Fejlesztői kézikönyv

A Squirrel-Run^{béta} nyílt forráskódú videójátékhoz

Általános információk

A játék github repository-ja elérhető bárki számára itt.

A játékhoz a THREE.js szintén nyílt forráskódú WebGL-3D Engine-t használtam, aminek honlapja itt található.

Ebben a fejlesztői kézikönyvben csak az általam írt/definiált függvényeket fogom részletezni, a 3D Engine-ben előre definiált függvények dokumentációja megtalálható itt.

A játék programja (a srcipt.js fájl)

3D Engine alapbeállításai

1-4. sor: importálja a 3D Engine forráskódját, onnan még az `{OrbitControls}`-t, az `{STLLoader}`-t és a `{TWEEN}`-t. Ezek a külső kódok azért kellenek, hogy a játék egyes részei minél esztétikusabban nézzenek ki.

`scene`: létrehozza a játék fő jelenetét.

`camera`: létrehoz egy kamerát, ahonnan a játékos éppen a játék megfelelő részét látja a képernyőn, a kamera képarányának pedig beállítja az aktív felhasználó monitorján futó böngésző képarányát `(window.innerWidth / window.innerHeight)`

`renderer`: létrehoz egy képkocka-renderelőt ami majd a játékos monitorjához igazítva frissíti a képkockákat a kamera aktuális szemszögéből. A 9. sorban pedig szintén hozzáigazítja a képarányát a már említett képarányhoz.

10. sor: a weblaphoz hozzácsatolja a `renderer` által legenerált `canvas`-t, így téve láthatóvá a felhasználó számára a játékot.

11. sor: mivel a `camera` egy 3D objektum, ezért van térbeli pozíciója / aktuális koordinátája. Ez a sor beállítja a camera pozícióját az általunk ideálisnak vélt helyre.

`textureLoader`: létrehoz egy textúra betöltőt, amivel később a játék hátterét állítjuk be.

A 3D modellek betöltése

17.sor: deklarál 7 változót, amik majd csak a `load3DModels()` függvényben kapnak értéket.

`hillGroup`: egy csoport, amibe majd belekerül egy lejtő, és 2 deszka. Azért kell csoportba foglalni őket, mert egy 3D modellnek egy színt tudunk adni, viszont ha elsőnek külön színezzük őket, és aztán foglaljuk csoportba, akkor már lehet őket egy 3D objektumként kezelni.

`barrierObjects`: egy lista, amibe majd később az akadályok 3D modelljeit fogjuk belerakni.

`load3DModels()`:

`loader`: 39.sorban deklarál egy 3D modell-betöltő változót, aminek segítségével tudjuk betölteni a modelleket.

`loader.load()`: első paraméterének meg kell adnunk a betöltendő modell elérési útvonalát (hogy milyen mappában van), amit be fog tölteni, a 2. paramétere pedig egy lokális függvény, amiben többek között transzformációkat hajthatunk végre a betöltött mostmár 3D objektumon, majd ha kell hozzáadhatjuk a jelenethez.

az általunk elvégzett transzformációk a mamut objektumon:

Először is lekérjük az eredeti modell hosszát a különböző tengelyeken, majd a `mammut`-ot arányosan skálázzuk úgy, hogy X tengelyen 1 egység széles legyen. Ezekután pozícionáljuk, majd hozzáadjuk a jelenethez. Az utóbbiakat megcsináltuk mind a 7 modellen, valamelyiknek a szélessége helyett a magasságát állítottuk 1 egységre (lásd: 57. sor), lejtő megfelelő részeit beleraktuk a `hillGroup`-ba, és az egyes akadályokat beletettük az akadályok összesített listájába.

a `load3DModels()` függvény definiálása után rögtön meg is hívódik (123. sor), így biztosítva a játék betöltésének leghatékonyabb módját.

`setSceneBackground()`: egy függvény, ami a `fileName` paraméter értékét beállítja a jelenet hátterének (ez a night mode-hoz szükséges)

`rAF`: egy változó, ami majd csak a `game3D()` függvény meghívásakor kap értéket, neve a requestAnimationFrame() függvényből ered

`isStopped`: egy változó, ami a menü ki-be kapcsolásához szükséges

`loadingScrene()`: ez a függvény hívódik meg legelőször, ez felelős a betöltő képernyőért

A billentyűlenyomás detektálása

`startEventListening()`: olyan függvény, amely meghívásakor elkezdi „figyelni”, hogy a játékos mikor nyomja le az adott billentyűt. Ha a játékos lenyomott egy billentyűt, akkor megnézi, hogy melyik billentyűt is nyomta le, és aszerint hajtja végre az utasításokat, természetesen ha az adott feltétel igaz. A feltételek a következők:

- Ha a játékos az `'A'` vagy `'D'` gombot nyomta le:
    - a `mamut` pozíciója az X tengelyen <= mint 2,
    - a `needToAnalyzeObjects` változó értéke igaz,

    akkor a `squirrel`-t, a `mamut`-ot, és a `camera`-t egy egységgel balra vagy jobbra lépteti (változtatja a koordinátájukat) a következők szerint:

    balra lépés: `'A'` gomb

    jobbra lépés: `'D'` gomb
- Ha a játékos a `'Space'` gombot nyomta le:
    - a `needToAnalyzeObjects` változó értéke igaz,

    akkor a `squirrel`-t, a `mamut`-ot, és a `camera`-t egy egységgel felfelé mozgatja (azaz változtatja az Y koordinátájukat), és meghívja az `animateObjectMotion()` függvényt, majd 200 ms-al utána visszaesnek a talajra (az Y koordinátájuk 0 lesz).
- Ha a játékos az `'Esc'` gombot nyomta le:
    - Megjeleníti / eltünteti a menüt, aszerint, hogy eddig látható volt-e vagy nem.

- ha a menü eddig nem volt látható, akkor
    - megszünteti a renderelő függvényt: `cancelAnimationFrame(rAF)`
    - megállítja az akadályok jövésének ismétlését: `clearInterval(barrierInterval)`
    - a játék átlátszóságát a felére (50%-ra) csökkenti (155. sor)
    - a menün belüli `'switch'` gombokat elkezdi figyelni, hogy kattint-e rájuk a játékos
    - az `isStopped` változó értékét negálja
    - láthatóvá teszi a menüt
- ha a menü eddig látható volt, akkor
    - újra meghívja a `game3D()` függvényt, aminek következtében a játékos újra tud már játszani (158. sor)
    - újra elkezdi az akadályok jövésének ismétlését (159. sor)
    - abbahagyja a menün belüli 'switch' gombok kattintás-figyelését.
    - a játék átlátszóságát visszaállítja 1-re (100%-ra)
    - eltünteti a menüt

- Ha a játékos újraméretezte a böngészőjét:
    - beállítja a `renderer` és a `camera` képerányát az új adatoknak megfelelően

A játék előkészítése

`oakLogRotations`: a betöltött `oakLog` objektum lehetséges elforgatásainak listája, az `init()` függvényben kap értéket, radiánban

`motionDuration`: adott karakter mozgásának (X tengelyen) időtartalma ms-ban

points: a játékos eddigi elért pontjai.

coinPoints: a játékos által eddig megszerzett $SR pénzérmék száma

increment: az akadályok sebessége

speed: két akadály inicializálása között eltelt idő, ms-ban

init():

- az oakLogRotations-nak beállít 2 lehetséges értéket: vagy 90 fok, vagy 0
- motionDuration: 200ms
- points: 0, ezt növeli majd a game3D() függvény egyik része
- speed: 2000ms
- controls: ezzel lehet változtatni manuálisan a camera aktuális nézőpontját
- floor: ez a játék padlója, ami fehér színű téglatest, pozícionálva
- 188. sor: hozzáad a jelenethez egy félgömb fényforrást
- increment: 0.2, ennyivel fog változni az akadályok pozíciója az aktuális FPS szám függvényében
- elforgatja és pozícionálja a hillGroup-ot, majd belerakja a barrierObjects-be

toggleNightMode(): Ha meghívjuk, beállítja a játék (jelenet) háttérképének forrásfájlát a document.getElementId('setting').innerHTML értékének megfelelően.

animateObjectMotion(object, coordinates, duration): az object paraméternak megadott objektumot animálva elmozdítja a megadott coordinates koordinátákra, duration időtartam alatt.

randomBarrier: egy random szám 0 és 2 között

activeBarriers: lista, melynek aktuális elemei az activeBarriersGroup-ok

`activeBarriersGroup`: lista, melybe az éppen klónozott akadály adatai kerülnek

`barrierInterval`: az egyes akadályok generálása között eltelt idő

`activeCoins`: az aktuálisan legenerált pénzérmék nyilvántartó listája

`freeSpaceForCoins`: a legenerálandó pénzérme lehetséges X koordinátájának listája

`activeCoinX`: az éppen legenerált pénzérme X koordinátája

`isOakLogVertical`: logikai változó, értéke attól függ, hogy az `oakLog` klónja vízszintes vagy függőleges-e.

`needToAnalyzeObjects`: logikai változó, értéke attól függ, hogy a `game3D()` függvény objektum mozgató része épp fut-e.

`updateCoins()`: 500ms-ként leklónozza a `coin`-t, majd belerakja az `activeCoins`-ba, `activeCoinX` koordinátával.

`initializeCoin(object, x, z)`: az `object` paraméternek megadott objektumot (ebben az esetben egy klónozott pénzérmét pozícionálja a megadott `x, z` koordinátákra. Ezután belerakja az `activeCoins`-ba, majd hozzádja a `scene`-hez

`updateBarriers()`: speed időközönként meghívja az `initializeBarrier()` függvényt, aminek a barrierObjects listából randomizáltan kiválasztott objektum klónját adja meg első paraméternek, második paraméternek a kiválsztott objektum indexét adja meg, így biztosítva, hogy az akadályok randomizáltan jöjjenek a `squirrel` felé.

`initializeBarrier(object, randBarr)`: pozícionálja az akadály, randomizáltan

`gameOver()`: ez a játék végén meghívódó függvény, ami megállítja a `game3D()` függvény azon részét, ami csak akkor fut le, ha a `needToAnalyzeObjects` igaz.

`game3D()`: ez az a függvény felelős a `renderer` képfrisstésért, az adakályok pozíciójának elemzéséért. Ha a `needToAnalyzeObjects` értéke igaz, akkor elkezdi mozgatni az `activeBarriers`-ben lévő akadályokat és az `activeCoins`-ban lévő pénzérméket mozgatni a mókus felé. Ha mókus egy akadállyal ütközött, akkor meghívja a `gameOver()`-t, ha egy `coin`-nal

ütközött, akkor pedig eltünteti az adott pénzérmét, és növeli a `coinPoints`-t. Különben (azaz amikor a játékos „meghalt”), a `camera.lookAt()` függvénynek beállítja a `{x: 0, y: 0, z: 0}` (world origin) értéket.