Pokemon battles

Facultatea de matematică și informatică
Informatică aplicată

Tiba Ștefan Vlad
grupa 10LF333

Pentru laboratorul de la materia Medii și instrumente de programare (MIP) am ales să creez un joc în consolă, inspirat din franciza Pokemon, cu numele Pokemon Battles. În acest joc îți alegi o creatură numită Pokemon cu care vei lupta împotriva altor Pokemoni. Fiecare Pokemon are un nume, tip, statistici (atac, protectie, puncte de viață și viteză), listă de atacuri și nivel.

Fiecare jucător are un nume, ID, o listă de Pokemoni (care momentan este redusă la un singur obiect), și spațiul activ, adică Pokemonul folosit în luptă.

Există un singur mod de joc, momentan, numit “Single Battle” în care se desfășoară o singură bătălie între doi jucători, care la început își aleg câte un Pokemon. Jucătorul a cărui Pokemon ajunge să aibă 0 puncte de viață pierde. Fiecare jucător trebuie să aleagă pe rând o mișcare pe care să o folosească Pokemonul din spațiul activ. Fiecare atac are puncte de putere (>= 0), tip (același sau diferit de utilizatorul mișcării) și un efect pe care îl poate aplica pe oponent (în funcție de șansa pe care o are). După alegerea mișcării, Pokemonul cu viteza mai mare va fi cel care va folosi primul mișcarea. Punctele de viață pe care le va scădea oponentului sunt calculate prin următoarea formulă:

DMG = {[(2 * lvl / 5 + 2) * pwr * att / def] / 50 + 2 } * STAB * eff

unde:

- lvl = nivelul pokemonului
- pwr = punctele putere ale mișcării folosite de Pokemon
- att = atacul Pokemonului care folosește mișcarea
- def = protecția Pokemonului pe care este folosită mișcarea
- STAB = 1.5 dacă mișcarea este de același tip ca al Pokemonului care o folosește sau = 1.0 opusul
- eff = 2.0 dacă tipul mișcării folosite este eficientă împotriva tipului Pokemonului oponentului (adică dacă atacul este de tip apă, iar oponentul este de tip foc, atunci eff = 2.0) sau = 0.5 tipul Pokemonului pe care este folosită mișcarea este rezistent împotriva tipului atacului utilizat (foc atacă apă) sau = 1.0 dacă niciuna dintre cele de mai sus nu este îndeplinită

Această formulă a fost luată și modificată din primul joc lansat de Pokemon, anume Pokemon Red/Green versiunea în japoneză.

Fiecare efect al unei mișcări poate varia de la “Oponentului i se va scădea protecia cu 20%” până la aplicarea unei “boli” cum ar fi “ardere” care după fiecare tură scade puncte viața celui care are această “boală”, “plantat” care funcționează ca “ardere”, dar acele puncte viață scăzute oponentului se adaugă pokemonului care a folosit atacul.

Astfel, bătălia continuă până când unul dintre Pokemoni rămâne fără puncte viață.

Lista tipurilor de Pokemoni existente până acum:

Iarbă/natură (rezistentă la apă, slăbiciune la foc), foc (rezistență la Iarbă, slăbiciune la apă), apă (rezistență la foc, slăbiciune la Iarbă), normal (rezistență la Iarbă, slăbiciune la piatră), piatră(rezistență la normal, slăbiciune la apă), electricitate(rezistență la electricitate, slăbiciune la piatră).

Lista Pokemonilor introduși până acum în joc:
Bulbasaur (iarbă), Charmander (foc), Squirtle (apă), Rattata (Normal), Geodude(piatră), Pikachu (electricitate).

Pentru proiectu am utilizat clasele:

Denumire clasă	membrii	metode
Player	(Toți publici) ● m_name (String) ● m_playerID (int) ● m_pokemonsOwned (listă de obiecte Pokemon) ● m_activeSpot (obiect Pokemon)	● getName ● showPokemonOwned (care afișează pokemonii deținuți de un jucător) ● changePokemonInActive Spot (schimba Pokemonul din spațiul activ cu unul nou) ●
Pokemon	privați: ● m_name (String) ● m_type (enum Type) ● m_owner (obiect de tip Player) ● m_attacks (listă de obiecte Attacks) ● m_canEvolve (boolean) - nefolosit momentan publici: ● m_stats (obiect Stats) ● m_statusCondition (enum StatusCondition)	● getteri si setteri pentru membrii privați ● getPokemonFromFile (returnează un Pokemon citit dintr-un fișier dat ca parametru (String), menționând deținătorul Pokemonului creat) ● changeHPRemaining (modifică punctele viață rămase ale obiectului scăzând cu o valoare dată) ● STAB (vezi legenda de la formula dată mai sus) ● effectiveness(vezi legenda de la formula dată mai sus “eff”) ● getDamageOfAnAttack(returneaza punctele viață pe care un atac le-ar scădea unui oponent) ● chooseAttack() returnează atacul ales de jucător ● printPokemonInfo (afișează informațiile despre un obiect Pokemon) ● applyEffectDMG (scade puncte viață în funcție de boala pe care o are obiectul Pokemon)

Stats	(toți membrii sunt privați și sunt de tip float) ● m_hp ● m_hpRemaining ● m_attack ● m_defense ● m_speed	● setteri si getteri pentru fiecare membru privat ● getStatsFromFile(în clasa Pokemon, statusurile unui Pokemon se citesc cu ajutorul acestei metode din același fișier) ● getTotalStats(returnează suma tuturor membrilor)
Attacks (clasă abstractă pentru atacurile Pokemonilor (fiecare atac are o clasă separată care moștenește clasa Attacks))	privați: ● m_attackName (String) ● m_power (int) ● m_moveType (enum Type) protected ● m_user (obiect Player)	● setteri și getteri pentru fiecare membru ● applyEffect (care este suprascrisă de clasele moștenitoare) ● printAttack (afișează numele atacului)
StatusCondition (enum class) - *None,Burned,Paralyzed, Poisoned, Seeded, Confused -*		
Type (enum class) - *Water,Fire,Grass,Normal,Rock,Electric -*	(Toți membrii sunt publici) ● name (String) ● weakTo (enum Type) ● resistantTo (enum Type) *weakTo și resistantTo sunt prestabiliți pentru fiecare tip*	● getTypeName(returnează numele tipului ca String) *weakTo și resistantTo sunt prestabiliți pentru fiecare tip*
GameMode (enum class) - *SingleBattle,Survival, ChangeOrderOfPokemon,none; -*	● option (enum GameMode) ● player (obiect Player)	● setter si getter pentru option ● selectGameMode (lăsa jucatorul sa selecteze modul de joc) ● createPlayerSetup (initializează un nou obiect Player la fel și un Pokemon ales de jucător) ● returnPokemonFile(adaugă calea numelui fișierului din care se citește un Pokemon) ● printGameModeHelp (afișează instrucțiuni)
Battle	(toți membrii sunt publici și de tip Player)	● beginBattle (care inițiază o bătălie între doi

	• player1 • player2	jucători, returnând la final câștigătorul) • printBattleScene (afișează Pokemonii activi și alegerile pe care le poate lua fiecare jucător) • takeTurn (jucătorul alege ce să facă în tura lui)

Laboratoarele folosite în proiect:

Laborator 1: Tipuri de date primitive si output

Din primul laborator am utilizat tipul de date int în clase pentru a reprezenta ID-uri, nivelul unui Pokémon și puterea unui atac. Tipul float a fost folosit pentru statusuri sau alți membri care implicau calcule cu precizie exactă, cum ar fi numărul de puncte de viață scăzute în urma unui atac. De asemenea, byte a fost utilizat ca variabilă decizională pentru a stoca valori mici, limitate la un maxim de 4 opțiuni.

Am folosit afișarea în consolă pentru date despre un Pokemon, atac, scena bătăliei, opțiunile puse la dispoziție pentru un jucător sau pentru mesaje de eroare.

Laborator 2: Input, while, for, if, switch

Din al doilea laborator am utilizat Scanner pentru citirea de date din fișiere (spre exemplu date despre un Pokemon) și de la tastatură pentru variabilele decizionale sau pentru introducerea numelui jucătorului. Am folosit afișarea în consolă pentru date despre un Pokemon, atac, scena bătăliei, opțiunile puse la dispoziție pentru un jucător sau pentru mesaje de eroare.

Funcții repetitive, precum while sau for, au fost folosite pentru a crea loop-uri la decizii (loop-ul se încheia doar în momentul alegerii unei opțiuni valide) sau pentru a parcurge o listă pentru a lucra cu anumite obiecte din interior. If sau switch au fost folosite pentru alegeri de decizii (la pachet cu while) sau pentru a stabili anumiți factori precum dacă unul dintre Pokemoni a rămas fără puncte viață după ce s-a terminat etapa de calculat punctele viață.

Laborator 3: Colecții Java (Array, List, Map)

Am utilizat List pentru a reține Pokemonii unui jucător și Priority queue pentru a memora atacurile alese de jucători, prioritatea cea mai mare fiind deținută de atacul Pokemonului cu viteza cea mai mare.

Laborator 4 și 5: Clase și moșteniri

Clasele sunt prezentate mai sus printr-un tabel, fiecare jucând un rol important în asamblarea unei logici în jocul final. Moștenirile au avut loc la nivelul clasei Attacks, care este o clasă abstractă pentru fiecare atac în parte. Fiecare atac are o clasă separată,

moștenind clasa Attacks, putând astfel fi implementate efectele aplicate de fiecare în parte, la fel și puterea împreună cu numele.

Laborator 6: Interfețe

În proiect există o singură interfață, denumită IGameMode. Aceasta definește toate metodele care sunt implementate de o clasă de tip enum. Interfața IGameMode servește drept contract pentru clasa enum, asigurând o structură unitară și oferind posibilitatea de a extinde funcționalitățile jocului.

Nu am folosit mai multe interfețe în proiect deoarece, până în acest moment, nu am găsit un rol clar pentru a adăuga altele decât IGameMode. Adăugarea de interfețe doar pentru a fi implementate fără un scop bine definit ar fi fost inutilă, așa că am preferat să mă concentrez pe utilizarea eficientă a IGameMode, care împlinește necesitățile actuale ale proiectului.