Haladó Programozás beadandó – WhyOpenGL

Takács Sándor Attila – RQRSJG
Készült: 2022.01.17.

Ez a projekt egy 3D game engine elkészítését és egy segítségével létrehozott játék fejlesztését valósította meg. A fájlok és a telepítés menete megtalálható a https://github.com/sanyja2000/whyopengl linken.

Játékmenet:

A játékban a szinteken levő puzzleök megoldásával lehet feloldani a szint zenéjének a darabjait. Minden szinten 4 puzzle található ami 4 darabját játsza le a zenének. Ha mind a 4 puzzle megoldására sor kerül akkor a pálya kártyája felemelkedik és megváltozik a pálya kinézete.

Jelenleg 3 különböző pálya található a játékban, bár ez a későbbiekben bővíthető:

- Ed Sheeran: Shape of you
- The Weeknd: Blinding Lights
- Wham!: Last Christmas

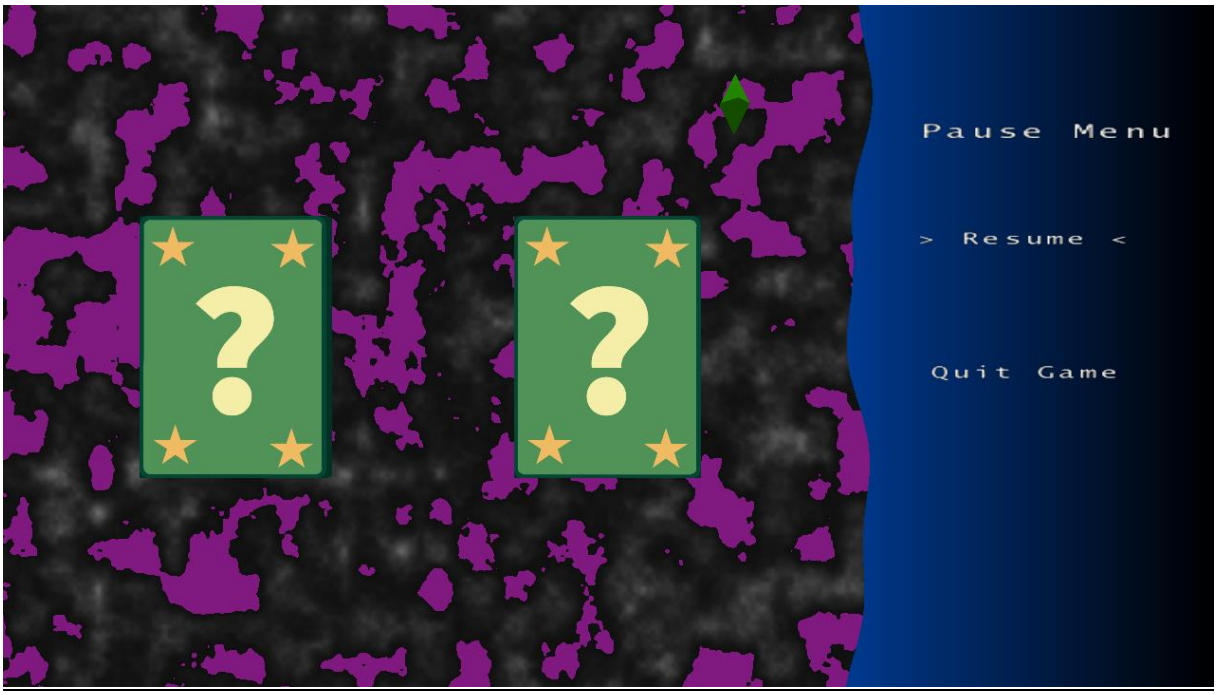
A 4 része a zenéknek Fl Studioban lett létrehozva pluginokkal és kiexportálva .wav fájlokká. Ezek a dob, basszus, akkordok és a dallam. Minden esetben ilyen sorrendben kerülnek lejátszásra párhuzamosan, függetlenül a megoldott puzzleök sorrendjétől.

Főmenü:

A főmenüben láthatjuk a pályákat szimbolizáló kártyákat. A nem felfedezett pályák kártyája egy kérdőjellel van jelölve. Ezek egy pálya teljesítése után megváltoznak a zene kártyájára.

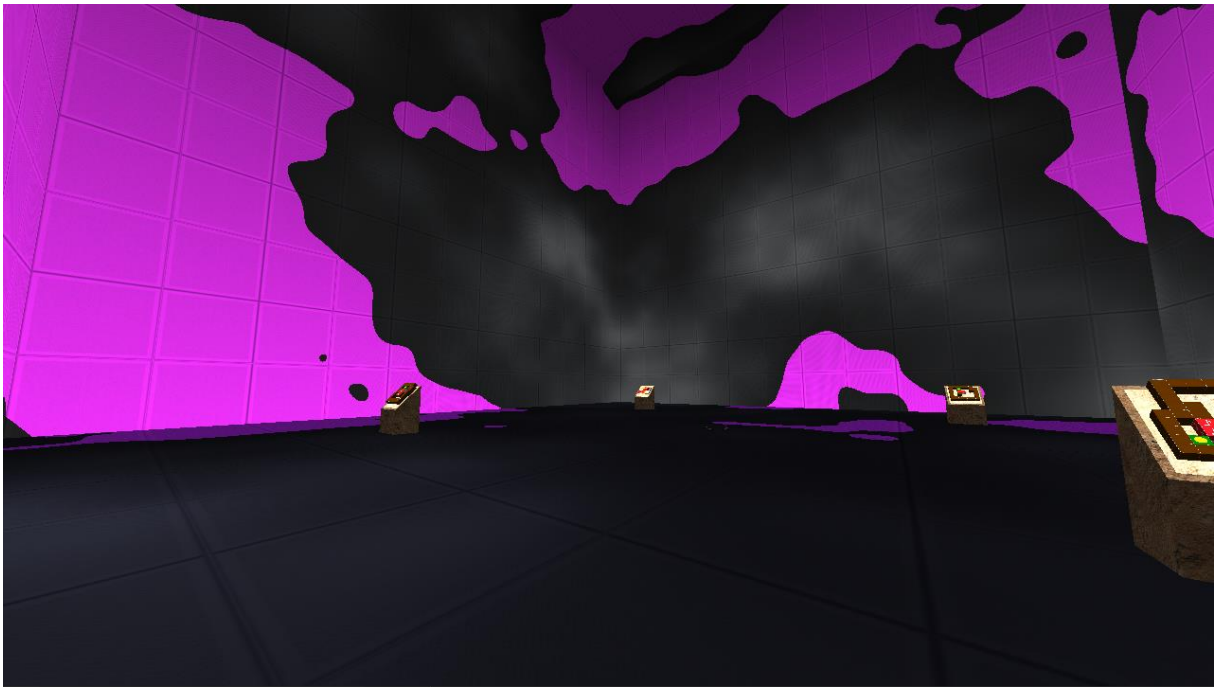
A kártyákra kattintva betöltésre kerül a megfelelő pálya. Ha a kártyák felé visszük a kurzort, akkor egy animáció játszódik le.

Szünet menü:

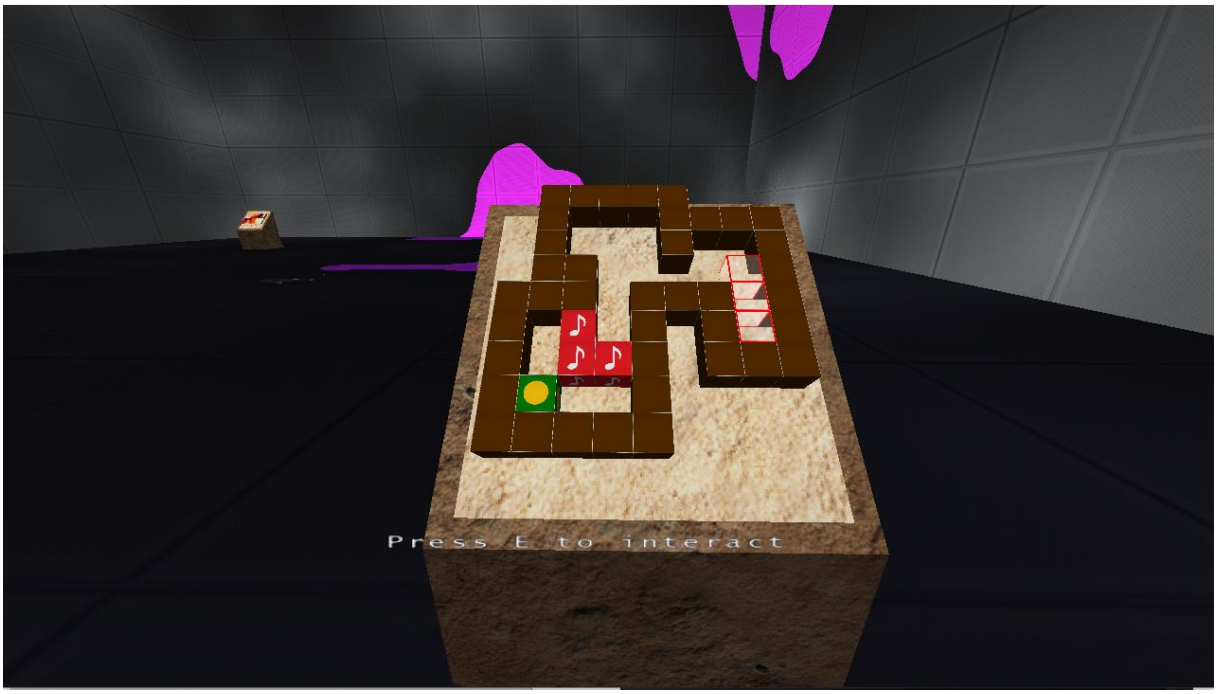


Az ESC billentyű megnyomására megjelenik a szünet menü a képernyő jobb oldalán. Ebben a menüben a [W] [S] [Space] gombokkal navigálhatunk. A jelenleg kiválasztott opció lebegő animációval mozog és >< karakterekkel van kiemelve.

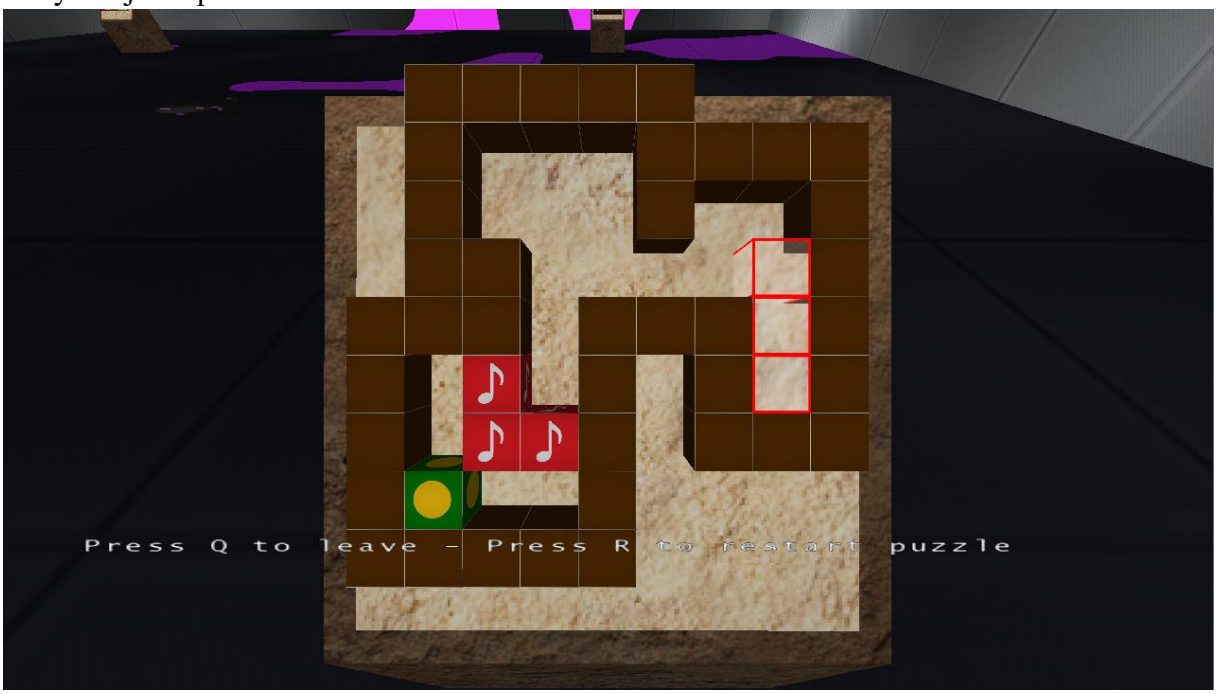
Betöltött pálya:



A betöltött pályán a [WASD] gombokkal és az egér használatával mozoghatunk. Ha egy interaktálható elem mellé érünk akkor az [E] gomb lenyomásával interaktálhatunk vele.

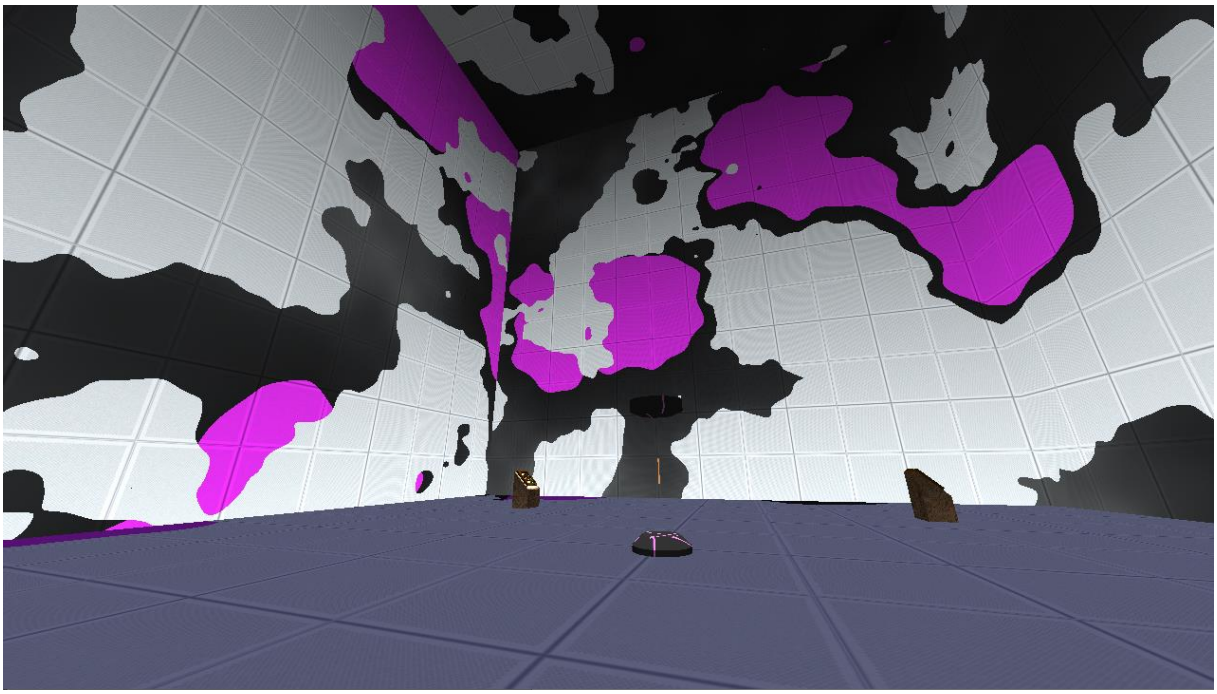


Az [E] gomb lenyomása után a kamera ráfókuszál a puzzle-re és a [WASD] gombokkal irányíthatjuk a puzzlet.



Az [R] gomb segítségével a sokoban játékot újrakezdhetjük.

A [Q] vagy [ESC] gomb megnyomásával kiléphetünk az adott puzzle-ből.



Amennyiben minden puzzle-t teljesítettünk egy szinten akkor a szint „megtisztul” és középen megjelenik a pálya kártyája.

Az [E] gombot megnyomva a játék visszatér a főmenübe, és a teljesített pálya már feloldva lesz.

A kód:

A kód 2 részre van bontva, a játékmotor (ami végzi a 3D renderelést, hangok és objektumok betöltését) az ”engine” mappában található. Az elkészített játék fájljai a főkönyvtárban találhatóak.

A játékmotor: Az engine legfőbb része a 3D kirajzolás. Ezt az OpenGL API valósítja meg.

A **renderer.py** fájlban található az API kód nagy része, valamint különféle segédfüggvények és osztályok. A különböző osztályok funkciói leegyszerűsítve a következőek:

- **VertexBuffer**: A vertex* adatokat tárolja egy bizonyos 3D objektumnak.
- **IndexBuffer**: A vertexbufferben megadott vertexek sorrendjét határozza meg, így elérhető, hogy egy vertex többször is használható.
- **VertexBufferLayout**: Meghatározza, hogy egy vertexbuffer milyen adattípusokat tartalmaz, és mennyit.
- **VertexArray**: Összekapcsolja a vertexbuffert és vertexbufferlayout-ot.
- **Shader**: Egy glsl nyelven írott program ami a videókártyán fut. Minden renderelt objektumhoz kell. Azt szabályozza, hogy az objektum hol(transzformációk), és hogyan(szín, textúra, animációk) jelenjen meg a képernyőn.
- **ShaderHandler**: A különböző shaderek betöltését és használatát segíti elő. Tartalmaz cache-t.
- **Texture**: Betölti és tárolja egy textúra adatait.
- **Renderer**: Ez az osztály felhasználja a megadott vertexarraybuffert, indexbuffert valamint shadert és kirajzolja a megadott elemeket.
- **FPSCounter**: Ez az osztály segíti a diagnosztikai adatok mérését(FPS, deltatime).

* Egy vertex adatok halmaza, amit a shader kap meg rendeleréskor. Általában egy vertex 3 koordinátát, 2 textúra koordinátát és 3 normál koordinátát tartalmaz, de ezek mellett még tartalmazhat bármilyen más tulajdonságot is(pl. szín, anyagminőség stb.)

Az **objloader.py** fájlban a **processObjFile** függvény visszaad egy vertex és indexbuffert amit egy bemeneti .obj fájlból olvas ki, és dolgoz fel.

Az **objHandler.py** fájl tartalmazza az **Object3D** osztályt. Ez az osztály az alapja a létrehozható objektumoknak. Magába foglal egy textúrát, valamint egy .obj fájlból létrehozott geometriát is. Ha létrehozunk egy példányt akkor azt egyetlen draw függvénnyel kirajzolhatjuk. Emelett lehet mozgatni, forgatni, és méretezni a különböző függvényekkel. Az összes játékban látható objektum ezen osztály példánya.

Emelett még tartalmazza a **prefabHandler** osztályt is, ami megakadályozza, hogy egy geometria haszontalanul, többször is be legyen töltve. Ilyen esetben csak egy másolatot csinál az eredetiről, sok időt és memóriát megspórolva.

A **fontHandler.py** fájl egy primitív szövegrenderert valósít meg, egy textúraatlasz metódus segítségével.

Az **audioHandler.py** fájl egy egyszerű 5 csatornás hangkeverőt valósít meg, threading használatával.

A játék:

A **main.py** fájlban található a **Game** osztály. Ez tartalmazza a játék adatait és függvényeit.

Az konstruktorban inicializálva van a játékmotor: beállítjuk az ablak méreteit, az egér és billentyűzet kezelő függvényeket. Ezek után betöltjük a szükséges shader fájlokat a shaderHandlerbe és létrehozzuk a különböző hang és textúra kezelőket. Ezt követően betöltjük a játék pályáját(jelen esetben ez a főmenü) és elindítjuk a mainloopot.

A showScreen függvény tartalmazza a mainloopot. Ez gondoskodik a játék futásáról, és a kilépésig ismétlődik. Először is lekezeljük az ESC gombot, ha le van nyomva akkor megnyitjuk a szünet menüt. Amennyiben a betöltött map típusa menü, akkor az egérmutató pozíciójából kiszámítjuk a kiválasztott kártyát és a megfelelő transzformációt alkalmazzuk rá.

Abban az esetben ha nem a menüben vagyunk, akkor a bemenetek alapján változtatjuk a játékos pozicióját és a kamera forgását. Mindezekután végigmegyünk a pálya betöltött objektumjain és kirajzoltatjuk őket. Ha éppen interakcióban állunk egy objektummal akkor a bemenetek az objektumnak lesznek továbbadva.

Ezt követően megrajzoljuk a felugró szöveget, és a szünetmenüt (ha meg van nyitva), majd megcseréljük a két buffert (a jelenleg ábrázoltat és a rajzolás alatt lévőt).

A **playerController.py** fájlban található a Player osztály. Ez az osztály tartalmazza a játékos adatait, és végzi az interakciókat a játékos és a betöltött pálya között(pl. játkos mozgatása, ütközés a pályával, kamera mozgatása).

Az **inputHandler.py** fájl az egér- és billentyűzetbemenetek kezelője. Ez segíti elő a könnyű interakciókat a bemenetekkel(pl. jelenleg lenyomott billentyűk, egér irányítása stb.)

A **maploader.py** fájlban található **MapLoader** osztály végzi el a pályák betöltését. Ennek a segítségével egy „scene” rendszert lehet létrehozni amiket map mappában található .json fájlokból állít elő. Ez a hierarhia kereshető a getObject függvénnyel a name paraméter alapján. A MapLoader típus szerint dolgozza fel a bemeneti .json fájl objektumait, amit különböző osztályonként ad a hierarchiához. Ilyen osztály lehet például a PuzzlePlane(ami a sokoban játék objektuma) vagy a Map(ami a map shaderével rajzolja meg a pálya geometriáját). A .json fájlban az objektumok példa szintaxisa a következő:

```
{"name":"pp1","puzzleId":"01","type":"puzzlePlane","pos":[-5,0.25,-5],"rot":[0,-
1.57,0],"scale":0.25,"mapfile":"res/puzzles/sokoban10.txt"},
```

Ahol a paraméterek:

- **name**: kötelező paraméter minden elemnek, id-ként szolgál
- **puzzleId**: a puzzle Id-je, elhanyagolható
- **type**: kötelező paraméter, megadja a hozzárendelendő osztály nevét
- **pos**: kötelező paraméter, megadja az objektum pozicióát
- **rot**: kötelező paraméter, megadja az objektum forgását
- **scale**: kötelező paraméter, megadja az objektum méretét
- **mapfile**: kötelező paraméter a puzzlePlane típushoz, a betölteni kívánt puzzle fájlját adja meg

A **classHandler.py** tartalmazza a különböző objektumok osztályait. Minden osztály példányosításkor megkapja a pálya prefabHandlerjét, ami megakadályozza, hogy többször legyen betöltve és feldolgozva ugyanaz a fájl. Ez melett megkapják még a props nevű argumentumot amiben elérhetőek az objektum .json fájlban meghatározott tulajdonságai.

Minden osztályhoz hozzá van rendelve minimum egy 3D objektum, amit a model változó tartalmaz. Ez meghatározza az objektum pozicióját, forgását, méretét, geometriáját és textúráját.

Az osztályoknak kötelező tartalmaznia a **draw** függvényt, ami a megrajzoláshoz kell. Ennek a függvénynek a paraméterei a következők:

- shaderhandler: a megfelelő shader kiválasztásához és felparaméterezéséhez kell
- renderer: a megrajzoláshoz lefutattásához kell
- viewMat: a nézet mátrix, a vetület és kamera mátrixának szorzata, összeszorzódik kirajzoláskor a shaderben az objektum model mátrixával

Opcionális függvényként tartalmazhatják a **moveWithKeys** függvényt az objektumok. Ebben az esetben, ha a játékos interakcióba lép az objektummal, akkor a billentyűzet bemenetei átadódnak az objektumnak, és azok alapján változtatható az állapota az interaktált elemnek.

Opcionális függvényként tartalmazhatják még az **update** függvényt is, ami minden képkocka frissítésnél lefut. Ebben a függvényben elérhető bemenetként az audioHandler és a képkockák között eltelt idő(deltaTime) a fizikai számításokhoz.

A fájlban található osztályok és szerepük:

- **Map**: a pálya geometriáját és textúráját tartalmazza. Külön shadere van és ennek segítségével érhető el az animáció a pálya teljesítésekor.
- **TeleportCrystal**: a pálya teljesítésekor ez a megjelenő kártya és a tartója. Interaktálás esetén betöltésre kerül a főmenü.
- **Decoration**: olyan statikus objektumok számára van létrehozva amik nem állnak interakcióban semmivel, szimplán csak dekoráció szempontjából részei a pályának.
- **PuzzlePlane**: a sokoban nevű játék talpzata. Ezzel interaktál a játékos, és ez tartja gyermekei között a PuzzleBox példányokat amik a puzzle részei. A res/puzzle mappából tölti be a puzzleöket és generálja a hozzátartozó geometriákat. Interakció közben a zöld kockát mozgatjuk és a megoldáshoz a piros hangjeleket a részben átlátszó kockák helyére kell tolni.
- **PuzzleBox**: a PuzzlePlane osztály gyermekeként van csak használva, segédosztály, példányosítása csak a puzzleplane konstruktorában fordul elő.
- **SlidePlane**: a tili-toli játék talpzata. Ezzel interaktál a játékos, és ez tartja gyermekei között a SlidePiece példányokat amik a puzzle részei. A res/puzzle mappából tölti be a kiválasztott képeket és generálja a hozzátartozó geometriákat. Interakció közben az üres helyet mozgatjuk, és megpróbáljuk az eredeti képet visszaállítani.
- **SlidePiece**: a SlidePlane osztály gyermekeként van csak használva, segédosztály, példányosítása csak a SlidePlane konstruktorában fordul elő.
- **ShaderPlane**: ezt használjuk a főmenü hátterének kirajzolására. Külön shaderrel rendelkezik, és ezzel valósítjuk meg a perlin noise-al való animációt.
- **Camera**: a pálya kamerájának a tulajdonságait tárolja, minden pályán csak egy lehet.

- **MenuCard**: a főmenüben található kártyák ezt az osztályt használják. Az update függvényben az egérkurzorhoz megfelelő transzformációt kezeljük le.
- **LoadingScreen**: a fekete téglalap animáció a pályák betöltése közben. Minden pályához hozzá van adva, betöltéskor shrink, átlépéskor grow animációval.
- **PauseMenu**: a szünet menüt foglalja magában. Ez nem egy pálya része, hanem a Game osztály konstruktorában van létrehozva, és ebből kifolyólag nem változik pályánként. Amennyiben menü típusú pálya van betöltve, akkor a „Main menu” felirat nem jelenik meg.

Források:

- Joey de Vries – Learn OpenGL
  https://learnopengl.com/
- Yan Chernikov – OpenGL tutorial
  https://www.youtube.com/watch?v=W3gAzLwfIP0&list=PLlrATfBNZ98foTJPJ_Ev03o2oq3-GGOS2