Contextul jocului: jocul este situat într-o vale întunecată din nordul continentului numită ”Bone Valley” unde vrăjitori sunt antrenați în artele întunecate. Personajul principal este unul dintre absolvenți acestei ”școli”. El se întoarce în vale pentru a pune un sfârșit răului comis de către locuitori săi folosind însuși puterile obținute de la ei.

Sistemul joc: joc tile-based cu vedere de sus în care jucătorul parcurge numeroase camere pe parcursul a 3 nivele. Camerele prezintă diferiți inamici și obstacole care trebuie învinse pentru a avansa. Eroul începe cu o singură abilitate, obținând una nouă la finalul fiecărui nivel. Abilitatea inițială este ”Frostbolt” , care este un proiectil rudimentar care încetinește inamicii dar este foarte eficient din punctul de vedere al costului de mana. Mana este a 2-a resursă a jucătorului pe lângă viață. Eroul poate folosi orice abilitate atât timp cât mana s-a este pozitivă, fiecare abilitate având un cost care se scade din totalul de mană atunci când este folosită. O problemă cu această interacțiune este faptul că viteza jucătorului scade în funcție de nivelul său de mană, el având doar 25% din viteza maximă atunci când mana este negativă. Celelalte 2 abilități sunt ”Fireball”, care lansează un proiectil ce lovește toți inamici din jurul punctului de impact și ”Timestop” care oprește timpul pentru câteva secunde, permițându-i însă jucătorului să continuie să se miște. Pentru a trece de la o abilitate la alta jucătorul trebuie să apese tasta R.

Conținutul jocului:

Inamici cu care eroul se înfruntă sunt de 3 tipuri:

-Scheleți: cei mai abundenți dintre inamic, ei au puțină viață dar sunt rapizi și lovesc repede de aproape.

-Vrăjitori: ei sunt cei mai periculoși inamici. Ei nu vor ataca direct eroul, alegând în schimb să creeze la infinit monștri hostili.

-Aruncători: monștri similari cu scheleți, care au la fel de puțină viață însă sunt mai înceți și atacă de la distanță.

Niveluri: jocul este împărțit în 3 niveluri cu diferite nivele de dificultate.

Nivelul 1: inamici sunt aproape complet doar monștri de grad mic. În ultima încăpere se află un vrăjitor inamic.

Nivelul 2: Este populat de toate cele 3 tripuri de inamici. Camerele sunt împărțite în arene largi cu mult loc de manevrare și depozite mici. Nivelul se termină cu o încăpere cu multe coridoare și mulți inamici, jucătorul fiind nevoit să atace încăperea din mai multe unghiuri, eliminând Vrăjitori unul câte unul.

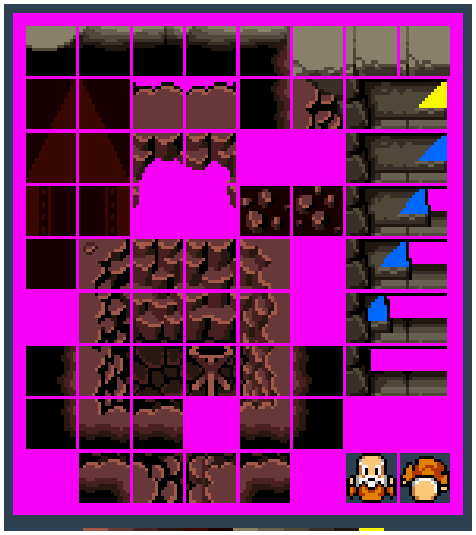
Nivelul 3: este format dintr-o singură largă încăpere unde se află ultima clasă de magie, arcana. După obținerea noii puteri, eroul este asaltat de mai multe valuri de inamici, fiecare cu compoziții diferite de inamici. Valurile sunt infinite și crescătoare în dificultate. Fiecare inamic învins în acest nivel contribuie la scorul final.

Interfața jocului:

Interfața este formată dintr-o bară ce se află în partea de jos a ecranului unde este scrisă viața, mana, abilitatea curentă și cooldown-ul curent. Meniul jocului este format din 2 butoane, Start și Exit.

Sprite-uri folosite:

Tileset-ul care îl voi folosi pentru crearea pereților unelor încăperi:

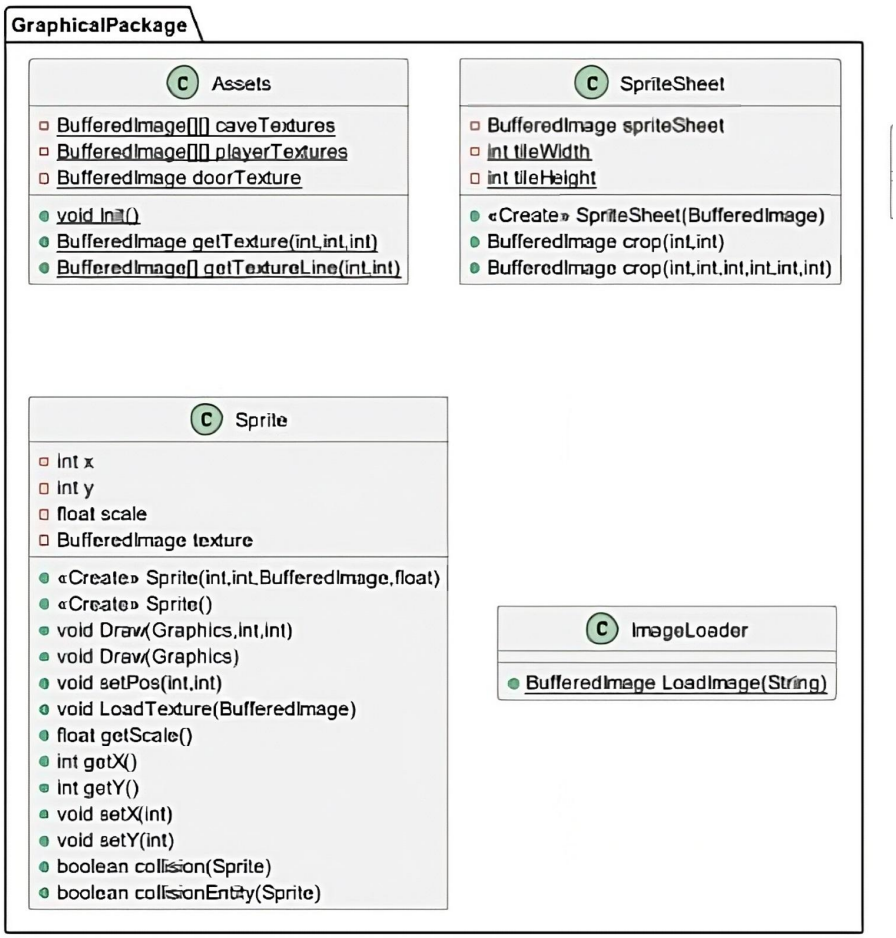


Resurse biografice utilizate:

- Tileset-ul este luat de pe www.spriters-resource.com

Arhitectural Design Document (diagrama UML):

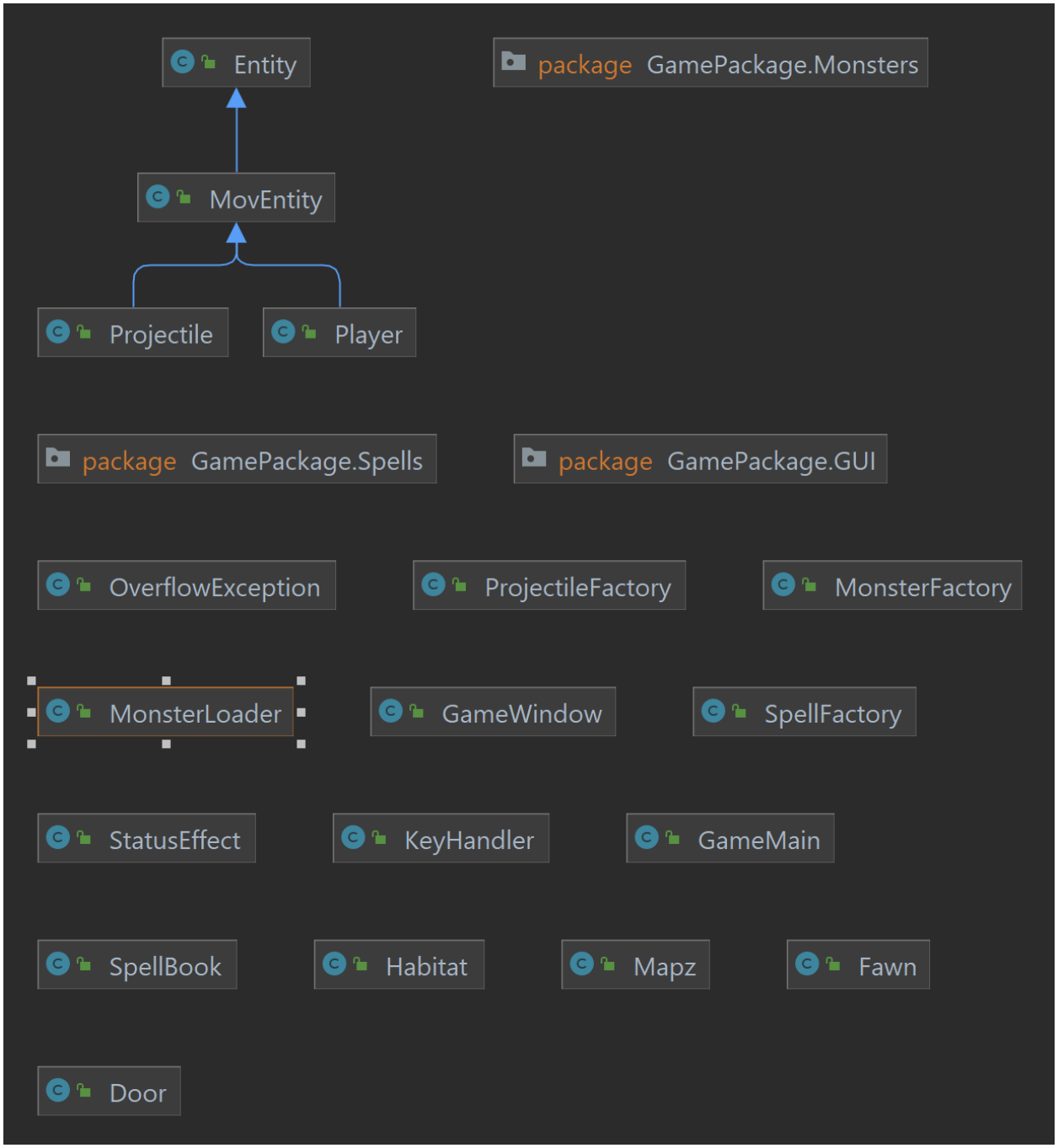
UML-ul este împărțit în 2 părți: Pachetul grafic și pachetul de joc.



Pachetul grafic stă la baza obiectelor desenate pe ecran. În clasa Assets sunt salvate toate ”Dalele” folosite pe parcursul jocului. Ele sunt instanțiate la începutul jocului printr-o apelare a metodei ”Init()”. În interiorul metodei este folosită clasa SpriteSheet ce are ca scop încărcarea unui tileset cu ajutorul clasei ImageLoader și împărțirea tileset-ului respectiv în numeroasele dale ce le conține cu ajutorul celor 2 metode crop. Al 2lea crop are ca argumente 6 variable de tip int. Primele 2 reprezintă poziția dalei în tileset, următoarele 2 reprezintă înălțimea și lungimea dalei, iar ultimele 2 reprezintă spațiul liber din dreapta dalei până la următoarea dală și respectiv spațiul liber dintre dala curentă și cea de sub ea.

Clasa Sprite reprezintă un obiect oarecare de pe ecran. Membri x și y sunt poziția sprite-ului pe harta, ”scale” este un număr cu care se mărește textura atunci când va fi desenată (de ex., dacă scale este 2, textura va fi de 2 ori mai mare) și ”texture” este dala care va fi desenată pe hartă la poziția respectivă. Metodele importante din cadrul clasei sunt cele 2 Draw, care desenează Sprite-ul la poziția sa x, y în raport cu jucătorul sau la o poziție (x,y) specifică dacă aceasta este dată ca parametru. Metoda collision verifică dacă 2 dale se intersectează, metoda collisionEntity face același lucru însă folosește doar jumătate din textura obiectului apelant pentru a verifica,

astfel nu se înregistrează o coliziune dacă doar vârful pălăriei unui personaj atinge un perete superior (lucru care ar arăta nenatural).



Al 2lea pachet numit ”GamePackage” se ocupă de toate mecanicile jocului. Clasa GameMain reprezintă rădăcina jocului. La crearea ei este folosit design pattern-ul singleton pentru a asigura că avem o singură instanța a jocului. Întregul jocul se desfășoară în interiorul metodei ”run()” unde sunt apelate metodele ”Draw()” și ”Update()”.

Clasa GameWindow reprezintă fereastra jocului. Ea conține lungime, înălțimea ferestrei cât și un obiect de tip KeyHandler ce înregistrează apăsările de taste odată ce este adăugat obiectului de tip JFrame ce reprezintă fereastra în sine. De asemenea, obiectul de tip Canvas trebuie adăugat la JFrame și reprezintă câmpul pe care se va desena în fereastră.

Clasa Habitat este colecția de sprite-uri ce formează mediul în care se desfășoară jocul. Obiectul de tip Mapz conține toate informațiile ce țin de ce dale trebuie desenate și unde într-o matrice 3-dimensională numită ”dataz”, prima dimensiune reprezentând camera curentă în care se află jucătorul, informație obținută din membrul ”currentMap”. De asemenea, avem membrul ”doorz” care funcționează în mod similar lui ”dataz” doar că înregistrează doar pozițiile ușilor din joc. Membrul ”isMapState” începe instanțiat cu valoarea ”false”, valoarea acestuia este verificată în metoda ”run()” de mai devreme, dacă valoarea sa este ”false”, se va apela constructorul cu parametri, parametrul int reprezentând nivelul la care se află jucătorul, informație obținută din variabila ”currentLevel” din GameMain. Clasa Habitat apelează metoda ”LoadHabitat()” cu ajutorul metodei ”getCurrentMap” din Mapz pentru a încărca matricea corespunzătoare nivelului curent care o parcurge și își creează propria matrice de obiecte ”Sprite”, folosindu-se de metoda ”getTexture” din clasa Assets pentru a obține diferitele texturi. Metoda ”getTexture” are ca parametri 3 variabile de tip int. Primul int reprezintă matricea din care trebuie luată textura (de ex valoare va fi 1 dacă se vrea o textură din membrul ”caveTextures”), iar celelalte 2 variabile sunt poziția din matrice de unde se extrage textura. În timp ce este construită matricea ”spritez” în Habitat, în funcție de ce texturi sunt întâlnite se va introduce 0 sau 1 în matricea ”collisionMap”. Dacă valoarea este 1, acea textura are coliziune și nu poate fi atinsă de către jucător. Ultimele 2 metode ale clasei Habitat, DrawHabitat și DrawHabitat2nd desenează conținutul matricei ”spritez” și ”doorz” respectiv în raport cu obiectul Player. Clasele Fawn și Spellbook sunt echivalentul lui Habitat doar că pentru inamici și abilități respectiv.

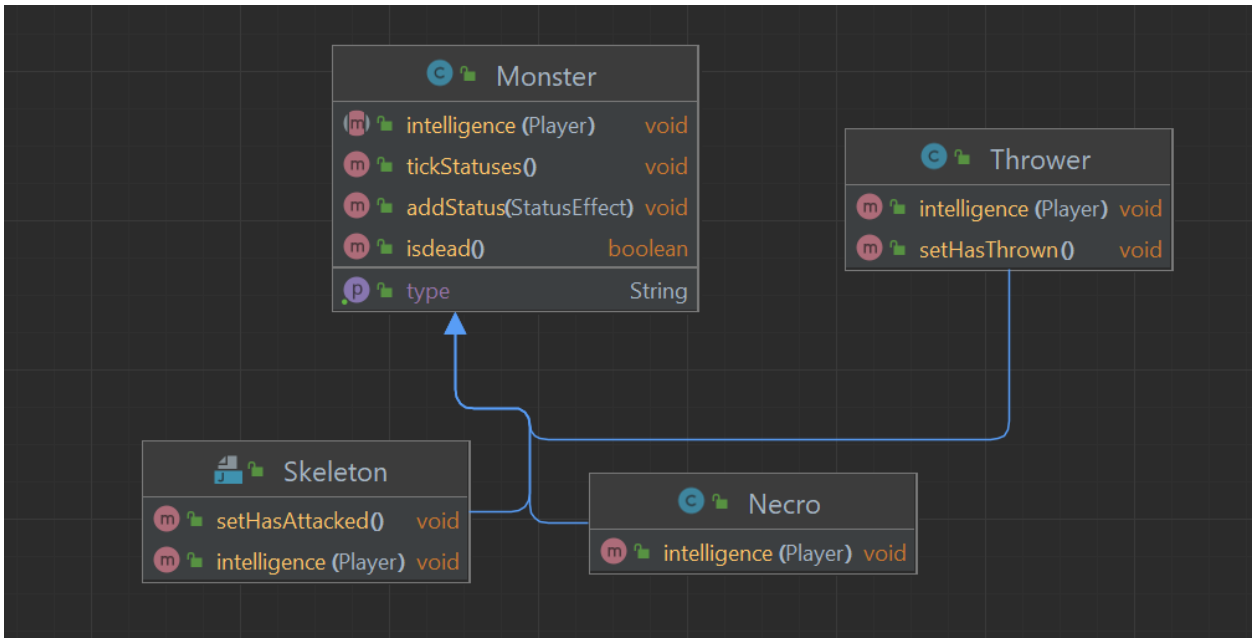
Clasa entity reprezintă baza pentru creea tuturor entităților din joc. Clasa MovEntity, derivată din entity, este la rândul său clasa de bază pentru toate entitățile ce se pot mișca pe ecran. Principalele mecanici implementate de către această clasă sunt, așa cum indică și numele, mișcare de entități pe ecran. Cele 4 obiecte de tip BufferedImage reprezintă diferitele texturi care sunt folosite atunci când o entitate este în mișcare pentru a simula o animație. Variabila ”speed” este viteza cu care entitatea respectivă se mișcă pe ecran, variabila ”buffer” reprezintă numărul de apeluri necesare a metodei ”Move” pentru a trece la următoarea textură din animație de mișcare. Variabila ”facing” este direcția în spre care este poziționat personajul, 0 find înainte, 1 dreapta s.a.m.d. ”CurrentPosture” este poziția la care ne aflăm în ciclul de animație, această variabilă este resetată la 0 atunci când direcția este schimbată sau atunci când ajungem la finalul unui ciclu de animație. În metoda ”collisionCheck” este apelată metoda ”collisionEntity” de catre variabila ”texture” pentru fiecare sprite din jurul entității, această metodă este apelată de mai multe ori în metoda ”Move”.

Clasa MonsterLoader este folosită pentru a interacționa cu baza de date ce conține inamicii din joc. Atunci când jucătorul intră într-o încăpere, sunt citite dintr-o baza de date informațiile

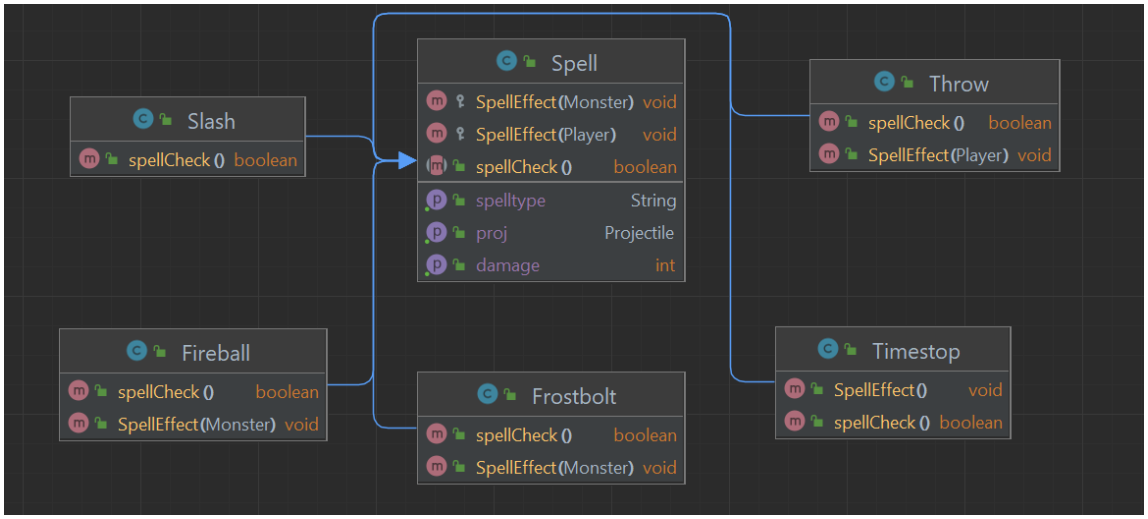
despre inamici care sunt apoi instanțiați în clasa Fawn. Atunci când camera este părăsită, inamicii din Fawn sunt introduși înapoi în baza de date, astfel, atunci când eroul intră din nou în camera respectivă, pozițiile monștrilor va fi pastrată

Ultima clasă este ”Player”. Funcțional, ea nu diferă momentan în niciun mod de la clasa din care se deriva ”MovEntity”. Singura diferență este prezența variabilelor int health și int mana care nu fac nimic în momentul de față.

În acest pachet sunt prezente și alte 3 sub-pachete: Pachetul GUI, care creează atât meniul de început cât și intermisiile dintre nivele și scoreboard-ul de la final, pachetul Monster și pachetul Spells.



Pachetul Monster se ocupă de diferiți monștri din joc, fiecare tip având o proprie clasă, toate extinse din clasa de bază Monster.



Pachetul Spells reprezintă toate abilitățile din joc, atât ale jucătorului cât și ale inamicilor.

Câteva elemente care nu le-am atins sunt unele variabile din clasa GameMain. Obiectul ”dude” de tip Player reprezintă entitatea controlată de jucător. Boolean-ul ”runState” este în permanență ”true”, el devenind ”false” doar atunci când programul este închis. Variabila ”bGround” este simpla textură neagră care este desenată sub toate celelalte texturi.

Sabloane de proiectare:

1. Singleton: acest șablon este folosit în abundență în proiect. Clasele GameMain, Player, Fawn, Habitat, Spellbook, GUI și Intermission sunt toate de tipul singleton. Acest lucru ajută prin faptul că putem accesa aceste clase absolut oriunde în proiect din cauza faptului că metoda ”getInstance” este mereu statică. Un exemplu ar fi metoda ”drawSpells()” din cadrul clasei Spellbook, unde se desenează pe ecran abilitățile curente. Pentru că ecranul este mereu centrat pe jucător, avem nevoie de coordonatele sale pentru a putea desena, așa ca apelam metoda Player.getInstance() pentru a le putea procura.
2. Factory: șablon folosit de 3 ori în proiect prin metodele SpellFactory, MonsterFactory și ProjectileFactory. Care sunt apelate în Spellbook, Fawn și în constructorul clasei Spell respectiv. Cum avem doar 6 tipuri de ”Spell” și 3 tipuri de ”Monster” ne e mult mai ușor să folosim o clasă de tipul factory pentru a instanția diferitele tipuri. Pentru a apela metoda ”build” în oricare din cele 3 este nevoie doar de un argument de tip String ce denominează tipul de ”Spell”, ”Monster” sau ”Projectile” și 2 argumente de tip x și y care reprezintă unde se află obiectul instanțiat pe ecran.

Algoritmi explicați:

1. Coliziunea: coliziunea între diferitele obiecte de pe ecran nu se face cu obiectul în sine, fiecare obiect care se desenează pe ecran find doar componenta ”Sprite” din fiecare lucru desenat pe ecran. Astfel, pentru a verifica coliziunea între oricare 2 obiecte trebuie doar apelată metoda ”collision” din clasa Sprite care calculează un pătrat în jurul celor 2 Sprite-uri folosindu-se de coordonatele lor, dacă cele 2 pătrate se intersectează, metoda ”collision” va returna ”true”.
2. Deplasarea pe ecran: Toate obiectele ce se mișcă apelează metoda ”Move()” din clasa MovEntity. Metoda primește ca parametru 2 variabile de tip int x și y ce reprezintă direcția în care urmează să se miște obiectul ”Sprite”, de exemplu, daca x = -1 și y = 0, mișcarea va fi direct spre stânga. După ce deplasarea este efectuată, se apelează metoda ”collisionCheck()”, ce verifică coliziunea cu toate tile-urile care se afla la 2 tile-uri distanță cât și cu toți inamici, daca metoda returnează ”true”, mișcarea este inversată. În metoda ”Move()” este calculată și animația mersului, la fiecare 20 de iterații find făcută trecerea la următorul cadru de animație sau atunci când direcția de mers este schimbată.

3. Gestionarea inamicilor: toți inamici de pe ecran se află în clasa Fawn în array-ul ”enemies”. Variabila ”monsterCnt” din Fawn reprezintă numărul curent de inamici. În fiecare iterație a programului, Fawn apelează metoda ”intelligence” care trece prin ”enemies” și verifică daca fiecare inamic în parte a ajuns cu viața la 0, daca este cazul, acest inamic este mutat la finalul vectorului, și variabila ”monsterCnt” este decrementată, astfel, acel inamic este înafara programului. Dacă inamicul nu este cu viața sub 0, i se apelează metoda proprie ”intelligence()” care diferă de la inamic la inamic. Pentru tipul ”Skeleton” se obține poziția Player-ului prin o apelare a metodei Player.getInstance(), după care se apelează Move() în așa fel încât obiectul să se apropie de către Player. Atunci când obiectul se află în 60 de pixeli de Player, el apelează metoda ”castSpell” din Spellbook, încercând astfel să atace jucătorul. Inamicul ”Thrower” acționează similar, însă acesta atacă player-ul indiferent de distanță. Inamicul ”Necromancer” se mișcă similar cu cei 2, doar că viteza sa este setată pe 0, astfel el doar se întoarce înspre erou, rămânând pe loc. Acest inamic nici nu apelează ”castSpell”, el apelează metoda ”buildMonster” din Fawn, creând un inamic în fața sa.

Resurse biografice:

- Dive Into Design Patterns de Alexandr Shvets
- Design Patterns de Erich Gamma, Richard Helm, Ralph Johnson și John Vlissides.