Zbornik radova Fakulteta tehničkih nauka, Novi Sad

UDK: 004.9
DOI: https://doi.org/10.24867/07BE36Zivlak

PROGRAMSKO REŠENJE ZA RAZVOJ MEĐUSCENA JUNITI POGONA

SOFTWARE SOLUTION FOR UNITY INTERSCENE DEVELOPMENT

Stevan Zivlak, *Fakultet tehničkih nauka, Novi Sad*

Oblast – RAČUNARSTVO I AUTOMATIKA

Kratak sadržaj – *Cilj ovog rada jeste pronalazak rešenja problema razvoja međuscena korišćenjem Juniti pogona. U radu se koriste C# skripte da bi se aktivirale međuscene u ključnim trenucima, kao i za sinhronizaciju trenutaka izvršavanja događaja unutar njih. Skripte se oslanjaju na interne Juniti mehanizme za detekciju kolizija. Ova solucija pruža velik stepen kontrole, ali postoje bolja rešenja kada je u pitanju brzina razvoja.*

Ključne reči: *Međuscene, razvoj video igara, Juniti pogon*

Abstract – *This paper aims to present a solution for the problem of cutscene development using Unity engine. C# scripts are used to listen for key game events, trigger cutscenes when such events occur and synchronize the order and timing of events within them. The scripts use internal Unity mechanisms for collision detection. This solution gives unprecedented control, but better solutions exist in terms of development speed.*

Keywords: *Cutscenes, game development, Unity engine*

1. UVOD

Većina video igara današnjice ima u sebi međuscene – delove igre sa smanjenim nivoom interakcije koji blokiraju gejmplej. One se obično koriste u svrhu proširivanja narativa igre, ili kao oruđe za izazivanje određenih emocija kod igrača kao što je osećaj gubitka kontrole i straha.

Razvoj međuscena je po svojoj prirodi problem automatizacije dešavanja događaja. Uključuje kontrolu kretanja modela u igri i kontrolu animacija. Dodatno, potrebno je kontrolisati i zvučne i specijalne akcione efekte. Na kraju, svi pomenuti elementi moraju se vremenski sinhronizovati među sobom.

Pronalazak adekvatnog programskog rešenja za razvoj međuscena veoma je bitno za postizanje balansa između brzine razvoja i mogućnosti kustomizacije koja je potrebna. Izbor Juniti pogona kao razvojnog okruženja motivisano je delimično tehničkim mogućnostima koje nudi, a delimično ogromnom rasprostranjenošću istog u nezavisnom razvoju video igara [1].

NAPOMENA:
Ovaj rad proistekao je iz master rada čiji mentor je bio dr Dragan Ivetić, red. prof.

2. GRADIVNI ELEMENTI MEĐUSCENA

Međuscene sadrže vizuelne i zvukovne gradivne elemente koje se prikazuju na ekranu. Od ključnog je značaja da su ovi elementi konzistentnog stila da bi komunikacija sa igračom bila efektna.

2.1. Vizuelni gradivni elementi

U prošlosti je 2D animacija u video igrama pravljena tako što bi se ručno crtao svaki frejm. Problem kod ovog pristupa je što je potrebno nacrtati ogroman broj slika za svaku animaciju (igre tipično imaju od 30 do 60 frejmova u sekundi), a treba i uložiti napor da se obezbedi da je prelaz između svaka dva frejma konzistentan. S druge strane, ovaj pristup omogućava da se art tim više posveti svakom frejmu, pogotovo senčenju, što daje finalnom proizvodu iluziju dubine koju imaju 3D modeli.

Slika 1. *Vektorski crtež*

Moderna kompjuterska animacija se oslanja na vektore i interpolaciju da bi smanjila količinu posla koji treba da se obavi. Na slici 1 (levo) se može videti izgled jednog neobojenog vektorskog crteža u programu za kreiranje vektorske grafike Adobe Illustrator CC. On je skup tačaka i linija koje ih povezuju.

Ono što je posebno kod ovakvih crteža jestee što je svaki deo lika moguće selektovati i nezavisno pomerati ili menjati. U sredini slike se može videti originalan izgled istog vektora u obojenoj varijanti, a desno je primer izmene gde je leva noga rotirana, brkovi su obrisani, a svetlo na kacigi je promenilo boju.

Na svakom delu vektorskog crteža moguće je izvršiti rotaciju, translaciju ili promenu veličine bez ikakvog dodatnog crtanja. Na mestima na neobojenom crtežu gde su tačke moguće ih je povući i izmeniti oblik okolnih linija, a moguće je i dodati nove tačke na mestu na kome ih nema. Takođe još jedna pogodnost vektora jeste što, iako su neki delovi crteža iznad nekih drugih (na primer brkovi prekrivaju deo lica), ni jedan deo se ne briše crtanjem preko njega, već se samo prekriva.

Slika 2. *Samo pet ključnih frejmova za animaciju*

Na slici 2 prikazana je animacija umiranja karaktera koja je napravljena korišćenjem originalnog vektora. Pet prikazanih slika nisu klasični frejmovi već mnogo ređi – ključni frejmovi. Ključni frejmovi reprezentuju najbitniji deo animacije i na osnovu njih se korišćenjem pogona u kome se razvija igra vrši interpolacija frejmova koji nedostaju. Ovo je ogromna ušteda na vremenu.

Na kraju procesa vektorske animacije obično se rasterizuju u bitmap formate, a 2D bitmap slike se često nazivaju sprajtovi.

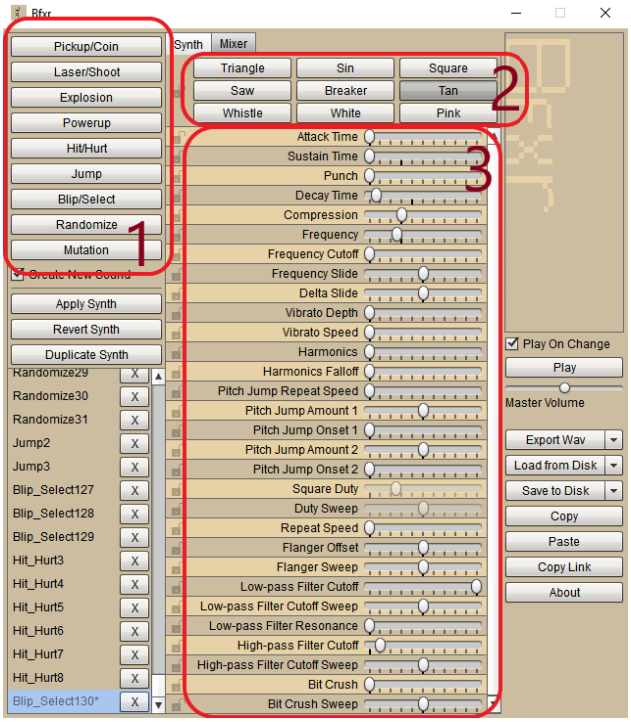
Da bi se renderovanje na ekran ubrzalo, praksa je da se velik broj sprajtova stavi u jedan fajl, a zatim taj fajl importuje u pogon u kome se razvija igra. Juniti pogon ima sistem za sečenje koji omogućava da se oni tokom razvoja koriste odvojeno, ali se svakako zajedno učitavaju u VRAM memoriju.

2.2. Zvukovni gradivni elementi

Zvučni elementi međuscena imaju neprikosnoven uticaj na atmosferu, emotivni tonalitet i osećaj uzbuđenja koji se potencijalno može dočarati korisniku. Ovaj nevidljivi uticaj postiže se pažljivim izborom elemenata muzičke podloge i zvučnih efekata koji poseduju međusobnu harmoniju tako da ni jedan ne ostavlja utisak kao da nije deo celine. Jedan od izazova u radu na ovom projektu bio je pronalazak efekata koji ispunjavaju ove kriterijume, dok sa druge strane imaju autore koji njihovo korišćenje nisu ograničili veoma restriktivnim licencama. U ovom cilju, korišćeni su – osim za dijalog – sajtovi koji ohrabruju mlade autore i ljude koji prave video igre da podele svoj rad sa drugima kao što su opengameart.org i freesound.org.

Razgovori u međuscenama su predstavljeni kroz progresivni prikaz slova na ekranu koje prate sintetički zvuci razvijeni uz pomoć programa Bfxr. Bfxr omogućava izbor velike količine parametara pri generisanju zvuka, kao što su frekvencija, oblik zvučnog talasa, njegova delta, dužina trajanja amplitude i mnogi drugi. Na slici 3 možemo videti tri izdvojene celine sa opcijama programa. U celini **1** su prečice koje omogućavaju generisanje nasumičnih zvukova koji podsećaju na zvuk raznih efekata kao što su eksplozije, udarci i slično. Zanimljivo je da je ovo odlična početna tačka jer se klikćući na neku od 9 opcija može doći do zvuka koji je blizu onoga što je željeno. Zatim, daljim korigovanjem oblika zvučnog talasa pod **2** i lepeze opcija pod **3** dolazi se do finalnog zvuka.

Izbor pod **1** u ovom slučaju uvek je bio *Blip/Select* jer su za ovu namenu bili potrebni (zbog povezanosti izgovora jednog znaka i svakog ponavljajućeg zvučnog efekta) kratki zvuci uglavnom niske frekvencije – u zavisnosti od toga koji lik govori. Visoka frekvencija može biti neprijatna kod ponavljajućih zvukova. Zanimljivo je napomenuti da svaki dodatan klik na *Blip/Select* daje sličan, ali donekle nasumičan ton, što znači da je preporučljivo kliknuti nekoliko desetina puta dok se ne dobije ton koji je najbliži onome što se traži.



Slika 3. *Opcije programa Bfxr*

Što se tiče parametara pod brojem **2**, korišćene su opcije *Whistle* i *Breaker*. *Whistle* je klasičan sinusoidni talas, sa dodatkom još jednog sinusoidnog talasa veoma niske amplitude koji ga prati na 20x većoj frekvenciji. *Breaker* je talas baziran na kvadratnoj funkciji koji je u suštini aproksimacija trougaonog talasa. Poredeći ova dva zvuka možemo uočiti da je *Breaker* homogeniji i kraći, dok je *Whistle* piskaviji i rezonantniji.

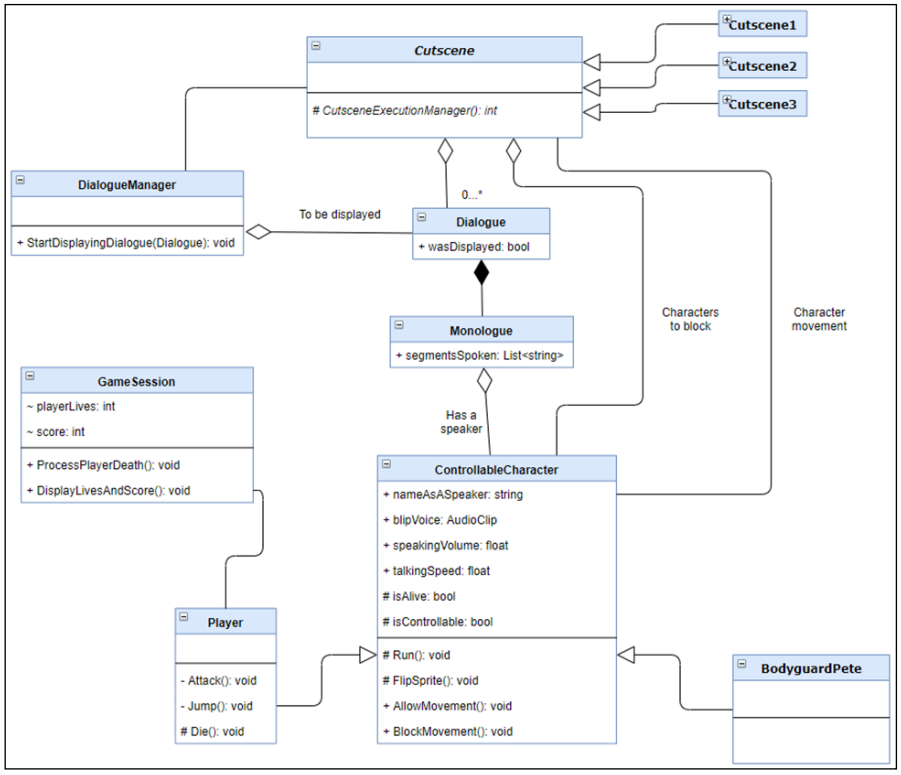
Veća izmena ostalih parametara pod **3** uglavnom nije bila potrebna jer su podrazumevane vrednosti od *Blip/Select* dale odlične rezultate. Frekvencija je naravno korigovana, a *Attack Time* i *Sustain Time* su ostali na niskim vrednostima zbog kratkog vremena trajanja koje je bitno za ovaj zvuk.

3. PROGRAMSKO REŠENJE

Elementi programskog rešenja imaju 3 prirodne celine: kontrolabilni likovi, dijalog i međuscene. Na slici 4, može se videti dijagram klasa celokupnog rešenja, na kome je očevidna povezanost između pomenutih celina – klasa *Cutscene* oslanja se na funkcionalnosti klasa *Dialogue*, *DialogueManager* i *ControllableCharacter* da bi međuscene u igri imale u sebi likove koji vode dijalog, imaju sposobnosti i kretanja i animacije koja to kretanje prati.

3.1. Dijalozi i njihov menadžer

Mnoge međuscene imaju u sebi animirane dijaloge. Svaki dijalog je definisan kao serija neprekidnih monologa (monolog može biti i samo jedna rečenica) koje vode različiti govornici. Svaki monolog se u memoriji čuva kao serija stringova koji predstavljaju rečenice koje on sadrži.



Slika 4. *Dijagram klasa programskog rešenja*

Pošto ima referencu na kontrolabilnog lika koji drži monolog, pristup klasi *Monologue* omogućava nam da odredimo i brzinu kojom govori taj lik, zvuk koji pravi, ime koje treba da se prikaže na ekranu i ostale informacije potrebne za animiranje dijalog oblačića. Osim serije monologa, klasa *Dialogue* ima i *bool* promenjivu *wasDisplayed* koja je namenjena da omogući proveru da li je dijalog prikazan. Ova promenjiva je namenjena za sinhronizaciju više dijaloga unutar međuscena, jer omogućava slanje upita instanci klase *Dialogue* koja daje odgovor na pitanje da li ju je u celosti prikazao menadžer dijaloga nakon što mu je prosleđena.

Klasa *DialogueManager* zadužena je za animaciju dijaloga na ekranu. Njene glavne metode su *StartDisplayingDialogue* i *DisplayNextSentence*.

StartDisplayingDialogue je javna metoda koja se može pozvati iz bilo koje tačke projekta, i kroz nju se pristupa svim funkcionalnostima dijalog menadžera. Ova funkcija kada dobije objekat dijaloga koji treba da prikaže izvršava animiranje otvaranja oblačića za prikaz dijaloga postavljanjem vrednosti parametra *IsOpen* u komponenti animatora na istinitu. Referenca na dijalog se čuva, a zatim se prikazuje prva rečenica prvog monologa koji on sadrži pokretanjem funkcije *DisplayNextSentence*. Pritiskom na dugme *space*, ova funkcija se pokreće ponovo i ispisuje narednu rečenicu. Kada više nema rečenica za ispis, *IsOpen* se postavlja na neistinitu vrednost, čime se animira zatvaranje oblačića za dijalog.

3.2. Međuscene

Međuscene su realizovane korišćenjem apstraktne klase *Cutscene* koja je neka vrsta šeme međuscene i konkretnih realizacija pojedinačnih međuscena koje je nasleđuju (označene su kao *Cutscene1*, *Cutscene2* i *Cutscene3* na dijagramu klasa). Klasa *Cutscene* funkcijom *OnTriggerEnter2D* proverava da li je došlo do kontakta sa objektom koji ima tag **Player**:

```
protected virtual void OnTriggerEnter2D(Collider2D collision)
{
    if (!cutsceneTriggeredAlready)
    {
        if (collision.gameObject.tag == "Player")
        {
            for(int i=0; i<charactersToBlock.Length; i++)
            {
                charactersToBlock[i].BlockMovement();
            }
            cutsceneTriggeredAlready = true;
        }
    }
}
```

Ukoliko jeste, funkcijom *BlockMovement* se blokiraju likovi iz liste kontrolabilnih likova koja se unosi iz Inspektor prozora. Na kraju, promenjiva *cutsceneTriggeredAlready* se postavlja na istinitu vrednost. Ovim se aktivira petlja u *Update* funkciji koja se poziva svaki frejm igre:

```
// Update is called once per frame
protected void Update () {
    if (cutsceneTriggeredAlready && !cutsceneCompleted)
    {
        if (CutsceneExecutionManager() == 1)
        {
            if (delay <= 0)//very small delay to avoid space from
            {           //skipping dialogue to also trigger jumping
                cutsceneCompleted = true;
                UnblockObjects();
            }
            else
            {
                delay -= Time.deltaTime;
            }
        }
    }
}

//this function should return 1 when it's done
protected abstract int CutsceneExecutionManager();
```

Ona poziva funkciju *CutsceneExecutionManager* sve dok ona ne vrati vrednost 1, a onda završava sa radom i vraća korisniku kontrolu nad nizom likova koji su prethodno blokirani. Ovo se ne čini odmah, već sa minimalnom pauzom. Razlog za postojanje pauze što je isto dugme u igri (*space*) određeno kao dugme za skok, i dugme za preskakanje trenutne rečenice dijaloga. Ako ne bi bilo pauze posle kraja dijaloga i ako bi do kraja korisnik došao preskakanjem poslednje rečenice – na kraju dijaloga bi kontrolabilni likovi izvršili akciju skok iako to ne bi bilo predviđeno ponašanje sa aspekta onoga što korisnik očekuje. Funkcija *CutsceneExecutionManager* je definisana kao apstraktna funkcija bez parametara koja vraća brojčanu vrednost. Svaka klasa koja nasleđuje klasu *Cutscene* dakle mora da sadrži implementaciju ove funkcije. Ona je namenjena da definiše sve automatizovane radnje na ekranu, kretanja igrača, dijalog i ostala ponašanja koja definišu tu jedinstvenu međuscenu. Na sledećem primeru se može videti jedna takva implementacija:

```
protected override int CutsceneExecutionManager()
{
    //display first dialogue
    if (displayDialogueOrder)...
    //lay down
    if (dialogues[0].wasDisplayed && !laidDown)...
    //dim the screen
    if (laidDown && !screenDimmingComplete)...
    //undim the screen
    if (screenDimmingComplete && !screenUndimmmingComplete)...
    //get up
    if (screenUndimmmingComplete && !gettingUpComplete)...
    //display 2nd dialogue
    if (gettingUpComplete && secondDialogueDisplayOrder)...
    //move blackopsguy and display third dialogue
    if(dialogues[1].wasDisplayed && !blackOpsGuyFinished)...
    if (dialogues[2].wasDisplayed)
    {
        return 1;
    }
    else
    {
        return 0;
    }
}
```

U međusceni u kojoj se implementira ova funkcija postoji veliki broj akcija, koje se izvršavaju hronološki zahvaljujući mnogobrojnim *if* naredbama. Npr. prva naredba se izvršava samo ako se dobije naređenje za ispis dijaloga, druga ako je dijalog ispisan izvršava animaciju leganja na zemlju, treća ako je lik legao na zemlju zamračuje ekran itd. Svaka naredba se izvršava samo ako je prethodna zadovoljena. Na kraju, ako se poslednji dijalog uspešno ispiše, vraćanjem vrednosti 1 se završava međuscena. Na ovaj način ova funkcija implementacijom samo jedne funkcije ima apsolutnu kontrolu nad događajima koji je čine i njihovim redosledom izvršavanja.

Jedna od osnovnih akcija u međuscenama – automatizovano kretanje likova – realizovano je u ovom projektu na isti sličan kao i kretanje uz pomoć tastature ili drugih ulaznih uređaja. Jedina razlika kod ovakvog kretanja je što brzina nije zavisna od ulaznih uređaja već je fiksna, a momenat (ili lokacija) početka i kraja kretanja su već unapred poznati. Momenat početka je u ovom slučaju utvrđen korišćenjem upravo pomenutih *if* petlji u funkciji *CutsceneExecutionManager* – tj. on zavisi od akcija koje prethode kretanju lika. Da bi se odredio momenat kraja, korišćen je poseban Juniti objekat sa skriptom i kolajderom koji je povezan sa njom. Kolajderi su Juniti komponente koje detektuju sudare objekata i omogućavaju programsko reagovanje na njih. Kada se lik koji se automatski kreće susretne sa kolajderom, skripta ovog objekta menja jedan od parametara skripte međuscene *Cutscene 2* i sa ovom izmenom dolazi do stopiranja kretanja lika. Ovo je veoma pogodno rešenje ukoliko je potrebno izvršiti promene jer se iz Juniti editora uvek na veoma lak način može promeniti lokacija kolajdera.

4. ZAKLJUČAK

U ovom radu su definisane međuscene kao sastavni deo igara i predstavljeni su njihovi gradivni elementi kao i načini njihovog razvoja i izbora. Nakon toga predstavljene su tehnike programskog razvoja specifične za Juniti okruženje, i na kraju, programsko rešenje za razvoj međuscena.

Jedan od problema koji se ispoljio tokom radu je relativno spora kreacija novih međuscena zbog potrebe za „ručnom“ sinhronizacijom elemenata svake međuscene jer su razvijene sa kompletnim oslanjanjem na skripte. Ovaj problem bi mogao da se reši korišćenjem *GameObjectRecorder* klase iz *UnityEditor* biblioteke. Ona se vezuje za Juniti objekte i snima vrednosti u vezi sa njihovom promenom položaja u sceni. Upotrebom ove klase bilo bi moguće snimiti pokrete likova koji bi se kasnije automatski pomerali na isti način. Ovo bi drastično ubrzalo rad na međuscenama.

Dalji pravac razvoja mogao bi biti razvoj međuscena sa grananjem, koje bi omogućile da se priča igre račva na nekoliko mesta. Ovako bi se igraču vratio stepen kontrole, čiji gubitak je jedan od čestih kritika u igrama koje imaju velik broj međuscena.

5. LITERATURA

[1] https://boingboing.net/2018/07/17/the-most-popular-engines-for-i.html (pristupljeno u oktobru 2019.)

[2] S. Lavelle, „Bfxr – Make Sound Effects for Your Games“, https://www.bfxr.net/, Bfxr – program za generisanje zvučnih efekata.

Kratka biografija:

Stevan Zivlak rođen je u Novom Sadu 1990. god. Master rad na Fakultetu tehničkih nauka iz oblasti Elektrotehnike i računarstva – odbranio je 2019.god.
kontakt: szivlak@gmail.com