# Bomberman

Echipă:

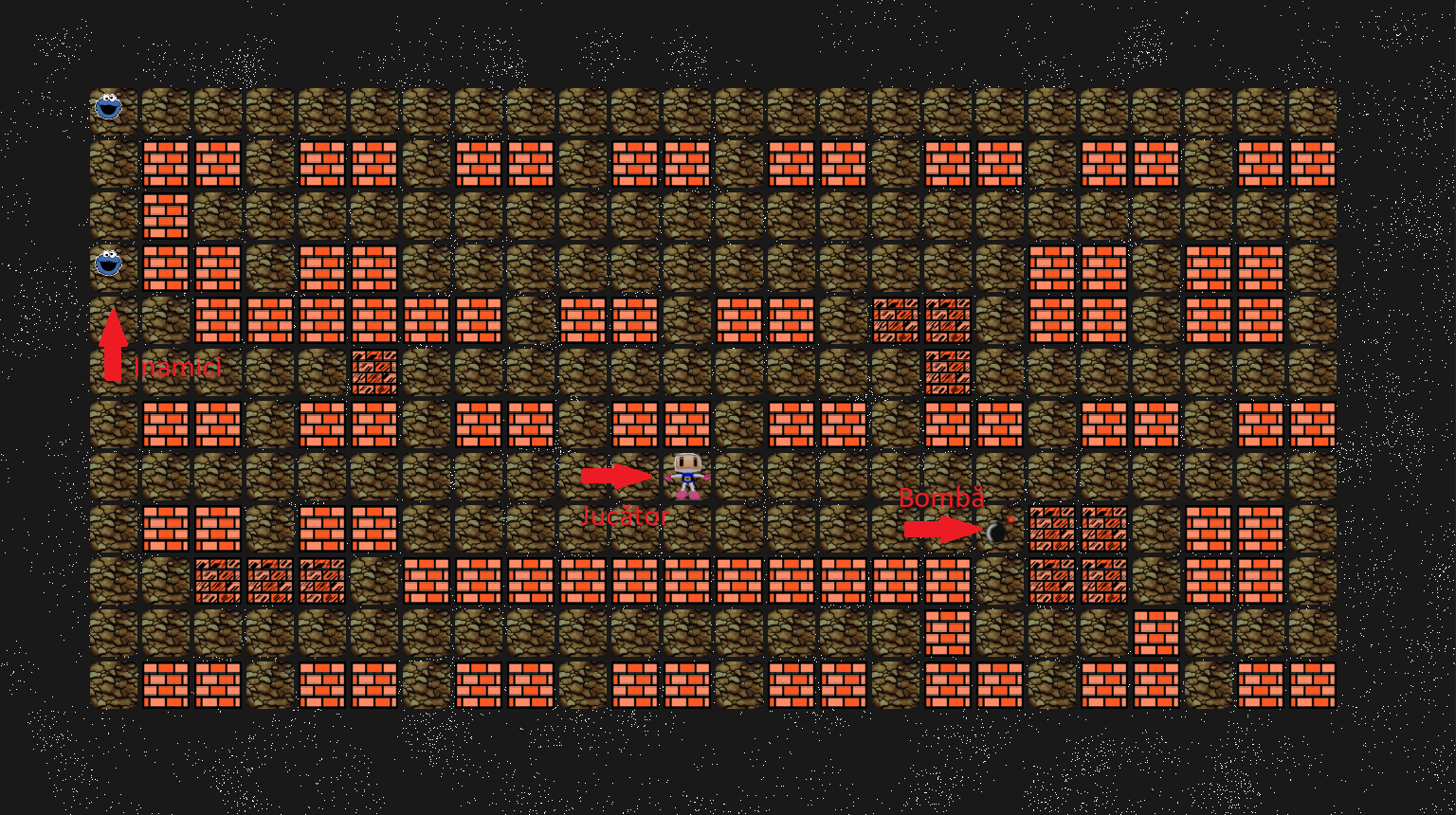
* Bardaș Denis Adelin
* Brănaci Șerban Mihai
* Chindea Cosmin Mihai

GitHub:

<https://github.com/Serban8/Bomberman>

Descriere

Echipa noastră a ales să realizeze jocul Bomberman. Scopul jocului este ca jucătorul să elimine inamicii prezenți pe hartă și să avanseze la nivelul următor. El folosește bombele pentru a sparge blocurile care stau în calea sa, cât și pentru a distruge adversarii.



Harta unui nivel al jocului. Am evidențiat principalele elemente: jucătorul, bomba și inamicii.

Codul proiectului a fost scris în limbajul C#, folosind IDE-ul Visual Studio.

Soluția de Visual Studio cuprinde 3 proiecte distincte, pentru fiecare componentă a jocului: BombermanBase pentru logică, BombermanMONO pentru partea de UI și BombermanTests pentru partea de testare.

UI

Pentru realizarea părții de UI am folosit framework-ul MonoGame A red and black logo

Description automatically generated Acesta are diverse avantaje, ușurând organizarea și afișarea elementelor grafice care constituie mediul de joc pentru utilizator.

A screenshot of a computer

Description automatically generated

*Proiectul responsabil cu afisarea interfeței este de tip MonoGame Desktop Application si are o structura specifica MonoGame:*

*În folderul Content se afla texturile, fonturile si celelalte asset-uri folosite pentru interfata grafica*

*LogicExtentions contine metode de extensie ale interfetelor din DLL-ul cu logica aplicatiei, pentru a putea afisa fiecare element si updata logica aferenta UI-ului*

*UIHelpers contine diverse clase specifice MonoGame create pentru a respecta principiile clean code si pentru a face cod reutilizabil si mentenabil*

*Clasa Bomberman.cs este "managerul" partii de UI, centralizand informatiile si instructiunile specifice UI.*



*Sprite-urile folosite pentru afișarea jocului.*

Framework-ul Monogame dispune de funcții și clase speciale pentru afișarea pe ecran a resurselor. În proiect, aceste clase și funcții sunt utilizate în principal în clasa Bomberman.cs, care administrează interfața grafică a jocul, dar și în alte locuri. Această clasă se găsește în cadrul proiectului BombermanMONO, care este proiectul de front-end al jocului. În acesta am folosit clase specifice precum GraphicsDeviceManager, SpriteBatch și Vector2.

La deschiderea jocului, resursele sunt încărcate de pe disk. Apoi, folosind funcțiile specifice MonoGame, elementele sunt prelucrate și afișate pe ecran.

A screen shot of a computer program

Description automatically generated

A computer screen shot of text

Description automatically generated

Funcțiile pentru încărcarea resurselor.

Back-End

Proiectul respecta normele OOP și diversele convenții specifice C#, utilizând denumiri de variabile specifice, interfețe, polimorfism și moștenire, printre altele. De asemenea, în realizarea proiectului am ținut cont de principiile Clean Code, pentru a păstra codul ușor de citit și mentenabil.

Din punct de vedere al design pattern-urilor, Observer, Factory și Strategy ne-au fost de ajutor în structurarea codului.

A screenshot of a computer

Description automatically generated

*Ierarhia proiectului cu cele 3 componente:*

*BombermanTests,*

*BombermanBase*

*și BombermanMONO*

*Partea de logica a aplicatiei este conținută intr-un proiect de tip DLL. Pe scurt, in acesta avem:*

*Clasele principale IEntity, ITile, ITileMap, IBomberman, care sunt elementele de baza ale aplicatiei. Aceste interfete sunt publice, iar ele sunt concretizare in clase interne.*

*Entitatile pot fi de doua tipuri: Player sau Enemy. De aici reiese si nevoia aplicarii design pattern-urilor Factory si Strategy. Factory initializeaza Entity-ul in functie de tip, Player sau Enemy, in timp ce strategy selecteaza cum se actioneaza la apelarea metodei Move*

*Pe langa acestea, se pot remarca diverse principii si implementari specifice OOP, cum ar fi polimorfismul, interfetele sau incapsularea*

*IBomberman.cs este clasa principala, concretizata prin Bomberman.cs. Aceasta trebuie instantiata si utilizata pentru a crea un joc.*

A screenshot of a computer program

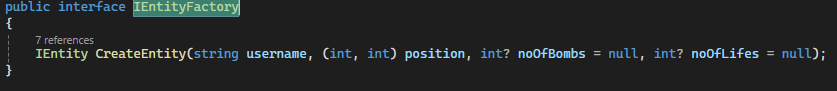
Description automatically generated

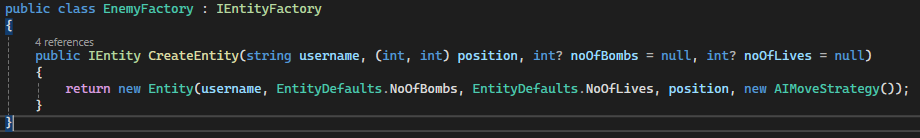
Design Patterns

Design pattern-urile folosite în proiectul nostru sunt:

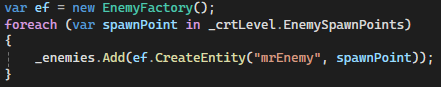
* Observer
* Factory
* Strategy

**Factory** - acest design pattern a fost folosit la crearea entitățior de Player și Enemy. Deși vizual aceste entități sunt complet diferite, în cod este folosita doar clasa Entity, care este configurată cu ajutorul design pattern-ului Factory în funcție de tipul real al entității.





Factory Design Pattern





Inițializarea player și enemy

**Strategy** – acest design pattern a fost folosit pentru a putea utiliza metoda Move atât pentru a mișca un Player, cât și pentru a realiza mișcarea automată a unui Enemy.

A black background with white text

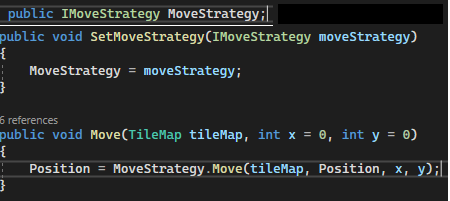
Description automatically generated

Implementarea interfeței IMoveStrategy

A computer screen with text on it

Description automatically generated

Implementarea strategiei de mișcare a player-ului



Utilizarea strategiei de mișcare

**Observer** - folosit pentru a notifica proiectul MonoGame asupra diverselor evenimente care apar odată cu derularea jocului. Conform definiției acestui design pattern, definim un Observer si un Observable.

Astfel, avem interfata IBombermanObserver care este implementată de clasa principala din Mono, Bomberman.cs.

A screen shot of a computer code

Description automatically generated

Definirea interfeței IBombermanObserver



Moștenirea interfeței

A screen shot of a computer screen

Description automatically generated

Implementarea metodelor interfeței

Clasa Observable este Bomberman.cs, care implementează funcții specifice, precum adăugarea si scoaterea subscriberilor, respectiv notificarea subscriberilor in functie de eveniment

A screenshot of a computer program

Description automatically generated

Unit Testing

Pentru proiectul de Unit Testing am folosit frameworkul MSTests.

În cadrul unit testelor am verificat corectitudinea codului scris din punct de vedere logic. Scopul principal este asigurarea ca fiecare metoda scrisă își îndeplinește scopul și nu are diverse efecte secundare. Am început prin a face teste simple, de exemplu, testarea că inițializarea entităților se face corect în cadrul testului TestEntitiesInitialization().

A computer screen with text

Description automatically generatedÎn continuare am verificat ca funcționalitățile jucătorului să funcționeze corect în limita hărții definită la începutul jocului prin teste precum TestPlayerMoveOutOfMap(), TestPlacingBomb() și LosingLife().

În următorii pași se testează funcționalitățile hărții și modul în care jucătorul interacționează cu ea. Testele cuprind și funcționalități mai complexe ale proiectului și modul de relaționare al proiectelor între ele, unde am folosit mai mult decât teste simple.

Diagrame

(Notă: pentru a vizualiza diagramele se poate da zoom – nu se pierde calitatea la zoom-in)

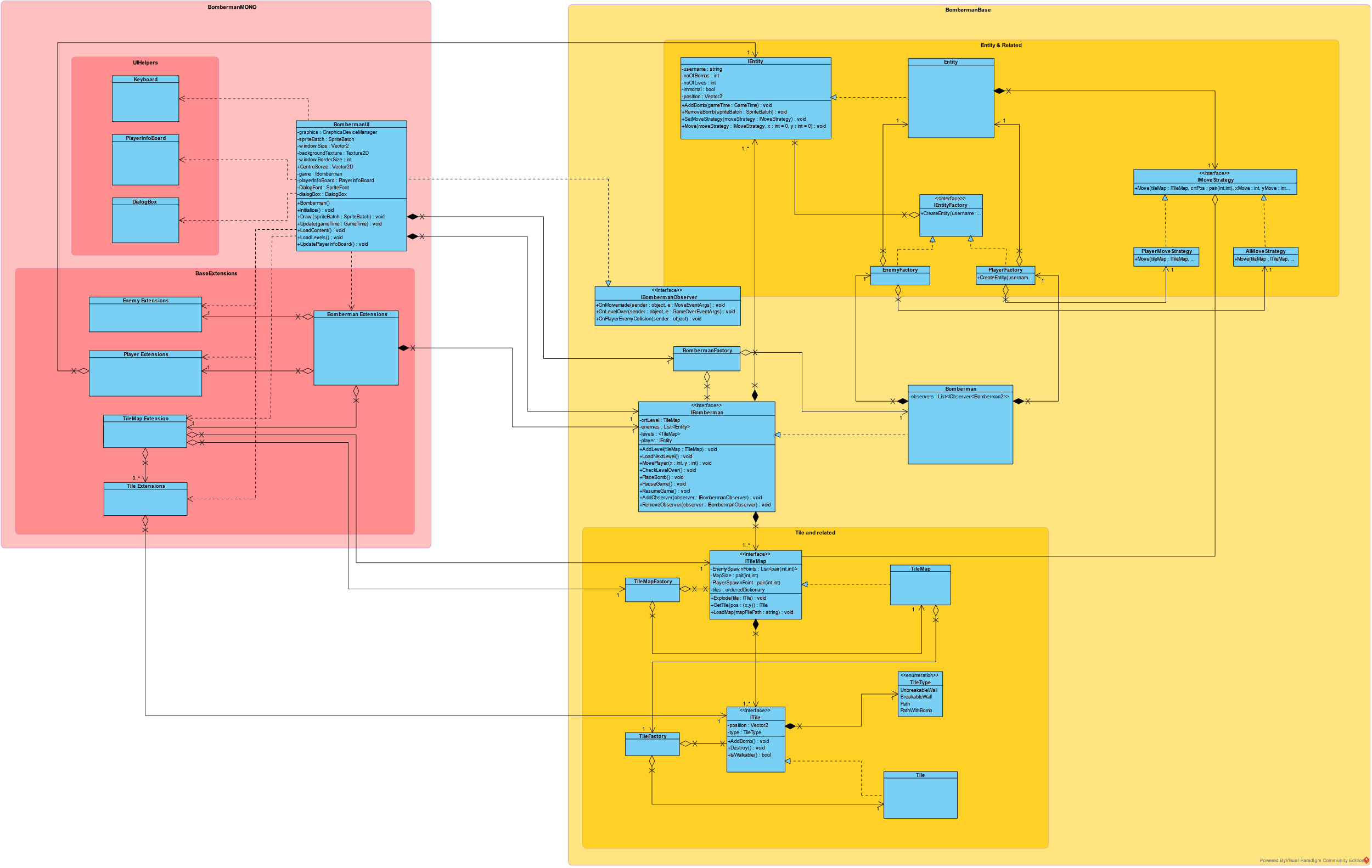
**Class**

Diagrama arată relațiile dintre clasele proiectului de baza (galben), cele dintre clasele proiectului de UI (rosu-roz), cât și legăturile dintre cele doua proiecte.

**A diagram of a diagram

Description automatically generatedUse Case**

Diagrama prezintă cum jucătorul interacționează prima dată cu meniul programului și apoi cu partea de joc. Adversarul este un actor virtual de aceea el poate interacționa doar cu partea de joc.

**A diagram of a game

Description automatically generatedActivity**

Diagrama de activități prezintă la scară largă toate starile prin care trece jucatorul, interfața grafică și jocul. Prima dată jucatorul deschide jocul. Interfața afișează meniul de start unde jucatorul introduce numele. Cu acea informație se inițializează jocul. Jocul trece prin schimburi de informație cu jucatorul iar la final există 2 stări posibile, cea de câștig a jocului și de pierdere a jocului.

**A diagram of a game

Description automatically generatedState Diagram**

Diagrama de stări prezintă stările jocului: în meniu, desfășurarea jocului, joc terminat și joc câștigat sau pierdut.

**Sequence A diagram of a computer process

Description automatically generated**

Aici evidențiem cum din punct de vedere temporal se desfășoară acțiunea în program. Jucătorul trimite un input interfeței. Această îl transmite mai departe către partea de logică. Aici se fac modificările necesare iar apoi ele sunt trimise către interfață pentru a fi afișate utilizatorului.

**Communication**

A diagram of a user interface

Description automatically generated

Diagrama de comunicare arată schimbul de informație de la jucător la cele 2 componente ale programului.

Concluzii

În concluzie, putem spune ca în realizarea acestui proiect am acumulat cunoștiințe noi în domeniul ingineriei software și, în general, al informaticii. Am învățat cum să structuăm un proiect care să utilizeze design patterns și am pus în valoare diagramele de modelare. De asemenea, am învățat diverse practici folositoare în dezvoltarea software, precum separarea logică-UI, lucratul cu interfețe sau utilizare texturilor.