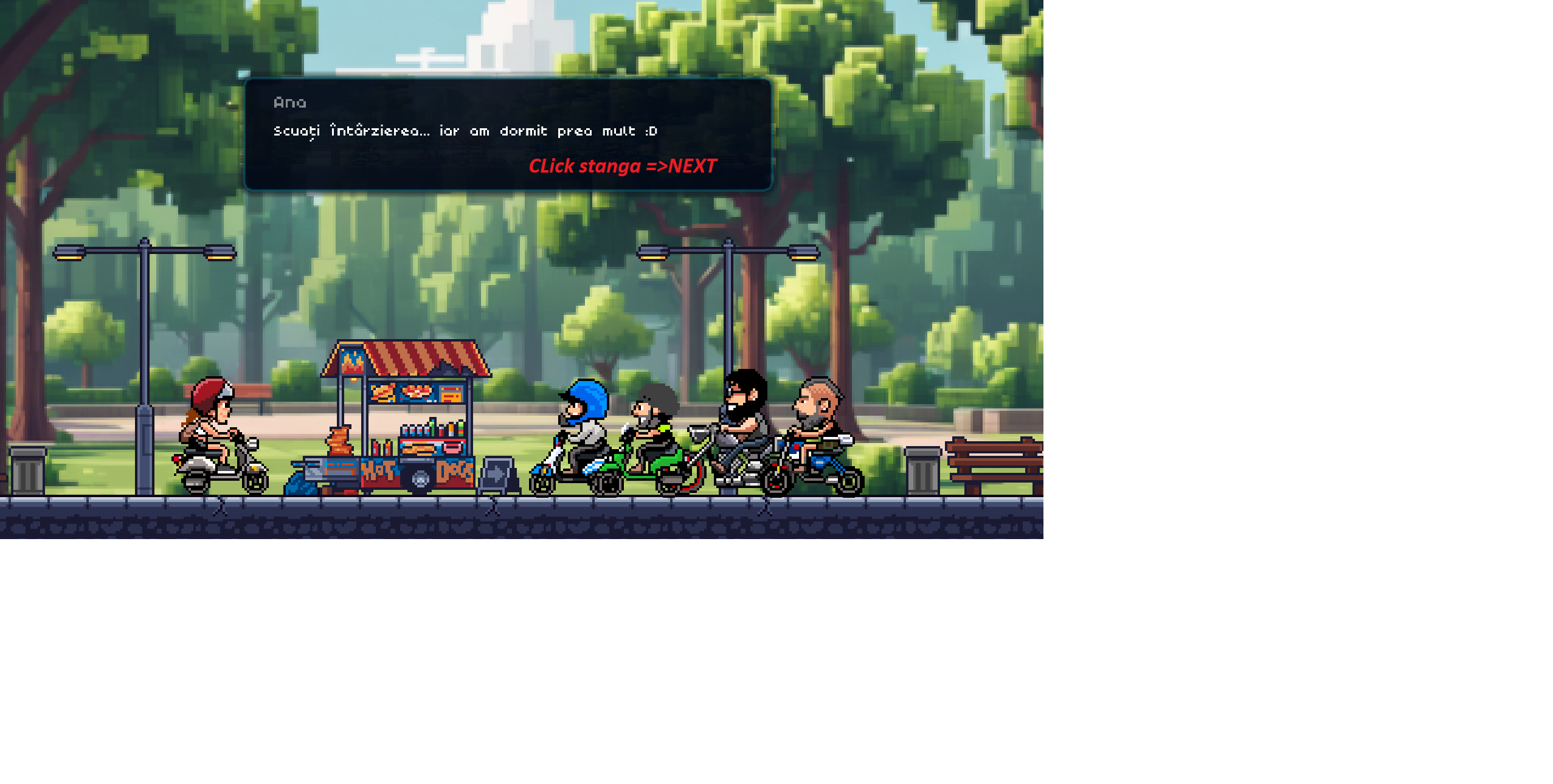


Fig. 56-Trecerea peste dialoguri

După terminarea secventelor cinematice, se pornește nivelul 1 al jocului. Aici vei putea controla personajul principal, Anusk, utilizând următoarele comenzi:

* Săgețile stânga/dreapta sau butoanele A/D pentru deplasarea în stânga și în dreapta
* Săgeată sus sau Space pentru sărituri



Fig. 57- tastele pentru control

## Activarea checkpoint-urilor

Pentru a salva progresul în joc, trebuie să te uiți după pisica Kira (Fig. 58). Ea este gri și te asteaptă să o mângâi. După ce te-ai apropiat de ea, vei observa că aceasta se ridică în picioare. În acel moment, progresul tău va fi salvat, iar în cazul în care pierzi o viată, vei reveni automat la ultima pisica atinsă. Dar ai grija, uneori, Kira se ascunde!



Fig. 58-Pisica kira

## Inamicii și pericolele

În fiecare nivel există zone periculoase sau inamici ce te vor împiedica, mai mult sau mai puțin, să progresezi. În cazul în care o să cazi de pe platforme, trebuie să sții că Anusk nu este o buna înotatoare, nu suportă înțepăturile și nici nu știe să zboare! Așadar, evită apa, țepii și platformele de pe care poți cadea pentru a nu pierde vieți.

În ceea ce privește inamicii, te vei confrunta în fiecare nivel cu măcar unul. Pentru a nu te mușca un păianjen (Fig. 59) sau un șobolan (Fig. 60), trebuie ori să îi eviți, ori să îi elimini, sărind deasupra lor . Tine minte! Sunt foarte mici și firavi, însa o singură mușcătură îți poate lua o viață.

|  |  |
| --- | --- |
|  |  |
| Fig. 59- Păianjen | Fig. 60-șobolan |

## Viața

Inițial, vei începe fiecare nivel cu un număr maxim de vieți (Fig. 61). Pe parcurs, în cazul în care nu ești atent, vei putea să le pierzi (Fig. 62). Atunci când nu vei mai avea niciuna, va trebui să reîncepi nivelul de la capăt, progresul tău pierzându-se.



Fig. 61-Viată inițială a nivelului 2



Fig. 62- Viața după ce s-au pierdut inimi

# Concluzii

Prin dezvoltarea jocului “În căutarea lui Pikachu”, am atins majoritatea obiectivelor propuse la începerea acestui proiect ambițios. Am creat un joc de acțiune ce atrage curiozitatea jucătorilor prin dorința lor de a afla ce s-a întâmplat cu Pikachu. Pentru a găsi un răspuns, ei trebuie să treacă prin toate peripețiile și provocările pe care le-am integrat în joc.

Prin implementarea mecanicilor esențiale am reușit să creez o bază solidă a proiectului, astfel că, în continuare, voi enumera realizările:

* construirea unor nivele unice: am creat două nivele, fiecare având un mediu diferit. Primul nivel se desfășoară într-o pădure, unde personajul Anusk pleacă în căutarea unor elemente ajutătoare pentru a repara motocicleta lui Petru, iar al doilea are loc într-un cadru urban, plin de platforme, obstacole și clădiri.
* crearea unei grafici încântătoare: am integrat o grafică detaliată cu un aspect atractiv, folosindu-mă de anumite pachete. Totodată am bifat și aspectul unic și personalizat al graficii jocului, explorându-mi creativitatea artistică și creând anumite elemente, precum steagul, inimile, personajul Anusk.
* macanici de joc: am implementat mecanica personajului principal pentru a permite jucătorului să îl controleze și să interacționeze cu mediul din joc. De asemenea, sistemul de salvare al progresului este un punct important pentru jocul “În căutarea lui Pikachu” deoarece el asigură o continuitate a poveștii, neexistând întreruperi în experiență.
* interfața prietenoasă: am implementat o interfață de utilizator intuitivă și ușor de navigat, cu puține elemente.
* poveste captivantă și originală: am reușit să transpun în joc povestea originală a dispariției prietenului meu Marius, poreclit și Pikachu. De la locul nostru de întâlnire obișnuit, Sala Sporturilor Olimpia, până în Poiana Brașov, acolo unde credeam că îl vom găsi, am inclus toate elementele care să reflecte aventura noastră reală; de exemplu, repararea motorului lui Petru cu un simplu șurub găsit în pădure.

În ceea ce privește dezvoltarea aplicației în viitor, aș dori să îmbunătățesc semnificativ experiența de joc. Câteva direcții pe care intenționez să le explorez în viitor sunt:

* crearea mai multor nivele: aș dori să mai adaug înca 4 nivele denumite astfel: “Baba Dochia și salvatorul” , “Circuitul Adancata”, “În căutarea raspunsurilor”, “Reuniunea salvatoare”. Fiecare va avea câte o provocare unică.
* introducerea de noi inamici: mi-am propus să creez inamici noi cu comportamente și abilități diferite. Un exemplu ar fi Baba Dochia care ne va ataca cu pietre și va fi mult mai greu de eliminat.
* adaugarea de elemente colectabile și upgrade-uri: se vor introduce bănuți cu ajutorul cărora jucătorii își pot cumpăra haine pentru personajul Anusk sau își

pot achiziționa abilități specifice, ca de exemplu trasul cu arcul sau aruncatul cu bomboane.

* lansarea jocului pe platforme: după îndeplinirea tuturor obiectivelor propuse, jocul “În căutarea lui Pikachu” va fi disponibil pe platforme precum EpicGames, Stream, Eneba, etc.
* optimizarea și îmbunătațirea performanței: pentru a crea o experiență mai plăcută jucătorului, îmi propun să optimizez codul pentru a reduce timpul de încărcare al nivelelor. De asemenea, aș dori ca jocul să fie disponibil pe mai multe configurații de hardware, iar pentru asta se vor realiza mai multe teste.

Aceste direcții de dezvoltare vor aduce jocul “În căutarea lui Pikachu” la un alt nivel, existând o mare posibilitate ca acesta să devină foarte cunoscut și descărcat de milioane de jucători.

# Bibliografie

* + - 1. Acres T., *GTA V turns 10: The impact and legacy of Rockstar's biggest game - and why sequel is taking so long*, [URL](https://news.sky.com/story/gta-v-turns-10-the-impact-and-legacy-of-rockstars-biggest-game-and-why-sequel-is-taking-so-long-12935879), data accesarii: 25 martie 2024
      2. Berthke E., *Game Development and Production*, Editura Wordware Publishing, 2003
      3. Celeste Wiki Contributors, Celeste Wiki, celest.ink, [URL](https://celeste.ink/wiki/Celeste_Wiki), data acesarii: 20 aprilie 2024
      4. Chris Bradfield, *Godot Engine Game Development Projects: Build five cross-platform 2D and 3D with Godot 3.0*, Editura Packt, 2018
      5. CounterStrike Player ,*Reset your score! /rs command,* [URL](https://gamebanana.com/mods/39607), data accesarii: 25 martie 2024
      6. Cullion C., *THE COMPLETE HISTORY OF STREET FIGHTER* ,[URL](https://tiredoldhack.com/2018/05/26/the-complete-history-of-street-fighter/), data accesarii: 27 martie 2024
      7. Doran J. P., Casanova M., Game Development Patterns and Best Practices, Editura Packt, 2017
      8. Godot Contributors, Godot 4.0 sets sail: All aboard for new horizons, Godot, [URL](https://godotengine.org/article/godot-4-0-sets-sail/#gdscript), data accesarii: 15 aprilie 2024
      9. Godot Engine Documentation Writers, Godot 4.1 Documentation, [URL](https://docs.godotengine.org/en/4.1/getting_started/introduction/introduction_to_godot.html), data accesarii: 15 februarie 2024
      10. Heckaday, *History of Video Games: Four Decades of Video Entertainment*, [URL](https://cdn.hackaday.io/files/1649347056536256/History%20of%20Video%20Games-Four%20Decades%20of%20Video%20Entertainment.pdf), data accesarii: 2 aprilie 2024
      11. Kent S. L., The Ultimate History of Video Games,  *From Pong to Pokemon--The Story Behind the Craze That Touched Our Lives and Changed the World,* vol.1,Editura Paperback, 2001
      12. Landesmuseum Z., *The History of Video Games*, [URL](https://www.landesmuseum.ch/landesmuseum/ausstellungen/wechselausstellungen/2020/games/medien/the-history-of-video-games-en.pdf), data accesarii: 25 martie 2024
      13. Melville R., *How Atari's Pong started the video game revolution 40 years ago,* [URL](How%20Atari's%20Pong%20started%20the%20video%20game%20revolution%2040%20years%20ago), data accesarii: 25 martie 2024
      14. Mukherjee S., *Video Games and Storytelling: Reading Games and Playing Books*, Editura Palgrave Macmillan, 2015
      15. Nguyen S., *Making a 2D Game with Godot 4 Engine*, [URL](https://www.theseus.fi/bitstream/handle/10024/862142/Son_Nguyen.pdf?sequence=4), data accesarii: 26 aprilie 2024
      16. Peacpx, Read Dead Redemption Wallpaper, [URL](https://www.peakpx.com/en/hd-wallpaper-desktop-oidlx), data accesarii: 24 martie 2024
      17. Raessens J., Goldstein J., *Handbook of Computer Game Studies*, Editura MIT Press, 2011
      18. Rome B., Hussey C., *Games' Most Wanted: The Top 10 Book of Players, Pawns, and Power-Ups*, Editura Potomac Books, 2013
      19. Salen K, Zimmerman E., *The Game Design Reader: A Rules of Play Anthology*, Editura MIT Press Ltd, 2005
      20. Spanos A., *Game of History: Games and Gaming as Historical Source*, Editura Routledge, 2021
      21. Sturgeon A., Kumar S., *Getting Started with Paint.Net*, Editura Packt, 2013
      22. Super Mario Wiki Contributors, *Super Mario Bros*, Fandom , [URL](https://mario.fandom.com/wiki/Super_Mario_Bros.#Gameplay), data accesarii: 20 aprilie 2024
      23. Thompson B., Helfand J., *The art of Graphic Design*, Editura Westvaco, 2018
      24. Wikipedia Contributors, *Space Invaders,* [URL](https://en.wikipedia.org/wiki/Space_Invaders), data accesarii: 2 aprilie 2024
      25. Wikipedia, *Adobe Illustrator*, Wikiwand, [URL](https://www.wikiwand.com/en/Adobe_Illustrator_Artwork), data accesarii: 11 martie 2024
      26. Wilde T., *Ori and the Blind Forest Review*, PcGamer, [URL](https://www.pcgamer.com/ori-and-the-blind-forest-review/), data accesarii: 20 aprilie 2024

**Anexe**

Scriptul Anusk.gd:

class\_name Jucator

extends CharacterBody2D

signal viata\_schimbata

@export var viteza : float = 200.0

@export var salt: float= -300.0

@export var salt\_dublu : float= -300

var gravity = ProjectSettings.get\_setting("physics/2d/default\_gravity")

var a\_sarit\_dublu: bool=false

var sari\_dublu: bool= false

var blocheaza\_anim: bool= false

var directie: Vector2= Vector2.ZERO

var putem\_controla: bool=true

var in\_aer=false

@export var max\_inimi: int =3

@onready var inimi: int= max\_inimi

@onready var anusk : AnimatedSprite2D =$AnimatedSprite2D

func \_ready():

ManagerJoc.jucator = self

#print(self)

func \_physics\_process(delta):

if not putem\_controla: return

# Add the gravity.

if not is\_on\_floor():

velocity.y += gravity \* delta

in\_aer=true

else:

a\_sarit\_dublu=false

if in\_aer==true:

anusk.play("stai")

blocheaza\_anim=false

in\_aer=false

# Handle jump.

if Input.is\_action\_just\_pressed("sus"):

if is\_on\_floor():

sari()

elif not a\_sarit\_dublu:

velocity.y=salt\_dublu

a\_sarit\_dublu=true

# Get the input directie and handle the movement/deceleration.

directie = Input.get\_vector("stanga", "dreapta","sus","jos")

if directie:

velocity.x = directie.x \* viteza

else:

velocity.x = move\_toward(velocity.x, 0, viteza)

move\_and\_slide()

actualizeaza\_animatia()

directie\_mers()

func directie\_mers():

if directie.x>0:

anusk.flip\_h=false

elif directie.x<0:

anusk.flip\_h=true

func actualizeaza\_animatia():

if not blocheaza\_anim:

if directie.x!=0:

anusk.play("mergi")

else:

anusk.play("stai")

func sari():

velocity.y= salt

anusk.play("sari")

blocheaza\_anim=true

#functie pentru pericol

func pericol():

inimi -= 1

if inimi<=0:

inimi=max\_inimi

Tranzitie.daca\_moare("res://Scene/levels/lvl1.tscn")

else:

viata\_schimbata.emit(inimi) #semnal pentru ca a pierdut o inima

putem\_controla=false

await get\_tree().create\_timer(0.5).timeout

reseteaza\_poz()

#print("peric")

func reseteaza\_poz():

ManagerJoc.respawneaza()

putem\_controla=true

print("reset")

if Input.is\_action\_just\_pressed("sus"):

if is\_on\_floor():

sari()

elif not a\_sarit\_dublu:

velocity.y=salt\_dublu

a\_sarit\_dublu=true

# Get the input directie and handle the movement/deceleration.

directie = Input.get\_vector("stanga", "dreapta","sus","jos")

if directie:

velocity.x = directie.x \* viteza

else:

velocity.x = move\_toward(velocity.x, 0, viteza)

move\_and\_slide()

actualizeaza\_animatia()

directie\_mers()

func directie\_mers():

if directie.x>0:

anusk.flip\_h=false

elif directie.x<0:

anusk.flip\_h=true

func actualizeaza\_animatia():

if not blocheaza\_anim:

if directie.x!=0:

anusk.play("mergi")

else:

anusk.play("stai")

func sari():

velocity.y= salt

anusk.play("sari")

blocheaza\_anim=true

#functie pentru pericol

func pericol():

inimi -= 1

if inimi<=0:

inimi=max\_inimi

Tranzitie.daca\_moare("res://Scene/levels/lvl1.tscn")

else:

viata\_schimbata.emit(inimi) #semnal pentru ca a pierdut o inima

putem\_controla=false

await get\_tree().create\_timer(0.5).timeout

reseteaza\_poz()

#print("peric")

func reseteaza\_poz():

ManagerJoc.respawneaza()

putem\_controla=true

print("reset")

#functie pentru pericol

func pericol():

inimi -= 1

if inimi<=0:

inimi=max\_inimi

Tranzitie.daca\_moare("res://Scene/levels/lvl1.tscn")

else:

viata\_schimbata.emit(inimi) #semnal pentru ca a pierdut o inima

putem\_controla=false

await get\_tree().create\_timer(0.5).timeout

reseteaza\_poz()

func reseteaza\_poz():

ManagerJoc.respawneaza()

putem\_controla=true

print("reset")

Scriptul checkpoint.gd:

extends Area2D

class\_name Checkpoint

var check\_activat= false

@onready var pisica= $AnimatedSprite2D

func activeaza\_check():

ManagerJoc.checkpoint\_curent= self

pisica.play('activ')

check\_activat= true

print("activat")

func \_on\_body\_entered(body):

if body is Jucator && !check\_activat:

activeaza\_check()

Scriptu ManagerJoc.gd:

extends Node

var checkpoint\_curent: Checkpoint

var jucator: Jucator

func respawneaza():

if checkpoint\_curent != null:

jucator.position=checkpoint\_curent.global\_position

print("pozitie init")

Scriptul comportament\_inamic.gd:

extends CharacterBody2D

class\_name Inamic

var jucator:Jucator

@export var viteza\_inamic:float

@onready var animatie=$AnimatedSprite2D

@onready var coleziune\_detectJuc=$DetectJucator/CollisionShape2D

@onready var coleziune\_atac=$Zona\_omorare\_jucator/CollisionShape2D

@onready var coleziune\_inamic=$CollisionShape2D

var gravity = ProjectSettings.get\_setting("physics/2d/default\_gravity")

var este\_mort:bool =false

var ataca\_jucator:bool=false

# Called every frame. 'delta' is the elapsed time since the previous frame.

func \_process(delta):

if este\_mort:

await get\_tree().create\_timer(1.6).timeout

queue\_free() #elimina inamicul din scena

else:

if ataca\_jucator:

await get\_tree().create\_timer(0.8).timeout

ataca\_jucator=false #ca sa poata sa atace de mai multe ori, nu doar o data

if not is\_on\_floor():

velocity.y += gravity \* delta

velocity.x= viteza\_inamic\*delta

if viteza\_inamic>0:

animatie.play("mers\_dreapta")

elif viteza\_inamic<0:

animatie.play("mers\_stanga")

move\_and\_slide()

func moare():

velocity.x=0

coleziune\_inamic.queue\_free()

animatie.play("moare")

este\_mort=true

coleziune\_detectJuc.queue\_free() #stergem coleziunea ca sa nu mai poata interactiona cu ea

coleziune\_atac.queue\_free()

func ataca():

ataca\_jucator=true

velocity.x=0

#cum ataca(stanga sau dreapta)

if jucator.position.x<position.x:

animatie.play("atac\_stanga")

else:

animatie.play("atac\_dreapta")

func \_on\_zona\_omorare\_jucator\_body\_entered(body):

if body is Jucator:

jucator=body

jucator.putem\_controla=false #nu il mai putem controla daca este atacat

ataca()

await get\_tree().create\_timer(0.5).timeout

jucator.putem\_controla=true

body.pericol() #respauneaza jucatorul si sterge inima

func \_on\_detectare\_zid\_body\_entered(\_body):

viteza\_inamic=-viteza\_inamic # merge in sens opus

func \_on\_detect\_jucator\_body\_entered(body):

if body is Jucator:

#print("detectat")

moare()

if ataca\_jucator:

await get\_tree().create\_timer(0.8).timeout

ataca\_jucator=false

if not is\_on\_floor():

velocity.y += gravity \* delta

velocity.x= viteza\_inamic\*delta

if viteza\_inamic>0:

animatie.play("mers\_dreapta")

elif viteza\_inamic<0:

animatie.play("mers\_stanga")

move\_and\_slide()

func moare():

velocity.x=0

coleziune\_inamic.queue\_free()

animatie.play("moare")

este\_mort=true

coleziune\_detectJuc.queue\_free() #stergem coleziunea ca sa nu mai poata interactiona cu ea

coleziune\_atac.queue\_free()

func ataca():

ataca\_jucator=true

velocity.x=0

#cum ataca(stanga sau dreapta)

if jucator.position.x<position.x:

animatie.play("atac\_stanga")

else:

animatie.play("atac\_dreapta")

func \_on\_zona\_omorare\_jucator\_body\_entered(body):

if body is Jucator:

jucator=body

jucator.putem\_controla=false

ataca()

await get\_tree().create\_timer(0.5).timeout

jucator.putem\_controla=true

body.pericol() #respauneaza jucatorul si sterge inima

func \_on\_detectare\_zid\_body\_entered(\_body):

viteza\_inamic=-viteza\_inamic # merge in sens opus

func \_on\_detect\_jucator\_body\_entered(body):

if body is Jucator:

#print("detectat")

moare()

await get\_tree().create\_timer(0.5).timeout

jucator.putem\_controla=true

body.pericol() #respauneaza jucatorul si sterge inima

func \_on\_detectare\_zid\_body\_entered(\_body):

viteza\_inamic=-viteza\_inamic # merge in sens opus

func \_on\_detect\_jucator\_body\_entered(body):

if body is Jucator:

moare()

Scriptul steag\_final.gd:

extends Area2D

@export var scena\_urmatoare: String= ""

func \_on\_body\_entered(\_body):

Tranzitie.schimba\_scena\_basic(scena\_urmatoare)

Scriptul zona\_pericol.gd:

extends Area2D

func \_on\_body\_entered(body):

if body is Jucator:

body.pericol()

Scriptul tranzitie.gd:

extends CanvasLayer

@onready var animatie= $AnimationPlayer

func schimba\_scena\_basic( nume\_scena:String):

animatie.play('tranzitie')

await animatie.animation\_finished

get\_tree().change\_scene\_to\_file(nume\_scena)

animatie.play\_backwards('tranzitie')

func schimba\_scena1( nume\_scena:String):

animatie.play('incepe\_jocul')

await animatie.animation\_finished

get\_tree().change\_scene\_to\_file(nume\_scena)

animatie.play\_backwards('tranzitie')

func daca\_moare(nume\_scena: String):

animatie.play('tranzitie\_moarte')

await animatie.animation\_finished

get\_tree().change\_scene\_to\_file(nume\_scena)

animatie.play\_backwards('tranzitie')

func schimba\_scena1( nume\_scena:String):

animatie.play('incepe\_jocul')

await animatie.animation\_finished

get\_tree().change\_scene\_to\_file(nume\_scena)

animatie.play\_backwards('tranzitie')

func daca\_moare(nume\_scena: String):

animatie.play('tranzitie\_moarte')

await animatie.animation\_finished

get\_tree().change\_scene\_to\_file(nume\_scena)

animatie.play\_backwards('tranzitie')

Scriptul lvl1.gd:

extends Node2D

@onready var container\_inimi= $Viata/ContainerInimi

@onready var jucator= $Anusk

func \_ready():

container\_inimi.seteaza\_max\_inimi(jucator.max\_inimi)

container\_inimi.updateInimi(jucator.inimi)

jucator.viata\_schimbata.connect(container\_inimi.updateInimi)

Scriptul bogdan\_pers.gd:

extends Area2D

@onready var anim= $Bogdan

func \_ready()-> void:

anim.play("stai") # Replace with function body.

func play\_anim(nume\_animatie:String) -> void:

anim.play(nume\_animatie)

Scriptul main.gd:

extends Node2D

func \_on\_button\_pressed() -> void:

Tranzitie.schimba\_scena1("res://Scene/CutScenes/intro.tscn")

#Tranzitie.schimba\_scena\_basic("res://Scene/levels/lvl1.tscn")

func \_on\_button\_2\_pressed():

get\_tree().quit()

Scriptul intro.gd:

extends Node2D

func \_ready():

$AnimationPlayer.play("scena1")

await get\_tree().create\_timer(1.2).timeout

DialogueManager.show\_example\_dialogue\_balloon(load("res://Materiale/dialog/dialog\_intro.dialogue"), "inceput")

func dialog\_term():

$AnimationPlayer.play("Plecare")

await get\_tree().create\_timer(3.1).timeout

DialogueManager.show\_example\_dialogue\_balloon(load("res://Materiale/dialog/dialog\_intro.dialogue"), "Cosmin")

await get\_tree().create\_timer(3.4).timeout

Tranzitie.schimba\_scena\_basic("res://Scene/CutScenes/lvl1\_cutscene.tscn")

Scriptul lvl1\_cutscene.gd:

extends Node2D

func \_ready():

$AnimationPlayer.play("animatie1")

await get\_tree().create\_timer(5).timeout

DialogueManager.show\_example\_dialogue\_balloon(load("res://Materiale/dialog/dialog\_intro.dialogue"), "lvl1\_cutscene")

func lvl1():

Tranzitie.schimba\_scena\_basic("res://Scene/levels/lvl1.tscn")