Cuprins

[Introducere 2](#_Toc18977260)

[De ce am ales Unity? 4](#_Toc18977261)

[Descrierea aplicației? 4](#_Toc18977262)

[CAPITOLUL 1 5](#_Toc18977263)

[Chitara ca instrument 5](#_Toc18977264)

[Tehnici pentru chitară 6](#_Toc18977265)

[Partituri de chitară 7](#_Toc18977266)

[CAPITOLUL 2 8](#_Toc18977267)

[Unity și lucrul în inspector 8](#_Toc18977268)

[1. Interfața 8](#_Toc18977269)

[CAPITOLUL 3 12](#_Toc18977270)

[Scripturi folosite 12](#_Toc18977271)

[1. Constante și locații 12](#_Toc18977272)

[2. Librăria NAudio 13](#_Toc18977273)

[3. InputManager 14](#_Toc18977274)

[4. Gameplay 17](#_Toc18977275)

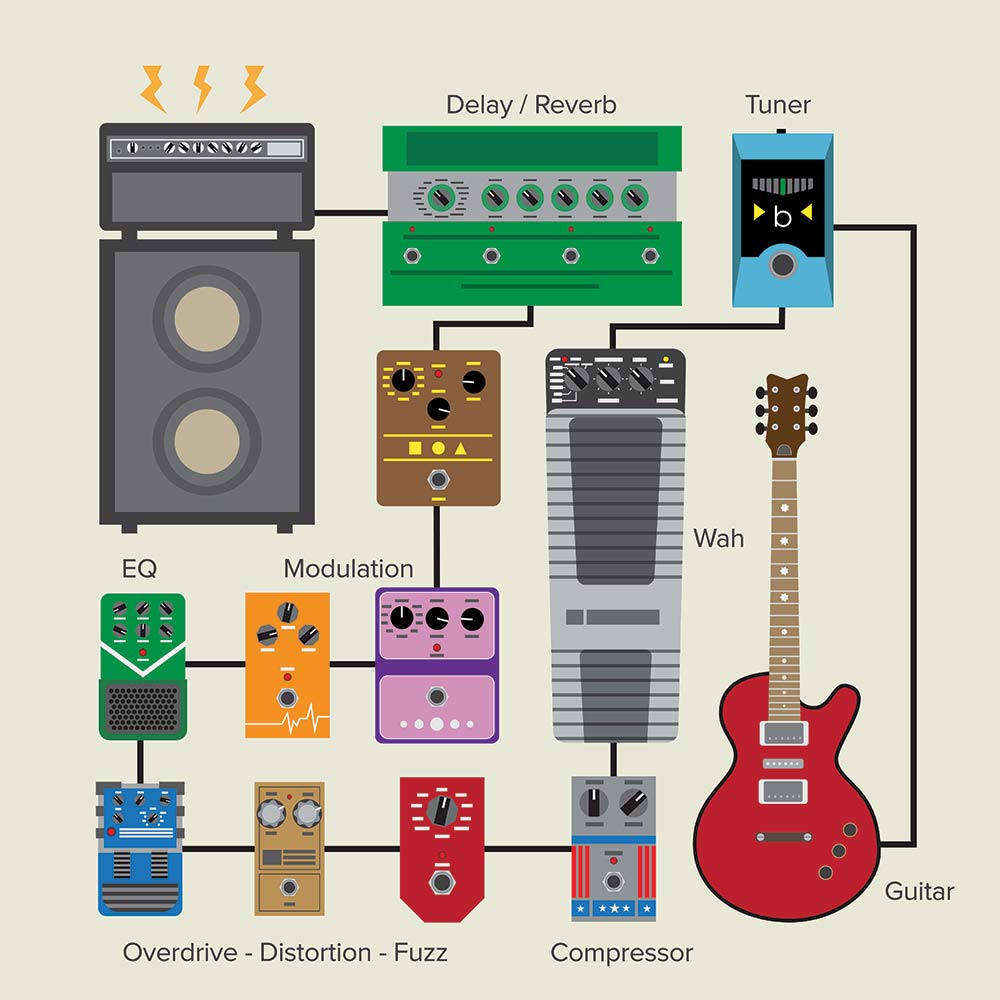
[Lista Figurilor 20](#_Toc18977276)

[Bibliografie 22](#_Toc18977277)

[*www.ub.ro; e-mail: stiinte@ub.ro* 23](#_Toc18977278)

# Introducere

Chitara electrică este un instrument deosebit, apărut în 1931 și a fost folosită mai întâi in trupe de Jazz, unde cântăreți cântau note individuale în compoziții de solo. O chitară electrică spre deosebire de una acustică sau clasică înregistrează vibrația chorzilor prin doze și o transformă în suntet digital (funcționează asemănator cu un microfon clasic). Acest sunet poate fi modificat prin diverse pedale care adaugă distort, delay, chorus, etc... Pentru ca sunetul să fie apoi auzit de la o boxă, trebuie trecut printr-un amplificator.



Imaginea – Setup chitară electrică

Pentru că acest set-up poate ocupa foarte mult spațiu și nu este portabil, numeroase alternative pentru compunerea pieselor sau pentru a practica instrumentul au fost dezvoltate.

-Ampkit este o aplicație IOS care lasă utilizatorul sa conecteze o chitara adevarata la dispozitivul lui IOS prin intermediul unui adaptor. Prin acest lucru utilizatorul poate practica oriunde, dar tot are nevoie de instrument.

-Rocksmith 2014 este simulatorul pentru PC si Mac care permite conectarea unei chitări adevărate la calculator și vizualizarea în timp real a notelor unui cântec pe pozițiile lor pe chitară. Prin acest mijloc jucătorul poate învăța în timp real o melodie și de a vedea greșelile făcute in timpul cântări acestea. Partea negativă este că necesită o chitară reală, un sistem care îl poate rula și un cablu adaptor, ceea ce nu il face portabil.



Imaginea – Rocksmith 2014

-GarageBand este o aplicație pentru dispozitivele IOS mobile în care utilizatorului i se oferă un instrument simulat de la chitară electrică până la fluier. Utilizatorul poate cânta la chitară folosind tehnicile specifice instrumentului, poate înregistra în timp real ce cântă și poate folosi efecte și alte instrumente pe fundal în timp ce cântă. Spre deosebire de cele două aplicații mai sus menționate, utilizatorul nu mai are nevoie de un instrument real sau de un adaptor, dar nu poate vedea portative în timp real și de a urmări daca nota cântată a fost greșită.



Imaginea – Garage Band IOS

Din aceste motive am dorit să combin partea de citire a portativelor din Rocksmith cu simularea chitării în timp real pe majoritatea dispozitivele mobile.

## De ce am ales Unity?

Am ales să folosesc Unity pentru abilitatea lui de a dezvolta și construi aplicați pe diverse platforme. Pe lângă asta, codul în Unity este scris în C# și are o mulțime de librării externe și stabilitate de care poți beneficia.

## Descrierea aplicației?

GuitarSimChart este o aplicație pe platformele IOS și Android dezvoltată în Unity care simulează o chitară adevărată și modul în care se cântă în mod normal la ea. Folosind touch-screen-ul dispozitivului utilizatorul poate cânta mai multe note în același timp și poate folosi glisarea ecranului pentru simularea îndoiri corzilor.

Are două moduri, modul Free Guitar, in care poți cânta liber, ca la o chitară reală și modul de Real Play, unde există Practice Mode, unde nu există scor iar cântecul nu poate fi întrerupt și modul Arcade unde jucătorul încearcă să obțină un scor cât mai mare.

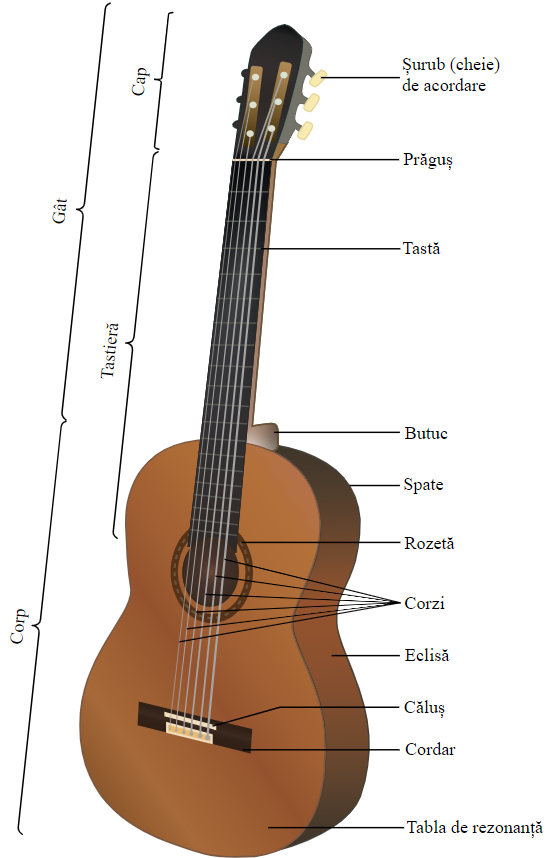


Imaginea – Main menu GuitarSimChar

Portativele sunt luate din fisiere midi convertite din portative de chitară adevărate folosind Guitar Pro 7, deci orice cântec care are un portativ compatibil cu Guitar Pro, poate fi incărcat în simulator pentru a putea fi cântat. Sau dacă utilizatorul stie să le creeze, poate face propriile portative, și să le încarce în aplicație.

# CAPITOLUL 1

## Chitara ca instrument



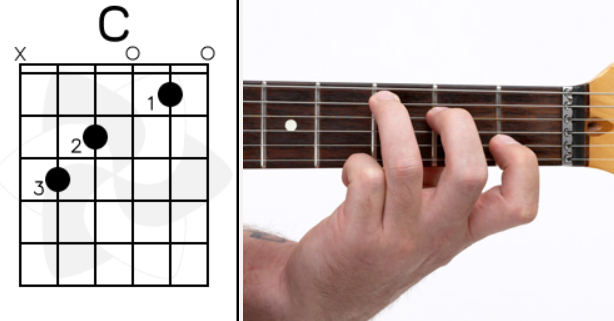
Imaginea – Elemente ale chitării

Chitara este un instrument muzical cu coarde ciupite. Ea are un gât lung, delimitat cu ajutorul tastelor (en. frets), şi o cutie de rezonanţă ale cărei ambe feţe sunt plane. În lateral, se formează „umerii” caracteristici, rotunzi, care dau corpului forma cifrei 8. Acordajul standard în cvarte perfecte (cu o singură excepţie) aduce cu cel al instrumentelor din familia violei (viola da gamba, violone ş.a.).

Pentru a produce o notă la chitară trebuie să ții apăsat cu mâna stângă pe coarda pe care urmează să o ciupești cu mâna dreaptă în intervalul tastei dorite.

Notele muzicale pot fi notate în stil European ca: “Do, Re, Mi, Fa, Sol, La, Si”, și still American ca: “C, D, E, F, G, A, B”. Doar numele acestor note diferă, Aplicația folosește numele în engleză.

Mai multe note cântate în aceiași gamă, în același timp, creează un acord. Aceste note sună bine împreună și sunt numite dupa gama folosită, ex. acordul C are notele (X 3 2 0 1 0). X înseamnă că prima coardă va fi muțită, 0 înseamnă lipsa unui deget pe coarda respectivă( coarda este liberă), iar numele înseamnă poziția degetelor pe tastatura chitării.



Imaginea – Acordul C

## Tehnici pentru chitară

Cântăreți la chitară au dezvoltat de-a lungul timpului mai multe tehnici:

-Indoire (bend) este atunci când o notă este susținută iar decetul care o ține urcă sau coboară în aceiași tastă. Acest lucru provoacă nota să crească în frecvență (pitch) până când trece la următoarele note.



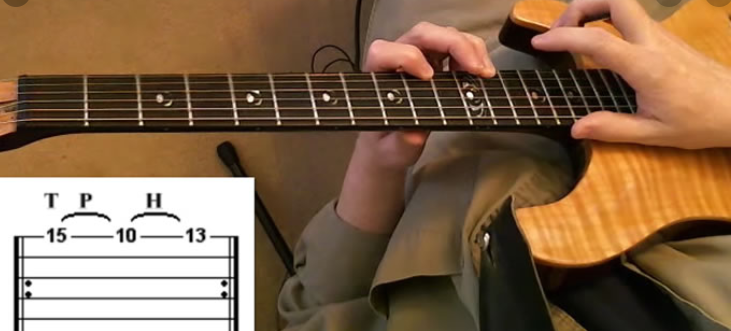
Imaginea - Tehnica de Bend

-Vibrato este asemănătoare unui bend dar coarda nu este urcată doar la o anumită notă ci este miscată rapid între note.

-Slide, folosit de obicei cu o sticlă sau un inel de metal pe un deget de la mâna sângă, este atunci când urci sau cobori dealungul gâtului chitării ținând apăsate corzile pe care le ciupești. Sunetul produs poate suna ca un bend, dar poate continua pentru câte taste parcurgi în alunecare.

-Hammer On este apăsarea unei taste cu putere cu un deget, pe o coardă care sună sau nu. Acest lucru face nota să sune fără a fi nevoie de a ciupi coarda. Pull Off este opusul, când o coardă apăsată este lăsată liberă brusc, ciupind-o.

-Tapping este combinația de Hammer On și Pull Off făcută cu 2 maini.



Imaginea - Tehnica de Tapping

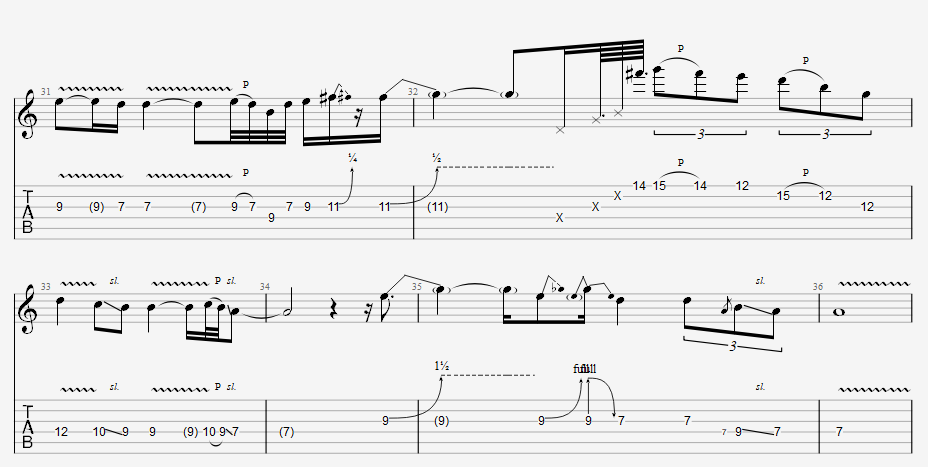
Aceste tehnici pot fi folosite în aplicație și sunt notate ca atare în modul Arcade.

## Partituri de chitară

Spre deosebire de o partitură de muzică generală, o partitură de chitară este compusă din două seturi de 5 respectiv 6 linii perpendiculare.

Setul de sus notat la început cu Cheia Sol este ca un portativ obisnuit, și arată timpul pentru fiecare notă, modul de intonație, și pauzele din timpul melodiei.

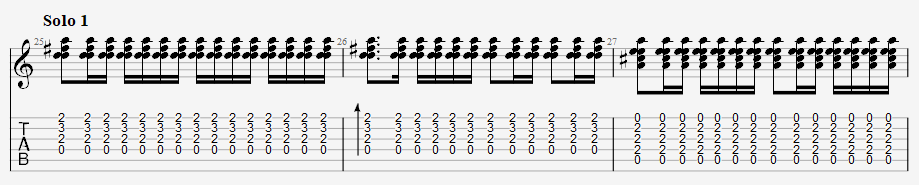
Setul de jos notat la început cu TAB (tabulatură) arată locul pe tastatură unde trebuie ținut degetul pentru a obține nota de mai sus.



Imaginea - Partitură lead

Pentru chitară există două tipuri de portative. Pentru cea de lead care se axează pe note cântate singure și partea de Solo de chitară. Pentru partea de ritm se axează pe foarte multe acorduri și batăi mai complexe.

Acele săgeți din fața unui acord arată modul de bătaie, care poate schimba cum sună acordul. Ele indică și cel mai optim mod pentru un anumit ritm.



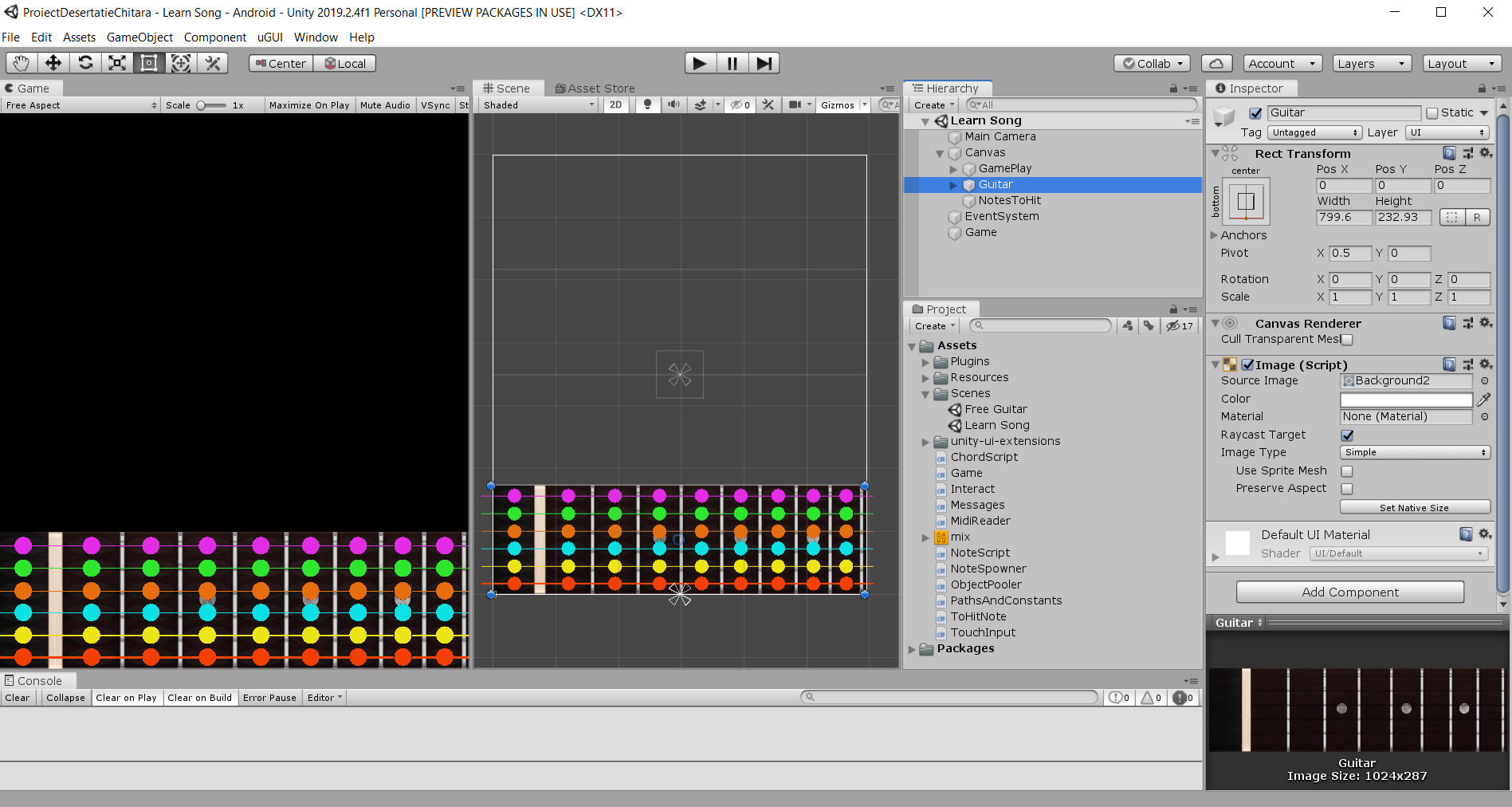
Imaginea - Partitură ritm

# CAPITOLUL 2

## Unity și lucrul în inspector

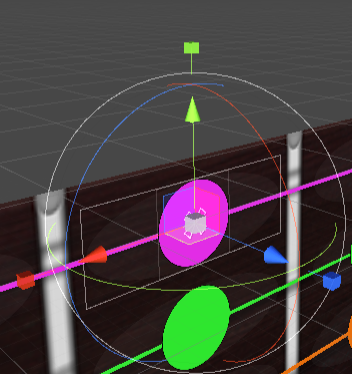
### Interfața

Pentru Unity aceasta poate fi modificată cum dorește utilizatorul, eu am ales setup-ul acesta deoarece oferă tot ce este nevoie de vazut, pe un singur ecran de la stânga la dreapta, iar fereastra de consolă este jos, și suficient de mare pentru a putea citi totul.



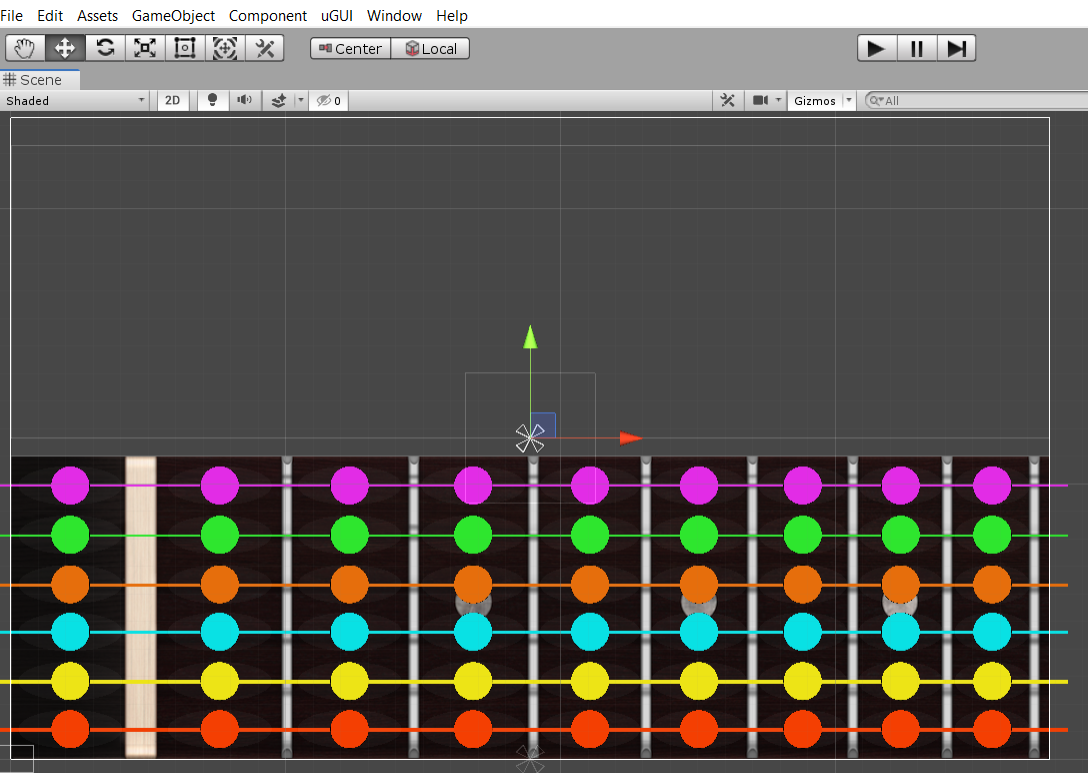
Imaginea - interfața generală

În fereastra de scenă se pot obesra uneltele de manipulare a obiectelor în stânga sus care pot fi accesate cu scurtăturile:

* Q – pan, miscă spațiul de lucru;
* W – translate, miscă obiectul în scenă pe cele trei axe X,Y,Z;
* E – rotate, roteste obiectul în scenă pe cele trei axe X,Y,Z;
* R – scale, redimensionează obiectul în scenă pe cele trei axe X,Y,Z;

Imaginea - multiTool

* T – scaleRect, folosit în special pentru sprit-uri, scalează într-un patrat(doar în 2 axe opuse de la cameră);
* Y – multiTool, este o combinație între translate, rotate și scale;



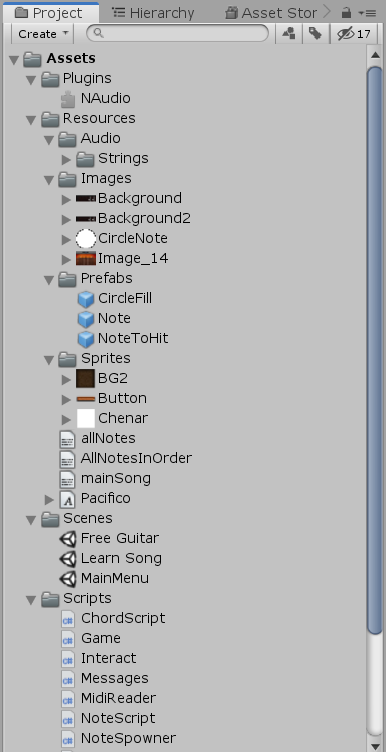
Imaginea - Fereastra de scenă

 În fereastra de Hierarchy se poate vedea și modifica organizarea obiectelor din scenă. Chitara este formată dintr-o imagine de fundal a tastaturi, deasupra ei având notele și deasupra lor liniile randate prin UILineRenderer folosind un script din Unity UI Toolkit plug-in.

NoteSpowner este un obiect gol pus deasupra chitări care contine punctele de unde se vor instanția notele.

NotesToHit este un obiect gol care va fi părinte tuturor obiectelor note care vor fi instanțiate.

Game este un obiect cu un script Game.cs cu clasa singleton Game care conține referințe la toate scripturile mai mari pentru usurința în găsire mai târziu.

 în fereastra de Project avem ierarhia subfolderelor din Assets. Aici se salvează toate resursele de care avem nevoie la Build.

Folderul de Plugins contine de obicei fisiere tip .dll de care sunt dependente anumite plug-in-uri cum ar fi NAudio.

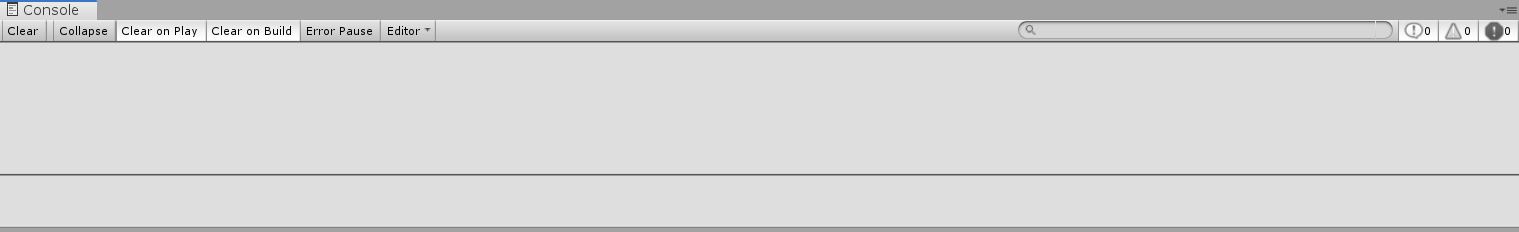
Folderul de Resources este foarte important, deoarece din el se vor lua resurse după rularea aplicației. În scriptul PathsAndConstants.cs salvăm constante cu numele subfolderelor acestuia pentru usurință a schimbării și posibilitatea mică de a greși.

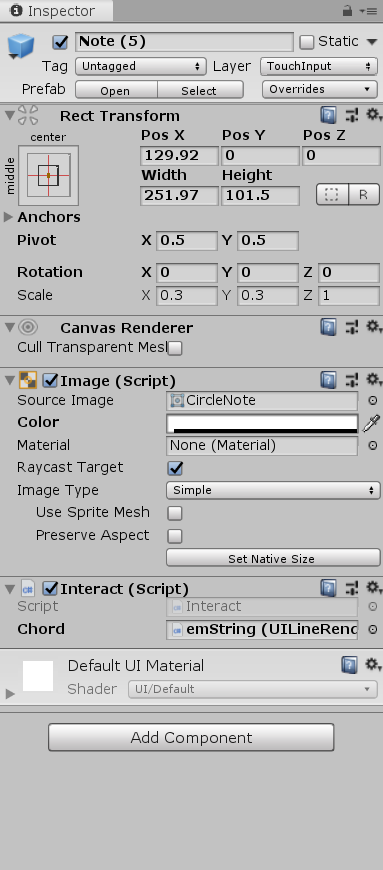
În folderul Audio/Strings sunt toate înregistrarile notelor de pe chitară. Cu acestea se va putea cânta în simulator.

Prefabs conține toate prefabricatele de care va fi nevoie. De aici se pot modifica cu ușurință, iar modificările vor apărea automat și în scenă. Prefabricatele în Unity au avut un update în versiunea de 2019.1 și acum pot exista prefabricate în prefabricate.

Folderul de Scenes conține toate scenele din joc. O scenă este totalitatea obiectelor și scripturilor încărcate la un moment dat. Atunci când o scenă se schimbă se face un refresh la memorie, dar va trece ceva timp până se va încărca următoarea scenă.

Consola este folosită pentru a face debug-ing pe aplicație atunci când nu funcționează ceva sau trebuie facute teste prin oprirea jocului la un moment dat, prin bifarea ErrorPause și folosirea în script a Debug.LogError("Error");



 Inspectorul este locul unde se pot vedea, modifica și adăuga sau șterge componentele unui obiect. Toate elementele UI au o componentă RectTransform unde gasim pivotul și modul de ancorare. Toate acestea sunt făcute in localTransform. Tag-ul este o metodă de sortare și verificare a obiectelor. Layerul are legătură cu fizica jocului și cu interacțiunile obiectelor. De exemplu se pot dezactiva anumite layere pentru a nu face contact cu Raycast-urile.

Componenta de Image conține sursa imagini, un mod de colorare de tip multiply, tipul de imagine, simplu, tiled, fill sau sliced. Tipul fill se foloseste cel mai des pentru loading bars sau bări de HP.

Pentru componentele script se pot face referințe externe în inspector pentru a nu adăuga o căutare sau link la pornirea aplicaîiei.

# CAPITOLUL 3

## Scripturi folosite

### Constante și locații

Această clasă include toate constantele pentru a fi mai ușor de accesat. Conține și toate subfolderele folderului Resources, unde sunt ținute toate prefabricatele și fișierele audio.

public class PathsAndConstants : MonoBehaviour

{

public static string NOTE\_PATH = "Audio/Strings/";

public static string PREFAB\_PATH = "Prefabs/";

public const int PENALTY\_EASY = 1;

public const int PENALTY\_MEDIUM = 2;

public const int PENALTY\_HARD = 3;

public static Color GetStringColor(StringColors color)

{

Color c = Color.red;

switch (color)

{

case StringColors.EColor:

c = Color.red;

break;

case StringColors.AColor:

c = Color.yellow;

break;

case StringColors.DColor:

c = new Color(0.035f, 0.91f, 0.92f);

break;

case StringColors.GColor:

c = new Color(0.93f, 0.45f, 0.047f);

break;

case StringColors.BColor:

c = Color.green;

break;

case StringColors.eColor:

c = Color.magenta;

break;

}

return c;

}

public enum StringColors

{

EColor,

AColor,

DColor,

GColor,

BColor,

eColor

}

public enum GameDificulty

{

Easy,

Medium,

Hard

}

public static Dictionary<GameDificulty, int> DictDificulty = new Dictionary<GameDificulty, int>()

{

{GameDificulty.Easy, PENALTY\_EASY},

{GameDificulty.Medium, PENALTY\_MEDIUM},

{GameDificulty.Hard,PENALTY\_HARD}

};

}

### Librăria NAudio

 NAudio este un API open source pentru .NET scris în C# de Mark Heath, cu contribuția multor alți dezvoltatori. Dispune de o multitudine de clase foarte utile pentru a construi propriile aplicații audio.

Utilități:

* Play back audio folosind o varietate de APIs
  + WaveOut
  + DirectSound
  + ASIO
  + WASAPI
* Full MIDI event model
  + Citește și scrie fișiere MIDI
  + Răspunde la mesaje MIDI
  + Trimite event-uri MIDI

Funcția PlaySelectedMidi este folosită pentru a citi fisierul midi salvat în resurse:

public void PlaySelectedMidi(string midiName = "AllNotesInOrder")

{

//Loading midi file "mainSong.bytes" from resources folder

//Its a midi file, extension has been changed to .bytes manually

TextAsset asset = Resources.Load(midiName) as TextAsset;

Stream s = new MemoryStream(asset.bytes);

//Read the file

midi = new MidiFile(s, true);

//Ticks needed for timing calculations

ticks = midi.DeltaTicksPerQuarterNote;

StartPlayback();

}

Funcția StartPlayback începe să treacă prin melodia MIDI încărcată

public void StartPlayback()

{

//9 is the number of the track we are reading notes from

//you'll have to experiment with that, i cant remember why i chose 9 here

for (int i = 1; i < midi.Events.Tracks; i++)

{

foreach (MidiEvent note in midi.Events[i])

{

//If its the start of the note event

if (note.CommandCode == MidiCommandCode.NoteOn)

{

//Cast to note event and process it

NoteOnEvent noe = (NoteOnEvent)note;

NoteEvent(noe, i);

}

}

}

}

Funcția NoteEvent apelată când o notă este cântată

public void NoteEvent(NoteOnEvent noe, int track)

{

//The bpm(tempo) of the track

float bpm = 150;

//Time until the start of the note in seconds

float time = (60 \* noe.AbsoluteTime) / (bpm \* ticks);

int noteNumber = noe.NoteNumber;

//Start coroutine for each note at the start of the playback

//Really awful way to do stuff, but its simple

StartCoroutine(CreateAction(time, noteNumber, noe.NoteLength, track));}

### InputManager

Clasa este împărțită în două tipuri de input. Unul pentru mouse pentru a putea testa la calculator, în inspector, și unul pentru touch, pentru a funcționa pe dispozitive mobile.

La atingerea ecranului se va creea un raycast spre pozitia UI după convertirea poziției degetului la world space. Un Raycast este o dreaptă care întoarce obiectele atinse pe parcursul iei.

La atingerea unei taste a chitări un mesaj este trimis către tasta respectivă pentru a scoate sunetul și verifica în cazul de joc, dacă nota este în partea potrivită pentru a mări scorul și a creste combo-ul.

public class TouchInput : MonoBehaviour

{

public LayerMask touchInputMask;

private List<GameObject> touchList = new List<GameObject>();

private GameObject[] touchesOld;

private Camera camera;

private RaycastHit hit;

private bool[] touchActive = new bool[10];

private GraphicRaycaster m\_Raycaster;

private PointerEventData m\_PointerEventData;

private EventSystem m\_EventSystem;

private List<RaycastResult> results;

private void Start()

{

camera = GetComponent<Camera>();

for (int i = 0; i < touchActive.Length; i++)

{

touchActive[i] = false;

}

//Fetch the Raycaster from the GameObject (the Canvas)

m\_Raycaster = GetComponent<GraphicRaycaster>();

//Fetch the Event System from the Scene

m\_EventSystem = GetComponent<EventSystem>();

}

// Update is called once per frame

private void Update()

{

///---------New Input

#if UNITY\_EDITOR

if (Input.GetMouseButton(0) || Input.GetMouseButtonDown(0) || Input.GetMouseButtonUp(0))

{

touchesOld = new GameObject[touchList.Count];

touchList.CopyTo(touchesOld);

touchList.Clear();

//Set up the new Pointer Event

m\_PointerEventData = new PointerEventData(m\_EventSystem);

//Set the Pointer Event Position to that of the mouse position

m\_PointerEventData.position = Input.mousePosition;

//Create a list of Raycast Results

results = new List<RaycastResult>();

//Raycast using the Graphics Raycaster and mouse click position

m\_Raycaster.Raycast(m\_PointerEventData, results);

foreach (RaycastResult result in results)

{

GameObject recipient = result.gameObject;

touchList.Add(recipient);

//Check if click in editor case

if (Input.GetMouseButton(0))

recipient.SendMessage(Messages.ON\_TOUCH\_STAY, result.screenPosition, SendMessageOptions.DontRequireReceiver);

if (Input.GetMouseButtonDown(0))

recipient.SendMessage(Messages.ON\_TOUCH\_DOWN, result.screenPosition, SendMessageOptions.DontRequireReceiver);

if (Input.GetMouseButtonUp(0))

recipient.SendMessage(Messages.ON\_TOUCH\_UP, result.screenPosition, SendMessageOptions.DontRequireReceiver);

}

GameObject recipient = results[0].gameObject;

touchList.Add(recipient);

if (Input.GetMouseButton(0))

recipient.SendMessage(Messages.ON\_TOUCH\_STAY, results[0].screenPosition, SendMessageOptions.DontRequireReceiver);

if (Input.GetMouseButtonDown(0))

recipient.SendMessage(Messages.ON\_TOUCH\_DOWN, results[0].screenPosition, SendMessageOptions.DontRequireReceiver);

if (Input.GetMouseButtonUp(0))

recipient.SendMessage(Messages.ON\_TOUCH\_UP, results[0].screenPosition, SendMessageOptions.DontRequireReceiver);

foreach (GameObject g in touchesOld)

{

if (!touchList.Contains(g))

{

g.SendMessage(Messages.ON\_TOUCH\_EXIT, hit.point, SendMessageOptions.DontRequireReceiver);

}

}

}

#endif

//Check input on touchscreen case

if (Input.touchCount > 0)

{

touchesOld = new GameObject[touchList.Count];

touchList.CopyTo(touchesOld);

touchList.Clear();

int nr = 0;

foreach (Touch touch in Input.touches)

{

if (!touchActive[nr])

{

Ray ray = camera.ScreenPointToRay(touch.position);

if (Physics.Raycast(ray, out hit, touchInputMask))

{

GameObject recipient = hit.transform.gameObject;

touchList.Add(recipient);

switch (touch.phase)

{

case TouchPhase.Began:

recipient.SendMessage(Messages.ON\_TOUCH\_DOWN, hit.point, SendMessageOptions.DontRequireReceiver);

touchActive[nr] = true;

break;

case TouchPhase.Stationary:

case TouchPhase.Moved:

recipient.SendMessage(Messages.ON\_TOUCH\_STAY, hit.point, SendMessageOptions.DontRequireReceiver);

break;

case TouchPhase.Ended:

recipient.SendMessage(Messages.ON\_TOUCH\_UP, hit.point, SendMessageOptions.DontRequireReceiver);

break;

case TouchPhase.Canceled:

recipient.SendMessage(Messages.ON\_TOUCH\_EXIT, hit.point, SendMessageOptions.DontRequireReceiver);

break;

}

}

}

nr++;

}

foreach (GameObject g in touchesOld)

{

if (!touchList.Contains(g))

{

g.SendMessage(Messages.ON\_TOUCH\_EXIT, hit.point, SendMessageOptions.DontRequireReceiver);

}

}

}

}

}

### Gameplay

Un ObjectPooler este o clasă care poate refolosi prefabricate deja instanțiate care nu mai sunt active în scenă. Pentru o eficiență mai mare, nu are rost să distrugi un obiect dacă stii că vei avea nevoie de el mai târziu, de aceia, mai bine se creează o listă cu obiecte, dacă e nevoie de un obiect, se verifică lista pentru obiecte inactive. Dacă nu a fost găsit nici unul se instanțiază unul nou, dacă va trebui înlăturat obiectul, el se va dezactiva pentru a putea fi folosit mai încolo când va fi nevoie de altul.

public class ObjectPooler : MonoBehaviour

{

private List<GameObject> notePool = new List<GameObject>();

public Transform NoteParrent;

public GameObject CheckNotePool(string noteName, PathsAndConstants.StringColors color)

{

foreach (var note in notePool)

{

if (!note.activeSelf)

{

GameObject newNote = note;

newNote.SetActive(true);

newNote.GetComponent<ToHitNote>().SetupNote(noteName, color);

return newNote;

}

}

return AddNoteInPool(noteName, color);

}

public GameObject AddNoteInPool(string noteName, PathsAndConstants.StringColors color)

{

GameObject newNote = Instantiate(Resources.Load<GameObject>(PathsAndConstants.PREFAB\_PATH + "NoteToHit"));

newNote.transform.SetParent(NoteParrent);

notePool.Add(newNote);

newNote.GetComponent<ToHitNote>().SetupNote(noteName, color);

return newNote;

}

}

Pentru a adăuga nota pe poziția tastei potrivite, SpownNote conține toate pozițiile tastelor ca și copii ai obiectului, de aceea vom folosi transform.GetChild(poz) pentru a obtine poziția prestabilită.

public void SpownNote(int poz, StringColors stringColor, string noteName, float duration = 0)

{

var note = objectPooler.CheckNotePool(noteName, stringColor);

note.transform.position = transform.GetChild(poz).transform.position;

}

Modul de operare al unei note. Miscarea este continuă în jos. Are o funcție pentru a o porni, și una pentru a combina culorile în caz de corzi multiple pe aceiași tastă.

public class ToHitNote : MonoBehaviour

{

private int NumberOfStrings = 1;

private float speed = 70f;

private Image imgComponent;

private string noteName;

public float destroyDistance = 4;

private void Awake()

{

imgComponent = Instantiate(Resources.Load<GameObject>("Prefabs/CircleFill"), transform).GetComponent<Image>();

}

//setup the note with the correct name and color

public void SetupNote(string note\_Name, StringColors color)

{

noteName = note\_Name;

imgComponent.color = GetStringColor(color);

}

private void Update()

{

Move();

}

//setup the note with the correct name and color

private void Move()

{

transform.position = new Vector2(transform.position.x, transform.position.y - speed \* Time.deltaTime);

CheckDestroy();

}

//combine notes colors for multicolor note for multiple strings on the same fret

public void GetNoteColor(int numberOfStrings, Color[] colors)

{

NumberOfStrings = numberOfStrings;

float fillPrecent = 1f / (float)numberOfStrings;

float rotation = 360 / numberOfStrings;

for (int i = 0; i < numberOfStrings; i++)

{

var part = Instantiate(Resources.Load<GameObject>("Prefabs/CircleFill"), transform);

var imgComponent = part.GetComponent<Image>();

imgComponent.color = colors[i];

imgComponent.fillAmount = fillPrecent;

part.transform.localRotation = Quaternion.Euler(0, 0, rotation \* i);

}

}

//check the distance between note position and the destroy position

private void CheckDestroy()

{

if (transform.localPosition.y < destroyDistance)

{

gameObject.SetActive(false);

}

}

}

Interact este clasa care controlează abilitatea de bend a unei corzi. Este pusă pe fiecare notă și reacționează la TouchInput cu events.

public class Interact : MonoBehaviour

{

private Image sprite;

private Vector2 oldPosition;

private float speed = 0.3f;

private Vector2 \_startPosition;

private Vector2 \_offsetToMouse;

private Transform viewPoint;

private const float MAX\_BEND = 120;

public UILineRenderer Chord;

private void Start()

{

sprite = GetComponent<Image>();

viewPoint = transform.GetChild(0);

\_startPosition = transform.position;

}

private void OnTouchDown(Vector2 point)

{

oldPosition = point;

\_offsetToMouse = \_startPosition - point;

Chord.Points[2].x = transform.localPosition.x;

}

private void OnTouchUp()

{

transform.position = \_startPosition;

viewPoint.localPosition = Vector2.zero;

Chord.Points[2].y = 0;

Chord.SetAllDirty();

}

private void OnTouchStay(Vector2 point)

{

transform.position = point + \_offsetToMouse;

viewPoint.position = new Vector2(\_startPosition.x, transform.position.y);

if (\_startPosition.y - MAX\_BEND >= point.y + \_offsetToMouse.y)

viewPoint.position = new Vector2(\_startPosition.x, \_startPosition.y - MAX\_BEND);

if (\_startPosition.y + MAX\_BEND <= point.y + \_offsetToMouse.y)

viewPoint.position = new Vector2(\_startPosition.x, \_startPosition.y + MAX\_BEND);

Chord.Points[2].y = transform.localPosition.y;

Chord.SetAllDirty();

//if (\_startPosition - MAX\_BEND <= point.y + \_offsetToMouse && \_startPosition + MAX\_BEND >= point.y + \_offsetToMouse)

// transform.position = new Vector2(transform.position.x, point.y + \_offsetToMouse);

}

private void OnTouchExit()

{

transform.position = \_startPosition;

viewPoint.localPosition = Vector2.zero;

}

}

Lista Figurilor

[Imaginea 1 – Setup chitară electrică 2](#_Toc18959457)

[Imaginea 2 – Rocksmith 2014 3](#_Toc18959458)

[Imaginea 3 – Garage Band IOS 3](#_Toc18959459)

[Imaginea 4 – Main menu GuitarSimChar 4](#_Toc18959460)

[Imaginea 5 – Elemente ale chitării 5](file:///C:\Users\GRyFf\Documents\GitHub\DesertatieGuitar\Lucrare_Disertatie_Coada_Catalin.docx#_Toc18959461)

[Imaginea 6 – Acordul C 5](#_Toc18959462)

[Imaginea 7 - Tehnica de Bend 6](#_Toc18959463)

[Imaginea 8 - Tehnica de Tapping 6](#_Toc18959464)

# Bibliografie

1. Dan V. Popa, 2014, O primă carte despre Unity 3D, vol. I, Bacău: Editura Alma Mater,pagina 54
2. Creating a Start Menu / Main Menu in Unity 5 <https://www.youtube.com/watch?v=FrJogRBSzFo> Data 05.06.2017
3. Dr. Kris Jamsa, Lars Klander, Totul despre C și C++,Editura: Teora 2015
4. UI Events - Make buttons do something <https://www.youtube.com/watch?v=kQ2Qc_0MIyI&t=295s> Data 02.06.2017
5. Set Objects Child to Active/Inactive <http://answers.unity3d.com/questions/894211/set-objects-child-to-activeinactive.html> Data 05.06.2017
6. Alex Okita, Learning C Programing With Unity 3D, CRC Press 2015
7. <https://docs.unity3d.com/Manual/CreatingAndUsingScripts.html> Data 17.04.2017
8. [https://docs.unity3d.com/ScriptReference/Rigidbody.MovePosition.html Data 27.05.2017](https://docs.unity3d.com/ScriptReference/Rigidbody.MovePosition.html%20Data%2027.05.2017)
9. <https://www.youtube.com/watch?v=5yT92DNqrJs> Data 16.06.2017
10. [https://docs.unity3d.com/Manual/nav-MoveToDestination.html Data 13.06.2017](https://docs.unity3d.com/Manual/nav-MoveToDestination.html%20Data%2013.06.2017)
11. <https://unity3d.com/>

|  |  |  |
| --- | --- | --- |
|  | **UNIVERSITATEA „VASILE ALECSANDRI” DIN BACĂU**  **Facultatea de Științe**  **Str. Calea Mărăşeşti, nr. 157, Bacău, 600115**  **Tel. ++40-234-542411, tel./ fax ++40-234-571012** [*www.ub.ro*](http://www.ub.ro)*; e-mail:* [*stiinte@ub.ro*](mailto:stiinte@ub.ro) |  |

DECLARAŢIE DE AUTENTICITATE

privind elaborarea lucrării de licență

Subsemnatul/subsemnata………………………………………………………………..

declar pe propria răspundere că:

1. lucrarea a fost elaborată personal şi îmi aparţine în întregime;
2. nu au fost folosite alte surse decât cele menţionate în bibliografie;
3. nu au fost preluate texte, date sau elemente de grafică din alte lucrări sau din alte surse fără a fi citate şi fără a fi precizată sursa preluării, inclusiv în cazul în care sursa o reprezintă alte lucrări ale mele;
4. lucrarea nu a mai fost folosită în alte contexte de examen sau de concurs.

Data, Semnătura,