**MayhemShot**

**Dokumentáció**

**Magyarosi Péter**

**XII.G – Matematika-informatika osztály**

**Felkészítő tanár: Jakab Irma-Tünde**

**2021-es tanév**

1. *Témaválasztás*

Videójátékok. Bizonyára mindannyiunk életében ott vannak valami féle módon, nagy befolyásuk a számítógép rohamos fejlődésével érkezett. Ugye, tizenéves fiatalként, aki már kis kora óta videójátékokkal van körülvéve, engem is foglalkoztat a téma. Mióta programozni tanulok, máshogy tekintek minden programra. Játék közben gondolkodom, hogyan is lehet egy-egy jelenlevő mechanikát implementálni, hogy mennyi időbe telhet ennek a megírása, valamint összetettebb dolgoknál elcsodálkzom, hogyan tudták kitalálni ezeket a bonyolult algoritmusokat. Pontosan ezért, már én is néhány éve elkezdtem foglalkozni játékok fejlesztésével, kezdve Space Invaders, majd Legend of Zelda klónokkal. Ma is egy ilyen játékot szeretnék bemutatni, amely néhány kedvenc indie shooter játékomról lett inspirálódva. Neve a **MayhemShot**.

1. *Rendszer követelmények*

Játékomnak nincsenek különösebb rendszeri/szoftveri elvárásai, elfut bármely átlagon aluli gépen is, viszont mivel 64 bites rendszeren és fejlesztői környezetben volt lekomplilálva a végső kód, ezért 32 bites operációs rendszeren nem fog működni.

**Minimális hardver/szoftver igény**

**Processzor:** 500 MHz **Memória:** 256 Mb **Videókártya:** DirectX 9 kompatibilis **Tárhely:** 50 MB üresen **Operációs rendszer:** Windows 7  
**Billentyűzet  
Egér**

**Ajánlott hardver/szoftver igény**

**Processzor:** 1GHz **Memória:** 512 Mb **Videókártya:** DirectX 9 kompatibilis **Tárhely:** 50 MB üresen **Operációs rendszer:** Windows 10 **Billentyűzet  
Egér**

1. *Felhasználói kézikönyv*

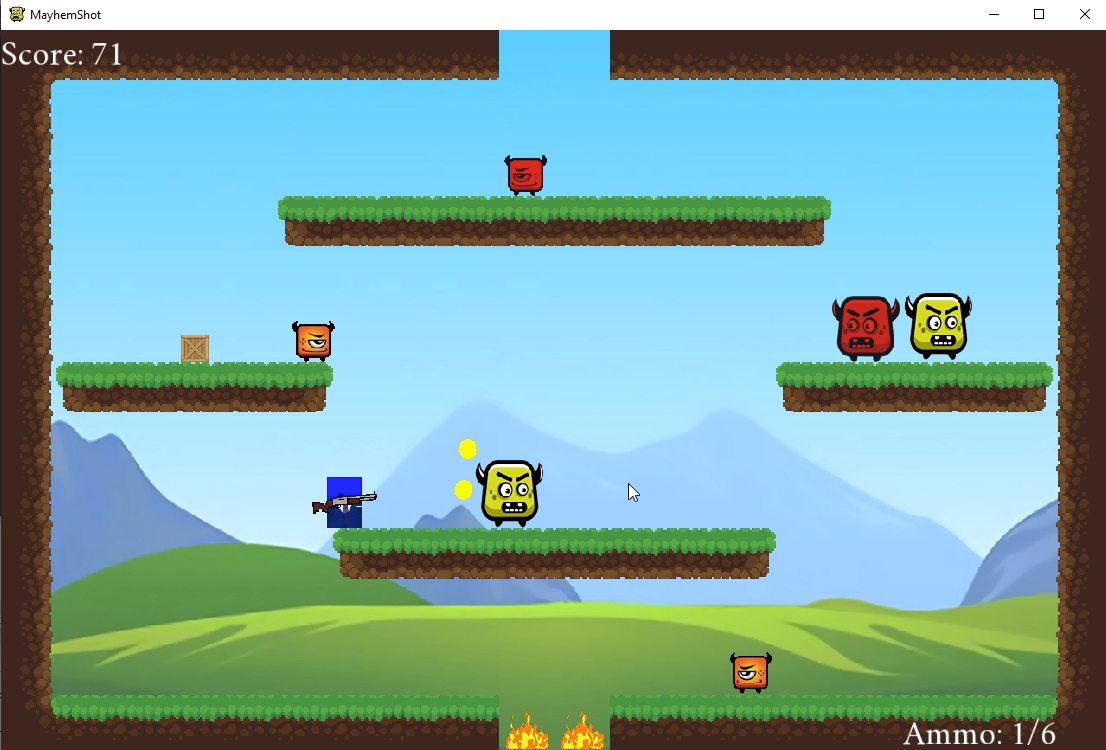
A játék egyszerű, könnyen átlátható felhasználói felülettel rendelkezik. A menű és egyéb helyeken található kevés szöveg angol nyelven írodott, hogy könnyebben lehessen szétosztani az interneten. Indításkor a főmenü jön elő, ahonnan választani tudunk a következő menüpontok közül:



* 1. **Singleplayer:** A program lényeges része itt található. Ha a felhasználó ide kattint, a fő-, egyjátékos mód felülete töltődik be.
  2. **Multiplayer:** A jövőben érkező újabb játékmód, jelenleg inaktív a gomb, viszont nemsokára bekerül ez is a teljes verzióba!
  3. **Options:** Beállítások menü, jelenleg innen tudjuk állítani a játék hangerejét, láthatjuk az adott kliensen szerzett legnagyobb eredményt, valamint kapunk egy rövid kis leírást a program készítőjéről is.
  4. **Quit:** Kilépés gomb, amikor rákattintunk bezáródik az ablakunk.

**A játéknak egyszerű menete van**: a képernyő közepére pottyant öltönyös karaktert irányítjuk, aki kap kezdetben egy kis pisztolyt, korlátozott lőszerrel. Célunk minél több pontot szerezni, amelyeket a képernyő tetejéről érkező ellenségek megsemmisítéséből kapunk. Lőni a bal egérgombbal tudunk, karakterünk mindig a kurzor fele fordul. Az ellenségek oldalra fognak haladni, és megfordulnak, amint fallal ütköznek. A képernyő alján egy tüzes lyuk található, amelybe ha beesnek, visszatérnek, csak éppen mérges formában, amely annyit jelent, hogy sokkal nagyobb sebességük van. Ezen kívűl kétféle ellenség típus létezik: egy kisebb méretű, amelynek kevesebb-, és egy nagyobb típus, amelynek több életpontja van**.** A pályán mindig található egy doboz, amelyen áthaladva a karakterünk új fegyvert kap. Ezután egy újabb doboz jelenik meg véletlenszerűen egy másik szilárd felületen, amelyet ugyancsak fel tudunk venni. A legjobb teljesítmény érdekében mindig legyünk doboz-közelben amikor éppen fogyunk ki a lőszerből, hogy utána egyből tudjunk újratölteni, és tudjuk folytatni a játékot!  
A bal felső sarokban láthatjuk a jelenlegi pontszámunkat, valamint a bal alsó sarokban a pillanatyni lőszerünk figyelhető meg

* + 1. A játék kétféle képpen érhet véget: hogyha ellenséggel ütközünk, vagy ha beleesünk a pálya alján található tűzbe. Ilyenkor egy másik menüpont jön fel, ahol kiírja a szerzett eredményünket, és 2 opciónk van: újrapróbálkozni, vagy visszatérni a főmenübe.
    2. A játék során bármikor megnyomva az Escape gombot, felhozhatjuk a szünet menüt, ahonnan újból elérhetjük a beállításokat, valamint visszaléphetünk a főmenűbe is.



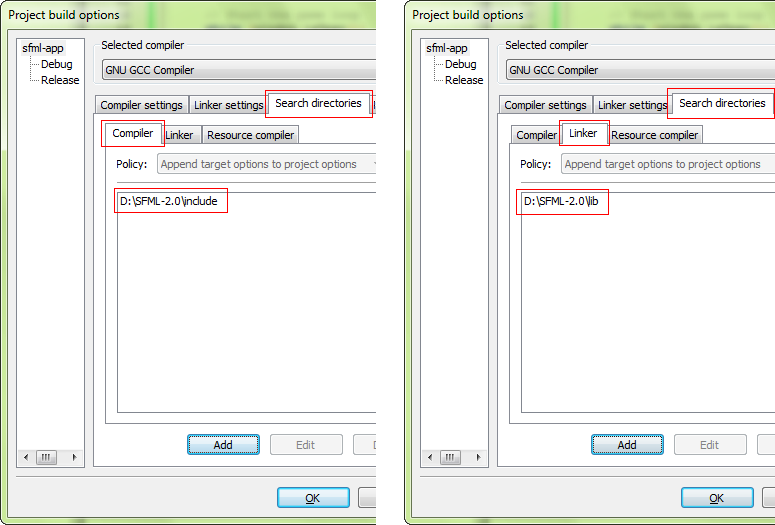
*Pillanatkép az egyjátékos módból*

1. *Programozói kézikönyv*

Programom C++ nyelvben írodott és a grafikus részéhez az SFML könyvtárat használtam. Azért választottam az utóbbit, mivel egyszerű felülettel rendelkezik, valamint azért, mert egy könyvtár használata esetében sokkal jobb belátást nyerek alkalmazásaim működésébe, mintha egy motrot, vagy egy összetettebb keretrendszert használnék. Ez esetben jobban meg tudom tanulni, hogy mi is folyik mindennek a hátterében. Fontosnak tartom hogy ezeket megtanuljam, mielőtt komplikáltabb fejlesztői környezetekhez lépek, mint például a Unity.

Setup

A megkapott kódot, amelynek a legfrissebb verziója a GitHub oldalamon is fent lesz (<https://github.com/magyarosipeter>), mielőtt futtatni tudjuk saját számítógépünkön, szükséges a projekt beállítások között újrakonfigurálni az SFML útvonalat, hiszen nem ugyanoda lesz a könyvtár feltelepítve, mint az eredeti kódot futtató gépen. A teljes folyamatot megtekinthetjük a „<https://www.sfml-dev.org/tutorials/2.5/start-cb.php>” oldalon, azonban nekünk lényegében csak a „Search directories” menüpont alatt kell kicserélnünk a “Compiler” és “Linker”-ben található útvonalat az általunk telepített SFML mappára.



A többi beállítás maradhat úgy, ahogyan a projektben van.

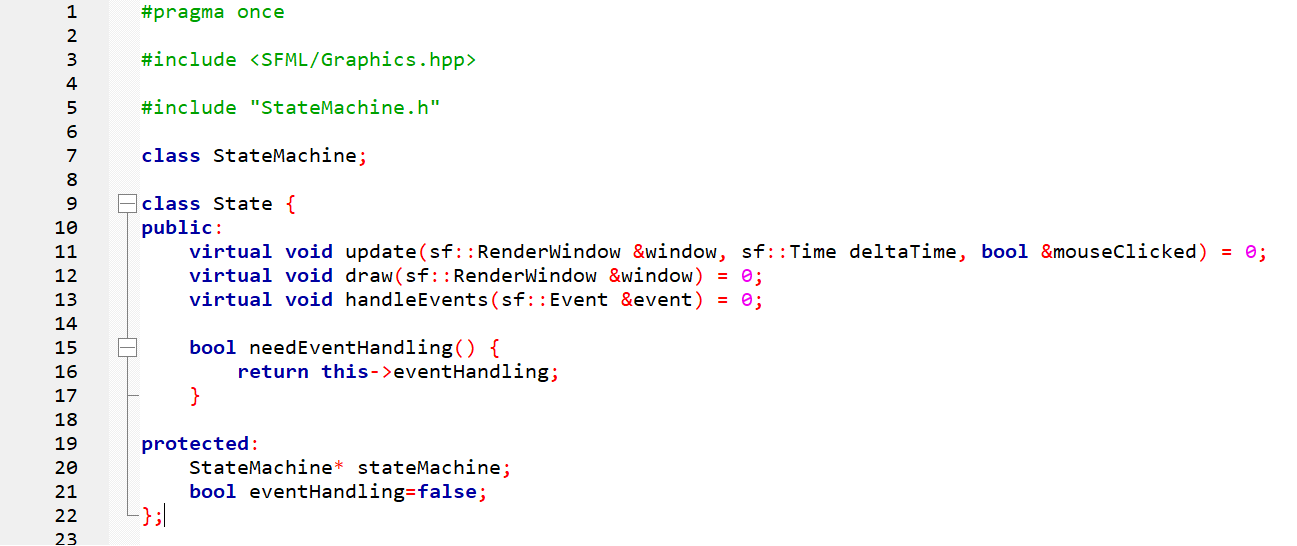
**FONTOS!** Ha jelenlegi (2021.V.10.) 2.5.1-es SFML verziónál újabbat töltünk le, és azt konfiguráljuk a projekt beállításaiban, akkor a .exe fájl (bin\debug\) mellett levő .dll fájlokat is frissíteni kell a legújabb csomagból kapott .dll fájlokkal, különben a programunk határozatlan futtatási errort ad!

Kódrészletek

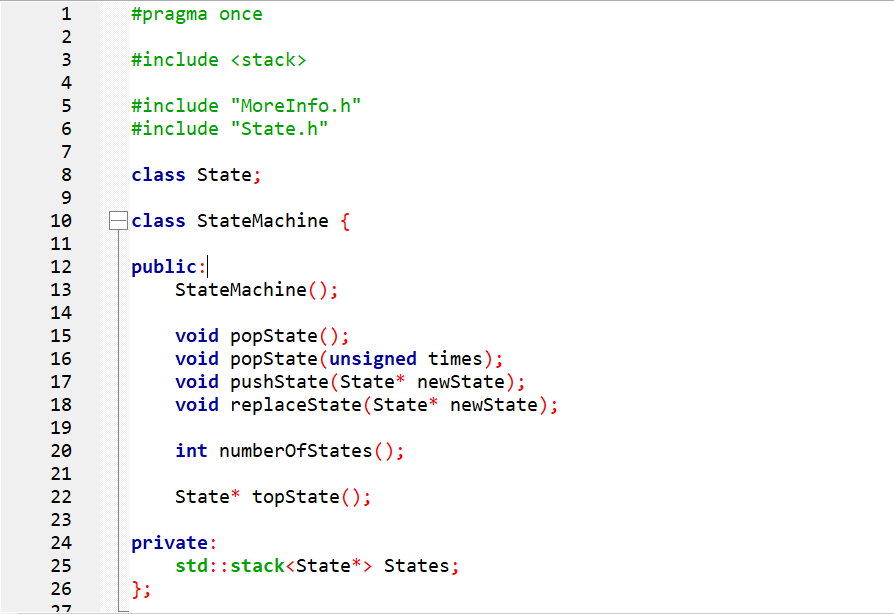
A programom kódolása közben nagy hangsúlyt fektettem az **objektum-orientáltság**ra. Ennek segítségével a felépítésem modulárisabb, könnyebben átlátható, valamint nagyobb mértékű az újrahasznosítható kódrészletek száma. Ez a szerkezet a legjobban a Game és StateMachine osztályaimnál figyelhető meg.

StateMachine

A játékom alapját egy úgynevezett **StateMachine** teszi ki, amely azt jelzi, hogy éppen melyik részében vagyunk a játéknak (pl. főmenü, beállítások menü, egyjátékos mód, stb…). Ennek verem tipusú felépítése van. Lehetőve teszi, hogy amikor például a szünet menüt megnyitjuk játék közben, ne vesszen el az éppeni játékhelyzet. Ezen kívűl egyszerűvé teszi a Game osztályom működését. Nézzük meg kódban a működését.

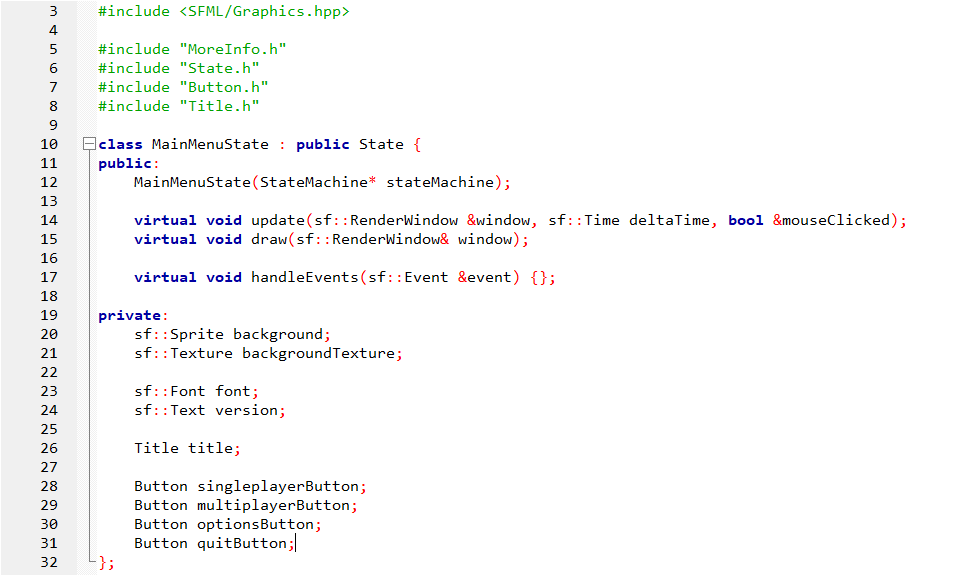


*Az* ***absztrakt*** *State osztályomnak nincsen konstruktora, nem lehet példányosítani, csupán* ***inheritancia*** *miatt van létrehozva. A State osztály mondja meg, hogyan kell felépüljön minden másik, tőle öröklődött State.*



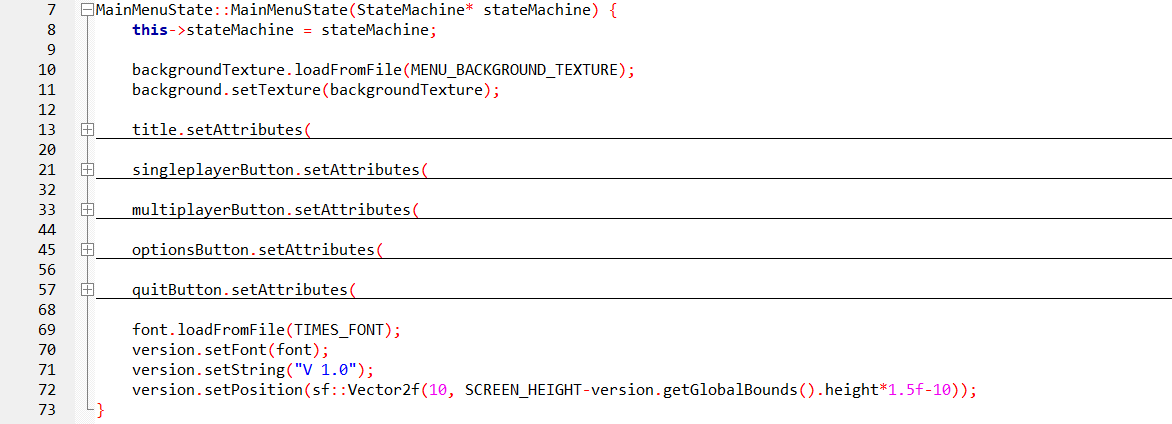
*A StateMachine osztályom. Fő alkotóeleme a privát részben létrehozott* ***verem****, amely különböző State típusokra mutató pointereket tárol. A publikus függvények csupán meghosszabbítják az std::stack beépített függvényeit biztonsági ellenőrzésekkel (pl. ne tudjuk pop-olni a legfelső elemet, ha üres a veremünk, valamint fontos tennivaló, hogy mielőtt pop-olnánk és elveszítenénk a legfelső elem memóriacímét, delete-eljük azt, különben memóriaszivárgásunk lesz)*

A stackemet továbbá **polimorfizmus** segítségével töltöm fel. Azért építettem fel így ezt a rendszert, mert ahogyan más nyelvek is, a C++ is megengedi, hogy a State osztályból származtatott osztályokat ugyancsak State pointer azonosítoba tudjam eltárolni. Nézzük meg hogyan épül fel egy ilyen származtatott osztály.

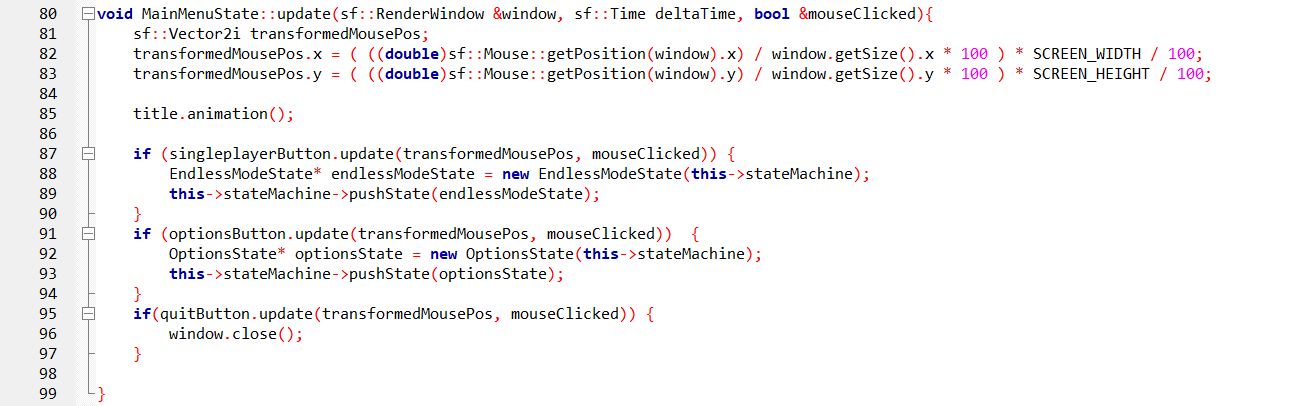


*Az előbbi képen megfigyelhető a főmenü felépítése. Ahogyan látható, ugyanazok a függvény fejlécek vannak a publikus elemek között, mint amik az anyaosztályban is vannak, a State-ben. Ezeket ebben az osztályban fogjuk definiálni. A privát elemek közé tartoznak az adott Statehez jellegzetes elemek, ebben az esetben a cím, a gombok és a háttér.*

Fontos tulajdonsága a **virtuális függvények**nek, hogy muszáj definiálni őket a származtatott osztályban. A MainMenuState-ben mivel nekem nincs szükségem event managementre (az sf::Event olyan dolgokat foglal magába, mint például szöveg gépelése), ezért üres függvényként definiálom. Ezt muszáj megtenni, különben nem fog lefutni a programunk (kompilálási hiba).

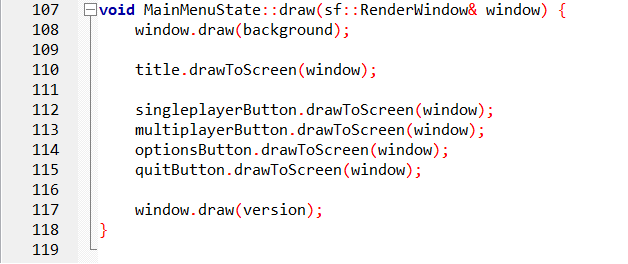


*Az osztály konstruktorának lényege, hogy inicializáljuk az osztály elemeit. Itt állítjuk be például a háttér textúráját, itt helyezzük el a gombokat, adunk nekik szöveget, színt, stb...*



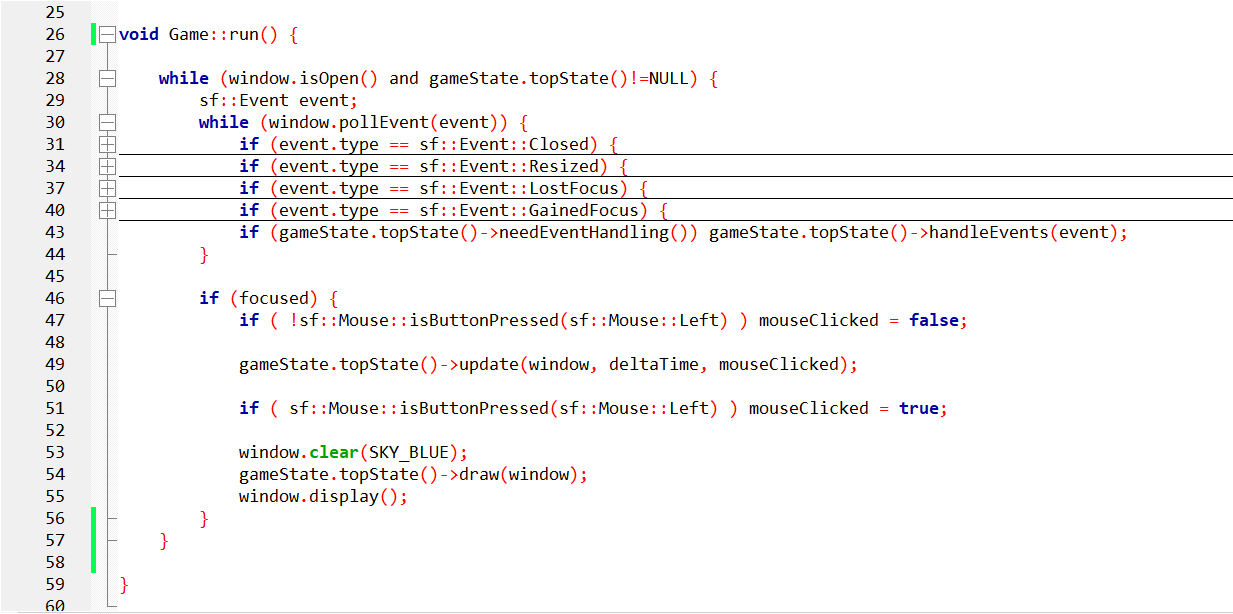
*Itt látható az update() függvény definíciója. Ez a függvény lesz meghívva a fő ciklusban minden iterációban és itt dolgozzuk fel a felhasználó bemeneteit. Ebben az esetben a gombokat ellenőrizzük és ha rákattintunk valamelyikre, új State-t tölt be a StateMachine-be.*

Az előbbiben megyfigyelhetjük a polimorfizmus tulajdánságait. Ugyan a StateMachine-ünk State\* típusú változókat tárol el a veremben és függvényeinek paraméterei is ilyent kérnek, mégis képesek vagyunk más, a State-ből inheritált osztályok példányait elhelyezni benne (EndlessModeState, OptionsState).



*A draw() függvényben egyszerűen kirajzoljuk az adott state elemeit a képernyőre.*

Ettől fogva a Game osztályunknak egyszerű felépítése van. Bármely legyen az aktuális State, a két sajátos függvényt fogjuk meghívni rá: az update() és draw(), abban az esetben, ha még eventekkel is foglalkozik a State, meghívjuk a handleEvents()-et is.

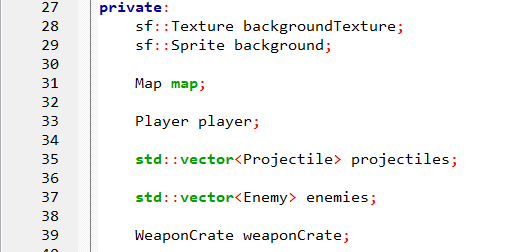


*A program addig fut, ameddig nyitva marad az ablak, vagy ameddig elfogynak a Statek.*

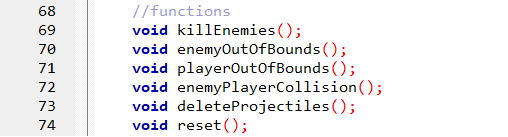
*A “focused” változó a Game osztály mezője, és az eventekkel kapcsolatos tulajdonság. Azt nézi, hogy lekattintottunk-e az ablakról, vagy sem. Előbbi esetben a program futtatása nem folytatódik, hanem ideigelenesen megáll.*

Adatok letárolása a játékban

A játékban szerepet játszó elemek, mint a játékosunk, az ellenségek, a lövedékek, a pálya mind osztályok formájában van létrehozva. Az ellenségeket és lövedékeket std::vector segítségével tárolom el, mivel ez ad lehetőséget arra, hogy szükség szerint tudjunk törölni közülük, vagy új elemet tudjunk hozzájuk adni.

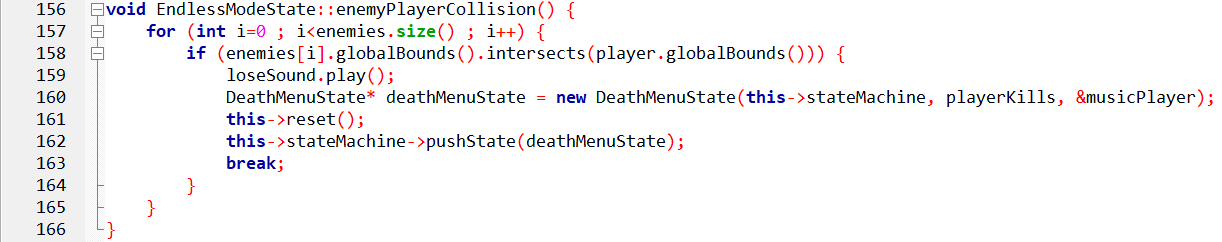


*Néhány mező az EndlessModeState-ben, azaz a fő játékmódunkban*



*Függvények az EndlessModeState-ben, amelyek a fenti mezők között biztosítják a kapcsolatot*

Ezekből nézzük meg például az ellenségek és a játékos közötti ütkozést:

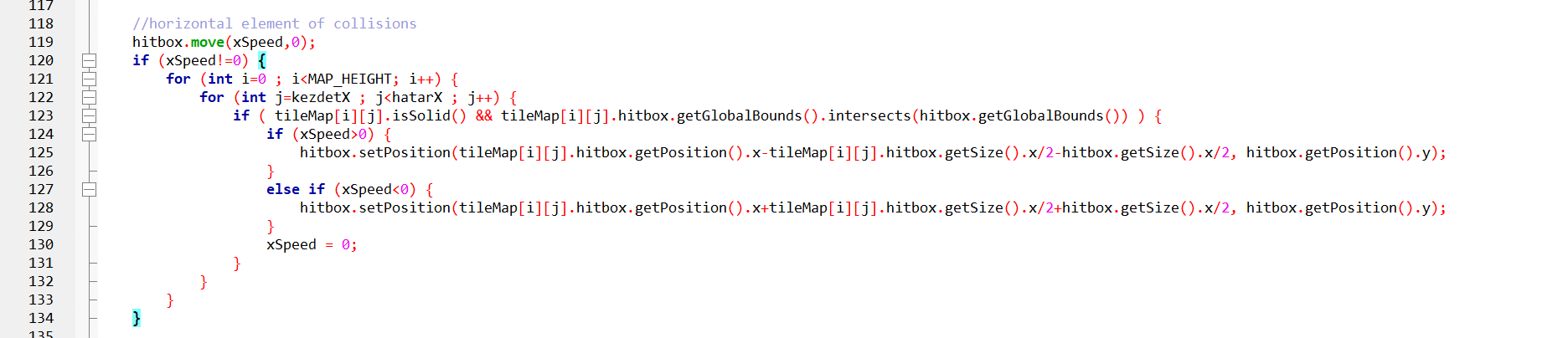


*A fenti függvény ellenőrzi le a játéknak azt a részét, hogy ütközik-e a karakterünk ellenséggel. Ilyenkor a teendő betölteni egy másik Statet, a Game Over menüt.*

Környezettel való ütközések

A 2D platformer játékok fő mechanikája a karakterünk folyékony és kiszámítható mozgása. Ehhez implementálni kellett a környezettel való ütközéseket, amelyeket a következő módon oldottam meg:

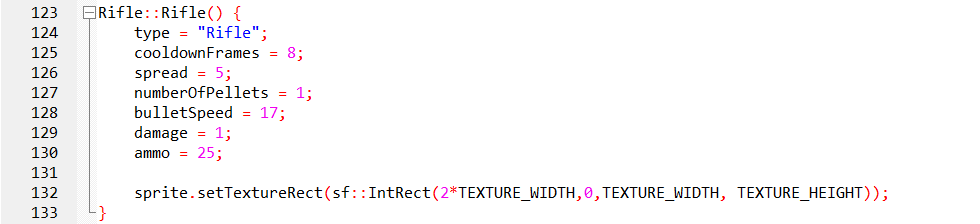
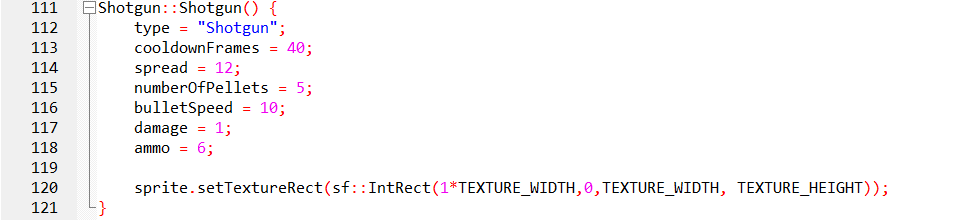
* feldolgozom a felhasználó bemenetét, és annak megfelelően beállítom a karakterünknek a sebességet (ha az A vagy D gombot tartjuk lenyomva, a karakterünk vízszintes sebességét állítjuk át, ha a Space gombot nyomjuk le és a földön található a karakter, függőleges sebességet adunk neki).
* a pálya elemekkel való ütközést 2 részre bontottam fel, x és y – vízszintes és függőleges irányokra, mivel másképp határozatlan esetek alakulhatnak ki (Mi történik ha egy kocka sarkába érkezünk? Felfele, vagy oldalra fogja kihelyezni a karaktert?) -> először elmozdítjuk a karaktert x irányban.
* iterálok azokon a kockákon keresztül, amelyek a karakter körül vannak és megnézem, hogyha szilárdak. Ha a karakterünk ütközik valamelyikkel, szükségünk van rá, hogy kimozdítsuk belőle
* ugyanezt megtesszük az y komponenssel is



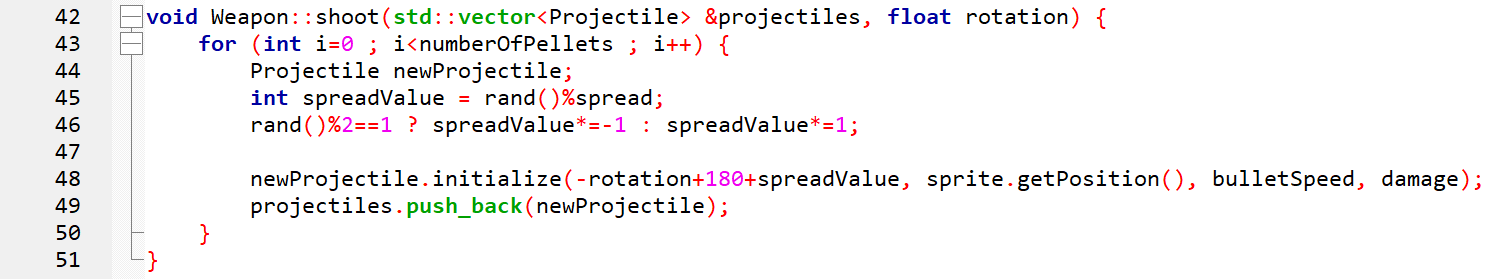
A *fent látható függvény az ütközések vízszintes komponensének megoldása. Elégséges, ha csak akkor nézzük, ha van sebességünk oldalra. Ekkor két for ciklussal megyünk végig a karakter körül levő kockákon és ellenőrizzük, ha szilárd-e és hogy történt-e ütközés. A két belső feltétel arra vonatkozik, hogy éppen balra, vagy jobbra haladt a karakterünk, mivel ez is szerepet játszik az új pozíció kiszámításában – ha jobbra megyünk és ütközünk, balra kell kihelyezzük a karaktert, ha balra megyünk, akkor pedig fordítva.*

Különböző fegyverek kezelése

Ahogyan azt játék közben is észrevehetjük, a különböző felvett fegyverek máshogy viselkednek: egyik gyorsabban lő mint a másik, kevesebbet sebez és egyéb tulajdonságokat fedezhetünk fel még.



A különböző típusú fegyver osztályok inheritálódnak egy fő, Weapon nevű osztályból, amelyben a fent is látható tulajdonságok vannak deklarálva. A Weapon osztály fő függvénye a lövés, amely így néz ki:



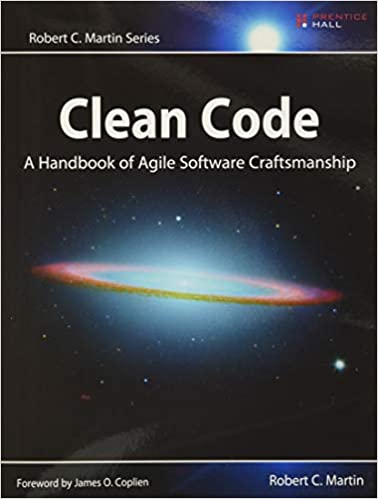
*A függvény paraméterül kapja az előbb említett Projectile típusú std::vector, hogy tudjon egyenesen beszúrni új elemeket. A fenti tulajdonságokat feldolgozzuk a következő képpen és hozzáadjuk az aktuális lövedékhez:  
spread: azt az értéket jelzi, hogy az eredeti iránytól hány foknyi eltéréssel indulhat el a lövedék – ez az érték adja meg azt a tulajdonságát a fegyvernek, hogy mennyire lő pontosan vagy pontatlanul.  
numberOfPellets: for ciklussal annyi lövedéket adunk hozzá a vektorhoz, amennyi az adott fegyvernél az érték – legtöbbnél csak egy lövedéket kapunk egy kattintással, viszont a “Shotgun” típusnál például több is létrejön.  
bulletSpeed: összsebesség, a Projectile típusunknak leadjuk, majd ő abból x és y sebességre bontja fel sin és cos függvények, valamint a forgás szögének segítségével.  
damage: leadódik a lövedékünknek, és ez határozza meg majd, hogy amikor ellenséggel ütközik, annak hány életpontját vegye le.*

1. *Fejlesztési lehetőségek*

Fejlesztési lehetőségek szempontjából rengeteg opció van a játékomnál. Ugyan már így is játszható állapotban van, a lehetőségek limitáltnak tűnhetnek, ezért szeretnék még minél több dolgot hozzáadni, névszerint:

* **Többjátékos mód**. Legelső prioritásom a már említett „multiplayer” játékmód befejezése lenne. Ezzel egy teljesen más élményt tudnák nyújtani a játékosnak és lehetőséget adna arra, hogy többen is tudjanak szórakozni a játékkal egyszerre.
* **Több pálya.** Jelenleg egyetlen pálya található a játékban. A következőkben szeretnék még több pálya felépítést és több kinézetet is hozzáadni. Ezekből választani lehet majd egy újabb menüpontban, amelyet a Singleplayer gombra kattintva érünk el, és csak akkor indul el a játék, miután kiválasztottuk a számunkra megfelelő pályát.
* **Több fegyver.** A jelenlegi választékhoz szeretnék hozzáadni más lövedékű fegyvereket is, például: rakétavető, amelynek lövedéke amikor ellenséget vagy földet ér, robbanást okoz és sugárban sebez. Ehhez szükség lesz a lövedék osztály átdolgozására.
* **Több ellenség.** Prioritásom egy új, repülő típusú ellenség hozzáadása is a játékomhoz.
* **Hang és zenecsatornák különvevése.** A program mostani állapotában a zenéknek létezik egy külön lejátszó, viszont a hangok össze-vissza vannak letárolva, azokban az osztályokban, amelyekhez kapcsolódnak. Ezért nem lehetséges a beállítások menüből külön beállítani a hangoknak és zenéknek a hangerejét. Ennek okán szeretném a hangokat is csoportosítani egy lejátszóba, ahonnan könnyedén meg lehet változtatni a kódolás szintjéről a hangerőt.
* **A map- és egyéb adatok titkosítása.** Jelenleg a pálya adatok szimpla szövegfájlba vannak letárolva, ezeket a felhasználó könnyen tudja módosítani, hogy tisztességtelen előnyhöz juthasson. Ezért szeretnék legalább egy minimális kódolást hozzáadni ezekhez a fájlokhoz, hogy az átlagos felhasználó ne tudjon belenyúlni.
* **Online toplista.** Az általam használt könyvtár, az SFML, a TCP és UDP socketeken kívűl lehetőséget ad a HTTP szerverekkel való kommunikációra is. Egy lehetséges fejlesztés lenne egy online oldal létrehozása, amely a legjobb játékosok eredményeit menti el, és jeleníti meg bárkinek az interneten. Ez a lehetőség azért tetszik nekem nagyon, mivel fejleszthetném ismereteimet nem csak C++ téren, hanem weboldalfejlesztés, design, adatbáziskezelés, networking szempontjából is, valamint egy kis kriptográfiát is tanulhatnék, mivel az érkező és küldött adatokat könnyű másképpen meghamisítani.
* **Általános kód optimalizálások.** Szeretnék még több kommentet hozzáadni az kód átláthatóságáért, valamint néhány helyen algoritmusokat lecserélni, amelyeknél létezik gyorsabb megfelelő.

1. *Szakirodalom*



Programozási szempontjából az alábbi forrásokat használtam fel:

* [Robert C. Martin: Clean Code](https://enos.itcollege.ee/~jpoial/oop/naited/Clean%20Code.pdf)
* <https://www.sfml-dev.org/tutorials/2.5/>
* <https://gamedevelopment.tutsplus.com/tutorials/finite-state-machines-theory-and-implementation--gamedev-11867>

A játékhoz szükséges média tartalmakat (hangok, textúrák) az alábbi oldalakról szereztem:

* https://craftpix.net/freebies/free-market-cartoon-2d-game-tileset
* https://sfbgames.itch.io/chiptone
* <https://opengameart.org/content/land-torn-monster-sprites>
* https://www.soundsnap.com
* https://www.audiomicro.com
* https://freesound.org
* https://www.storyblocks.com
* <https://www.bfxr.net>

A játékban található zenék saját készítésűek.

**Tartalomjegyzék**

1. Témaválasztás 2
2. Rendszer követelmények 2
3. Felhasználói kézikönyv 3
4. Programozói kézikönyv 5
5. Fejlesztési lehetőségek 14
6. Szakirodalom 15