

# Programozási Technológia 1.

3. Beadandó 5. Feladat dokumentáció  
Saját játék (Dungeon Crawler)

**Csabai Balázs**  
**E6J9GS**

**2018/1**  
**Gyakorlatvezető: Nagy Sára**

# Feladat

## Kincskereső (Dungeon Crawler)

A jól ismert Legend of Zelda játék egy változatát hozzuk létre. Előre definiált pályákon kell a játékosnak végigmennie (Kazamata/Dungeon) ahol különféle ellenségekkel kell megküzdenie. Ehhez alapvetőleg egy kardot fog tudni használni a játékos, de fel lehet venni még más tárgyakat. A játék célja a pálya fölfedezése és a végén a kincs megszerzése. Ekkor az eltöltött idő, megölt szörnyek, felvett tárgyak alapján kiszámolja a program, hogy mennyi pontot ért el a játékos. Amennyiben elfogy a kalandorunk életerő pontja úgy véget ér a játék és veszett a játékos.

A pályák szobákra vannak osztva amelyek között ajtókon keresztül lehet átmenni. Ezek az ajtók lehetnek kulcsra zártak, vagypedig lezárva addig amíg szörnyek vannak az adott szobában. A játék úgy nehezedik, hogy egyre bonyolultabb, nagyobb pályákon egyre erősebb ellenségekkel kell megküzdenünk.

A játékhoz szükség lesz egy UI-ra ami a jelenlegi állapotunkat mutatja, valamint arra, hogy a karakterek mozogjanak, érzékelje a program ha egymáshoz, vagy egy falhoz érnek. Meg kell oldani, hogy a szoba áthaladásakor a látható pálya átmozduljon az új szoba fölé, valamint egy táska rendszert kell csinálni ahol a tárgyaink közül választhatunk (Egy gombbal a következő tárgyra ugrik), és ha a 'használ' gombot benyomjuk akkor a kiválasztott tárgy használatát kell megírni.

A nehezebb pontja a játéknak az AI elkészítése, mivel többféle ellenség típus lesz a játékban. Olyanok lehetnek mondjuk amik csak 2 fal között mozognak a játékostól függetlenül, míg más ellenségek akár követhetik is a játékosunk. Egyes szörnyek akár lőhetnek is a játékosunkra.

A kazamatában található kulcsok az egyszerűség kedvéért bármilyen zárt ajtót képesek kinyitni, viszont egy kulcs csak egy ajtót tud kinyitni. Ehhez fontos lesz a pályák többszörös leellenőrzése, hogy ne lehessen olyan állapotba jutni, hogy nem tudunk eljutni a pálya végére.

# Megvalósítás Általánosan

## Feladat Pontosítása

A játékosunk tud mozogni, ütni, tárgyakat használni, tárgyakat kiválasztani és leszüneteltetni a játékot. A pálya szobákra van osztva a szobák között ajtókon lehet átmenni.

Az ajtók lehetnek:

- Nyitva
- Zárva: Ekkor kulcs kell a kinyitáshoz
- Beragadva: A szobában található ellenségeket kell ez esetben megölni.
- Fal megrepedve: Egy bomba elhelyezésével fel lehet robbantani.

Minden pályán el kell helyezni egy fő kincset mely a játékos célpontja.

A használható tárgyak:

- Kard: Az alap fegyvere a játékosnak
- Bomba: Sebez és kinyitja a befalazott ajtókat
- Íj és nyíl: Egy távolsági fegyver, használ nyílvesszőket
- Csónak: Le lehet tenni egy víz mezőre és ekkor át lehet jutni a másik oldalára a víznek
- Kalapács: Egy faltípust tűntet el és sebez is.

Az ellenségek:

- Egy helyben állnak
- Föl-le mozognak
- Föl-le mozognak. Ha egy vonalban vannak a játékoskal akkor irányt váltanak
- Ugyanaz mint az előző, csak időnként lőnek ha meglátják a játékost.

A karakterünknek van:

- Életereje (Ha elfogy akkor meghal és veszítünk)
- Tárgyai
- Nyílvesszők
- Bombák
- Kulcsok
- Bónusz kincs: Pluszpontért

A játék végén a pontszámot a következő módon számoljuk ki:

50x a megmaradt idő, az összegyűjtött bónuszkincsek 500-szorosa. A megölt ellenség száma (típusonként változó egy ellenség értéke)

## Feladat Elemzése

A feladatot 4 részre lehet osztani:

1. Főmenü elkészítése
2. Pálya beolvasása/Megépítése
3. Játék futtatása
4. Végeredmény kiírása

## 1. rész elemzése

Az első részhez egy olyan keretre van szükségünk, ahol könnyen lehet választani egy pályát, és az ahhoz lévő pontokat is el lehet érni. Ehhez két részre osztottam a főmenüt. A bal oldalon található a választható pályák névvel és nehézségi szinttel együtt, míg a jobb oldalon a játékot indíthatjuk el, megnézhetjük, hogy a kiválasztott pályán ki mennyi pontot szerzett és még egyéb más gombokat is elhelyeztem. Többek között itt található az Options gomb amellyel az irányítást lehet testre szabni.

## 2. rész elemzése

A pályákat egy-egy .map kiterjesztésű szövegfájlban tároltam melyben meg lehet határozni a kezdő állapotát a játéknak és, hogy a pályán hol és mi található. A ';' -vel kezdődő sorokat a program nem veszi figyelembe így könnyen lehet kommenteket írni a pályánkba is. A pálya nevét és nehézségét a fájl neve határozza meg ilyen formában: {PÁLYA NÉV}.{NEHÉZSÉG}.map. A lehetséges nehézségi fokok easy, normal, hard.

## 3. rész elemzése

A játékhoz futtatásához mindenképp szükségem volt egy rendszerre mellyel könnyen lehet dolgokat megjeleníteni, mozgatni. Ezeket nevezem a programomban Actor-oknak. A játék futtatása közben minden Tick-nél lefutnak az Actor-nak az Update() függvénye majd kirajzolódnak a pályára. Ezeket egy rendezett listában tároljuk amiknek a kulcsa egy-egy egész szám. Mivel a Java-ban nem találtam erre megfelelő adatszerkezetet (van multiplicitása és rendezve van folyamatosan) ezért létrehoztam a saját OrderedList nevű osztályomat mely erre képes és akár még egy felsorolót is lehet belőle készíteni. Ezzel egyszerűen egy for ciklusban meg lehet hívni 'Render' sorrendben az összes element. A Render sorrend nem csak az Update() függvény meghívásában játszik szerepet hanem a megjelenés sorrendjében is. Mivel a játékban több Actor fedheti egymást ezért szükséges tudnunk valamilyen módon, hogy melyik objektumnak kell "főttebb" lennie.

Az irányítást a Controls statikus osztály végzi. Ez figyeli, hogy milyen gombokat nyomott meg a felhasználó és amennyiben megegyezik az egyik gomb a beállított gombbal akkor egy tömbben beállítja annak a gombnak az értékét benyomotttra. Ekkor az Actor-ok le tudják kérni ennek a gombnak az állapotát és ehhez mérten viselkednek.

## 4. rész elemzése

A játék 3 féle képpen érhet véget.

Az első egyszerűbb vége amikor a felhasználó kilép a játékból. Ilyenkor rákérdezünk, hogy biztos, hogy ezt szeretne volna és ha kaptunk megerősítést akkor visszalépünk a főmenübe. A második amikor a játékos veszít (elfogy az életeréje). Ekkor kiírjuk, hogy sajnos nem sikerült nyernie és felajánljuk, hogy meg akar-e próbálkozni még egyszer a pályával vagy lépjen-e vissza a program a főmenübe.

A harmadik amikor a játékos nyer. Ekkor amennyiben elért elegendő pontszámot bekérjük a nevét és beleírjuk a Top listába. Ha nem ért elég pontot akkor hasonlóan amikor veszített a játékos kiírjuk, hogy nem ért el elegendő pontot és megkérdezzük, hogy akar-e még egyszer próbálkozni.

## Terv és Osztálydiagram

A játék legfőbb őssztály az Actor. Ez a constructor-ában belerakja saját magát az actors Ordered List-be. Ezáltal a Game objektum Tick-enként meghívja az Update függvényét majd kirajzolja a pályára. A játékot a Board osztály futtatja mely egy JPanel objektum. Erre rajzoljuk ki az összes Actor-t. A játék logikája az Update függvényekben zajlik.

## Osztályok

A **MainMenu** osztály feladata a főmenü megjelenítése és működtetése. Itt jelenítjük meg a különböző tábla méreteket, valamint itt lehet elérni a játék leírását és információt a program eredetéről. Az osztály egy singleton és a JFrame osztályból ered.

Változók:

- `main` : MainMenu: A singleton objektum elérésére szolgáló statikus referencia.
- `levels` : Vector<Level>: A
- `icon` : Image: A

Metódusok:

- `MainMenu()`: Konstruktor. A GUI elementeket hozza létre, valamint beállítja az ablak méretét, pozícióját és ikonját.
- `getLevels()`: Beolvassa a pályákat és eltárolja a levels vektorban.
- `showDialog()`: Megjelenít egy dialógust testreszabható címmel, szöveggel és gombokkal.

**Board** osztály a játékot megjelenítő panel. Ez futtatja a játék tick-eit, játsza le a hangeffekteket és jeleníti meg a jelenlegi állapotot.

Változók:

- `level` : Level: Referencia a jelenlegi pályára.
- `game` : Game: Referencia a Game Singleton osztályra. Innen szerzi meg a játék jelenlegi állapotát
- `scale` : float: A játék méretét szabályozza
- `music` : Clip: A játék háttérzenéje.

Metódusok:

- `ExitGame()`: Kilép a főmenübe.
- `RestartGame()`: A kezdőállapotra állítja a játékot
- `EndGame()`: Ha vége a játéknak akkor a fentebb leírt módon viselkedik a függvény
- `paintComponent()`: A játék megrajzolása

**Game** osztály feladata a játék fő logikájának futtatása. A Colliderek leellenőrzése, az Actor-ok Tick()-jének meghívása, a játék végének leellenőrzése.

Változók:

- `game` : Game: A singleton objektum elérésére szolgáló statikus referencia.

- level : Level: Referencia a jelenlegi pályára.
- EnemyCountInRoom : Int: Ellenségek száma a szobában (A beragadt ajtókhöz).
- isPaused : Boolean: Le van-e szünetelve a játék.
- score : Int: Jelenlegi pontszám
- boardOffset : Point: Mennyivel van elcsúsztatva az összes actor.
- actors: OrderedList<Actor>: A játékban lévő actor-ok láncolt listája.

Metódusok:

- Tick(): Meghívja az Actor-ok Tick-jét, leellenőrzi a Collidereket
- PlayClip(): Beírja egy listába a paraméterként megkapott hangklippet, hogy a board lejátssza a következő tick-en
- MoveToRoom(): Mozgatja a boardOffset-et majd meghívja az actorokra az OnRoomChange függvényt
- FindActorByName(): Megkeres egy actor-t amelynek a neve megegyezik
- GetID(): Ad egy egyedi azonosítót
- Win(): Jelzi a Board-nak, hogy nyert a játékosunk
- Lose(): Jelzi a Board-nak, hogy veszített a játékosunk
- HasCollision(): Leellenőrzi, hogy 2 collider má ütközött-e egymással régebben, vagy sem

**Actor** a legtöbb osztály ősosztálya.

Változók:

- id: Int: Egyedi azonosítója az Actor-nak (A Game GetID-ből kapja).  
Összehasonlításnál elegendő így csak ennek az értékét megnézni
- name: String: Az actor-nak egy név azonosítója, nem feltétlenül egyedi
- Size, Position: Point: Actor mérete és pozíciója (a pozíció relatív a szülőjétől)
- isActive, isHidden, isFlipX, isFlipY, staticPosition: boolean: Aktív?, Látható, Tükrözve van-e?
- parent: Actor: A szülő objektuma
- animator: Animator: Animátor. Az animációk lejátzásában segít
- collider: Collider: Ütköző. Két objektum egymáshoz érésére tud reagálni
- sprite: Pair<String, String>: A megjelenítendő képkocka
- tag: Tag: Az Actor-ok csoportosítása, enumerátor
- components: Vector<MyComponent>: Plusz komponensek, hogy el lehessen külön részekre osztani a kódot.

Metódusok:

- Tick(): Meghívja az Update metódust, az animátorát, megnézi, hogy meg kell-e semmisítenie önmagát (akkor ha aktív az actor)
- Update(): Fölülírható metódus, az osztály gyerekeinek. Lefut minden Tick után.
- Getterek, Setterek, OnShow, OnHide, OnEnabled, OnDisabled, OnRoomChange()

**Animator** a hozzáfűzött actor-nak a sprite-ját állítja be tick-enként.

Változók:

- frame: Pair<String, String>: Ugyanaz mint a Sprite, csak az animator-ban van eltárolva
- booleans, integers, strings: TreeMap<String, Type>: A kommunikációt hozza létre az animator és más objektumok között

- active: boolean: Aktív?

Metódusok:

- Init() : Felülírható metódus mely az animator létrejöttkor fut le.
- Animate() : Felülírandó metódus, ide lehet írni kódot ami kiszámolja a megjelenítendő frame-et.
- Getterek, Setterek

**Collider** a hozzáfűzött actor-nak a más actorokhoz való ütközését lekezelő osztály

Változók:

- isTrigger : Boolean: Át lehet menni az objektumon és csak érzékelje, vagy ne.
- active : Boolean: Aktív?
- bounds : Rectangle: A collider szélei
- distance : Int: Maximum távolság aminek ütközés után még "egymáshoz érnek" számít a két collider.
- ignoreTags : Vector<Tag>: Tag-ek amiket nem vesz figyelembe. (át is lehet menni ilyenkor)

Metódusok:

- Init() : Felülírható metódus mely az animator létrejöttkor fut le.
- OnCollisionBegin, OnCollisionStay, OnCollisionEnd : Felülírható metódusok az ütközés eseményeire.
- Getterek, Setterek

**Item** egy absztrakt osztály amely a felvehető tárgyakat örökölteti

Változók:

- sprite : String: A UI-on megjelenő ikon, és egyedi neve a tárgynak.
- player: Player: Referencia a Player-re

Metódusok:

- CanUse() : Felülírandó függvény. Megadja, hogy tudjuk-e a jelenlegi állapotban használni a tárgyat.
- Use() : Felülírandó metódus, használja a tárgyat

**Level** a pályák beolvasában és megépítésében van szerepe.

Változók:

- name : String: A UI-on megjelenő ikon, és egyedi neve a tárgynak.
- difficulty : Difficulty: Referencia a Player-re
- fileName : String: Referencia a Player-re
- hp, maxHP, bombs, maxBombs, arrows, maxArrows, keys, time, map[[[]], equipment[]: A pálya adatai

Metódusok:

- Loadlevel() : Fájlból beolvassa a pályát.
- BuildLevel() : Létrehozza a pályát

**Enemy** egy Actor-ból származtatott absztrakt osztály amely az ellenségeknek

Változók:

- health, maxHealth : Int: Az ellenségek életerejé és maximális élete
- canRespawn : Boolean: Újra tud-e éledni az ellenség ha újra belépünk a szobába
- canBePushed: Boolean: El lökődik-e az ellenség ha megütjük valamivel

- `immune : Vector<Immunity>`: Milyen dolgokra immunis az ellenség (Az `immunity` egy enumerátor, elemei `Sword`, `Bow`, `Bomb`, stb.)
- `killScore : Boolean`: Újra tud-e éledni az ellenség ha újra belépünk a szobába
- `dropItems : Vector<PickupItems>`: Milyen `killCount` értékre milyen tárgyat dobjon el az ellenség miután meghalt
- `dmg : Int`: Mennyit sebez a játékosba.
- `killCount: int`: Ez a szám alapján fog az ellenség valamilyen tárgyat eldobni (nő 1-el minden megölt ellenség után)

Metódusok:

- `Move()` : Felülírandó metódus mely akkor hívódik meg ha tud mozogni az Ellenség
- `TakeDamage()` : Ha sebződik az ellenség akkor vonja le az életét, és ha lehet akkor lökje el valamilyen irányba
- `OnRoomChange()` : Ha abba a szobába mozog a játékos ahol az ellenség van akkor aktiválja és ha már meghalt de újra tud éledni akkor újraéleszti.

**Player** osztály a karaktert jelzi amelyet a játékosunk irányíthat

Változók:

- `arrowCount, bombCount, health, keyCount, maxArrowCount, maxBombCount, maxHealth, speed, swordDmg, treasureCount: Int`: A játékos táskája/állapota
- `items: Vector<Item>`: A tárgyak amiket a játékos birtokol.

Metódusok:

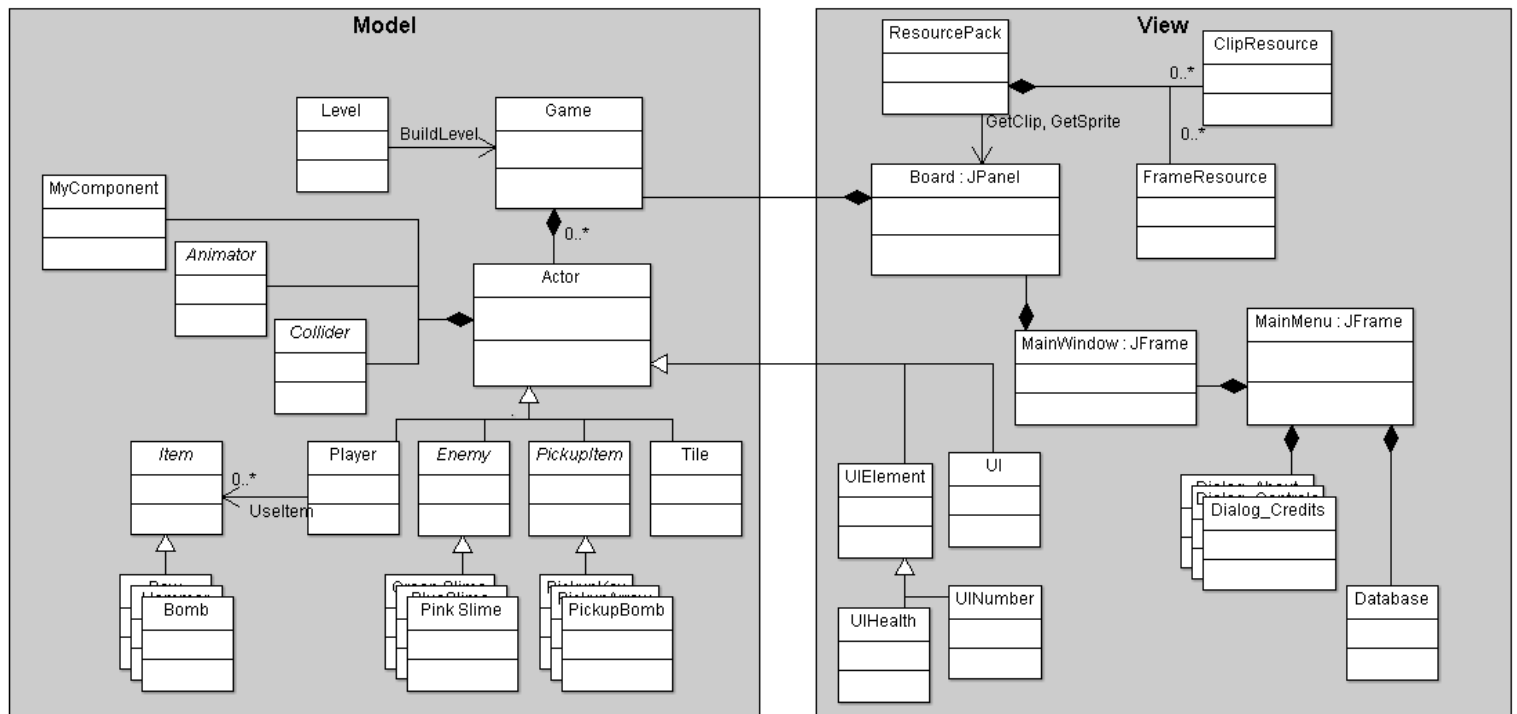
- `CanDoAction()` : Visszaadja, hogy tud-e a játékos mozogni
- `HasItem()` : Van-e a játékosnak egy megadott tárgya
- `RoomChange()` : Ha másik szobába megyünk akkor ez jelzi a Game-nek, hogy szobát kell váltani
- `TakeDamage()` : Sebzi a játékost
- `Die()` : Ha nincs több élete a játékosnak akkor meghívja a `Lose()` függvényt és veszít a játékos

**UI** Létrehozza a képernyőnek a statikus részeit melyeket a játékos folyamatosan lát ugyanott (mennyi élete van, stb.)

**PickupItem** egy olyan Actor amihez ha a játékos hozzáér akkor fölveszi a hozzá tartozó Item-et (Absztrakt osztály)

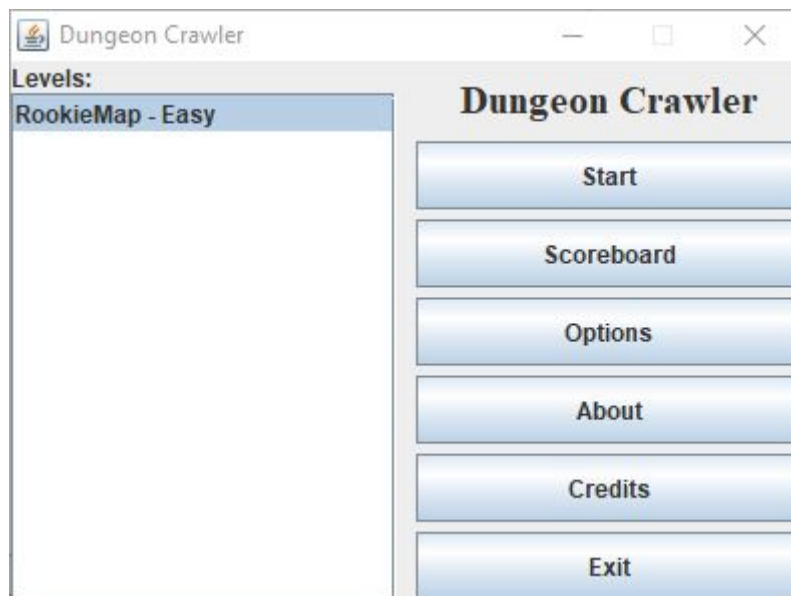


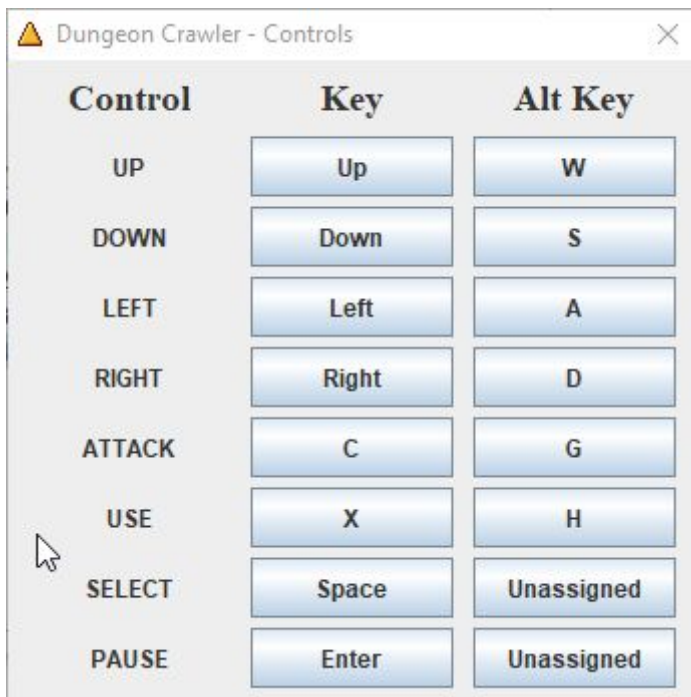
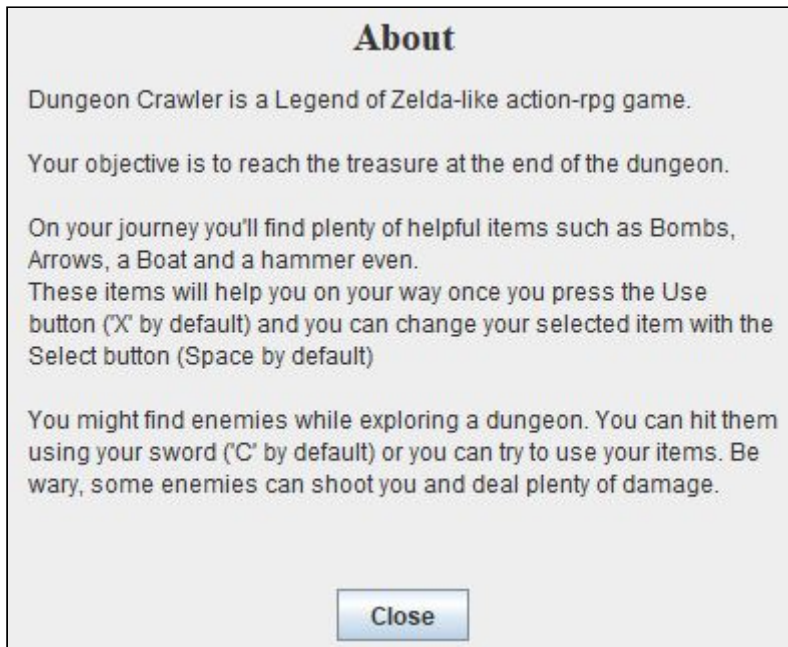
## Osztály Diagram:



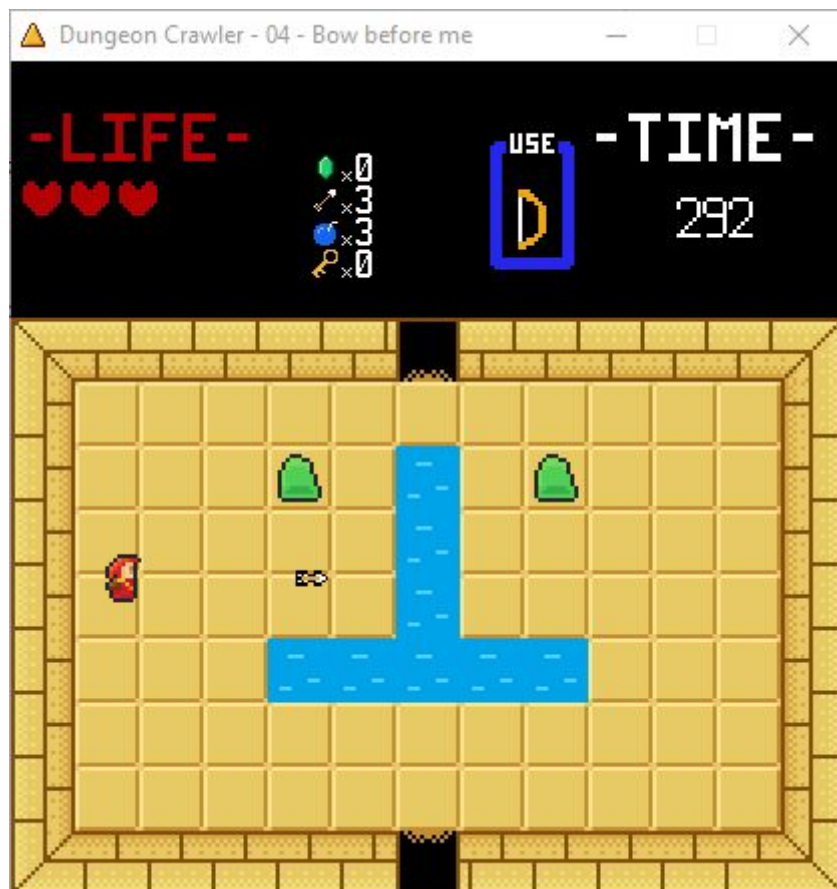
## Megvalósítás

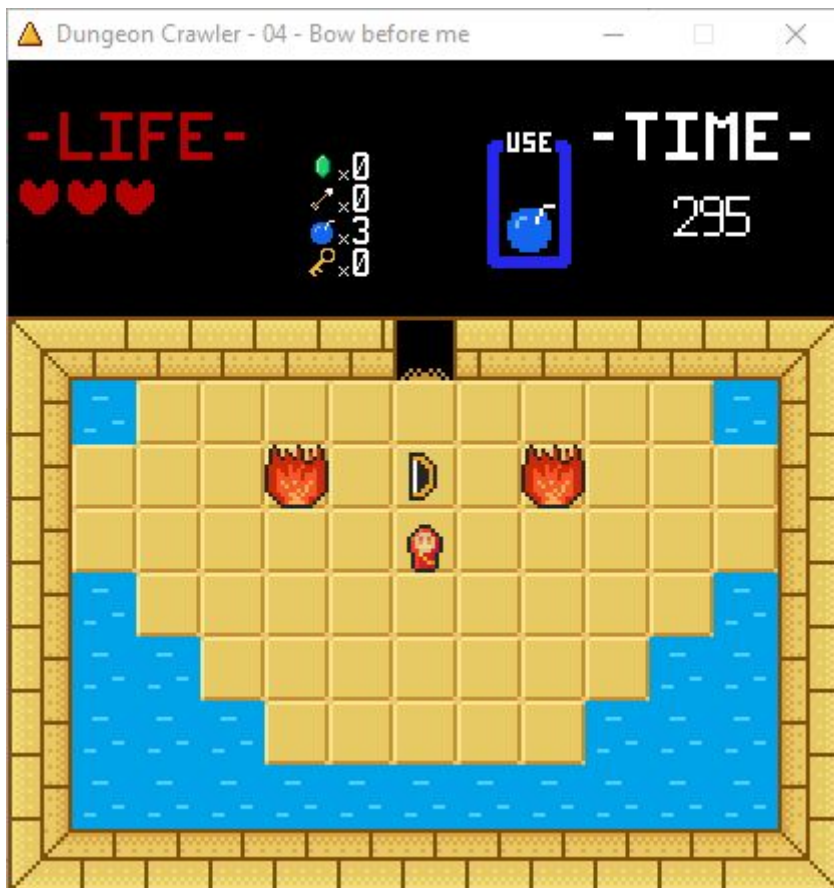
### 1. rész

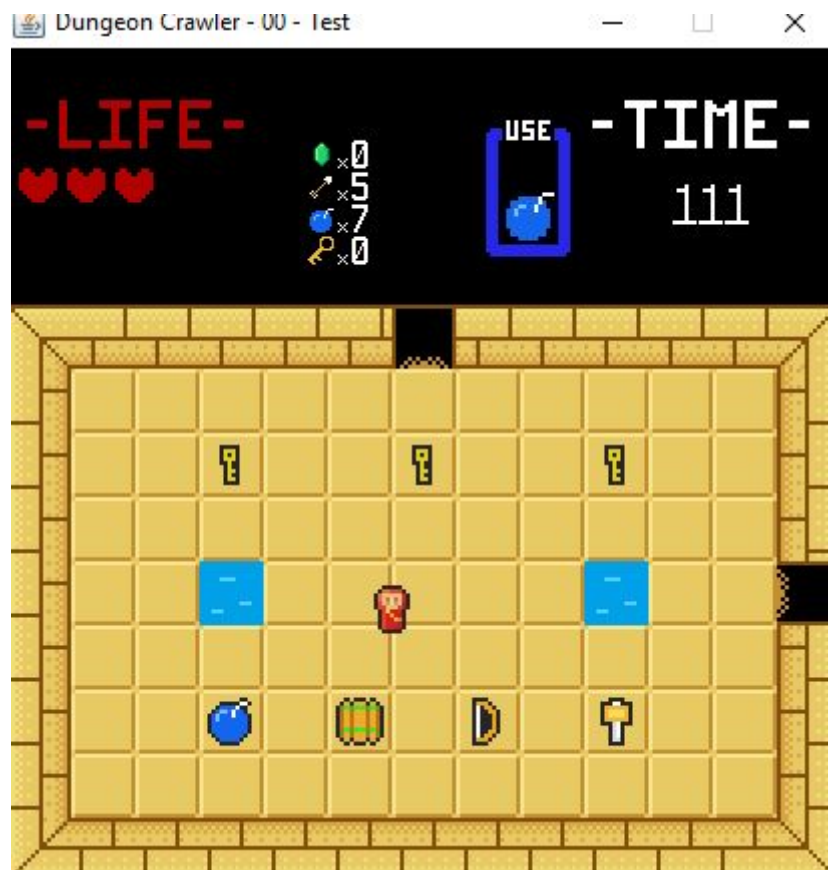
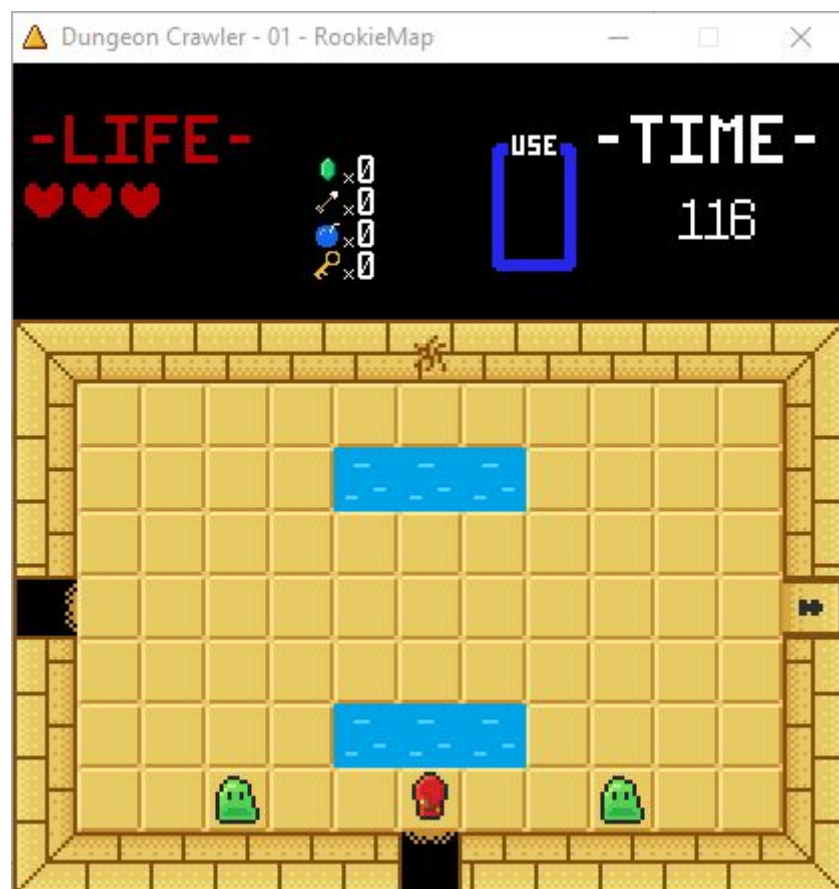





## 2. és 3. rész








## 4. rész

 New High Score! ×

**Congratulations! You've reached a new High Score (13550)**

Player Name:

**Send**

 Dungeon Crawler - 04 - Bow before me ×

Name	Score
Balázs	13550
Peti	10250
Gábor	10000
Peti	8550
Feri	7000