Viktor Slavković

ECS 2D Scriptable Game Maker

U ovom radu je opisan jednostavni metod izrade 2D igara zasnovan na arhitekturi Entity Component System (ECS) i generalizovanoj organizaciji ovakvih igara. Organizacija je u potpunosti vođena događajima, a reakcije na događaje u igri dalje definišu prostorni i vremenski odnos objekata. Kompleksni događaji su programirani skriptama za koje je napisan skript jezik specifične namene – YANSL. Rezultat rada pokazuje da je primenom ovog metoda moguće lako izgenerisati svrsishodnu multimedijalnu igru za čiji razvoj nije potreban visok nivo znanja programiranja jer je težište razvoja premešteno na opisivanje karakteristika objekata u igri i njihovih odnosa.

Uvod

Pravljenje igara je složen i sveobuhvatan proces, pa je cilj programera da takav proces što više uproste implementacijom tehnika koje će generalizovati zajedničke celine u igrama. Ovakve generalizacije su najčešće predstavljene korisnički-orijentisanim programskim paketima koji omogućuju razvoj igara uz minimalno poznavanje programiranja. Dobar primer toga je popularni Game Maker kompanije YoYo Games.

Cilj ovog projekta je pokušaj da se napravi jednostavan metod za izradu 2D igara u vidu jednog takvog programa koji je zasnovan na Entity Component System (ECS) arhitekturi. Zadatak je da se olakša razvoj igara time što je promovisana generalizovana organizacija koja predstavlja osnov za laku izradu takvih igara. Generalizovana organizacija podrazumeva:

- Opštu logičku osnovu koja se nadograđuje definisanjem scena u igri, objekata u scenama, kao i definisanjem prostornih i vremenskih odnosa tih objekata.
- Tehničku osnovu za realizaciju ideje koja obuhvata interakciju korisnika sa igrom putem ulaznih uređaja računara i mogućnost reprodukovanja raznovrsnog multimedijalnog sadržaja.
- Jednostavan skript jezik kojim se definiše ponašanje objekata u igri.

Osnovna ideja je da se celokupna definicija igre smesti u projektni fajl iz kojeg će se generisati igra u vidu izvršne aplikacije. Kako je struktura igre hijerarhijska, za skladištenje projekta je izabran XML format. C++ programski jezik je često korišćen za razvoj igara jer omogućava kompajliranje izvornog koda u brze nativne aplikacije, pa je generisani kod igre u ovom projektu takođe u tom jeziku. Za kompajler je izabran MinGW (Minimalist GNU for Windows) zbog mogućnosti integracije u projekat i lake i besplatne isporuke.

Arhitektura Entity Component System (ECS)

Ideju i koren ECS arhitekture je prvi put široj publici predstavio Scott Bilas na GDC (Game Developers Conference) (2002). Ovu temu je na GDC Canada 2009. osvežio Marcin Chady u izlaganju „Theory and Practice of Game Object Component Architecture" (Chady 2009).

Entity Component System predstavlja relativno nov šablon za izradu igara koji promoviše drugačiji pristup organizacionoj logici igara. Kako sam naziv kaže, ECS igra se sastoji iz tri dela:

Viktor Slavković (1995), Kraljevo, Beogradska 44/3-9, učenik 3. razreda Gimnazije Kraljevo

MENTOR: Milan Gornik, GSE, Crvenka

Entity. Svaki objekat u igri predstavlja entitet koji se sastoji iz komponenti. Entitet je predstavljen svojim unikatnim identifikatorom (naziva se ID).

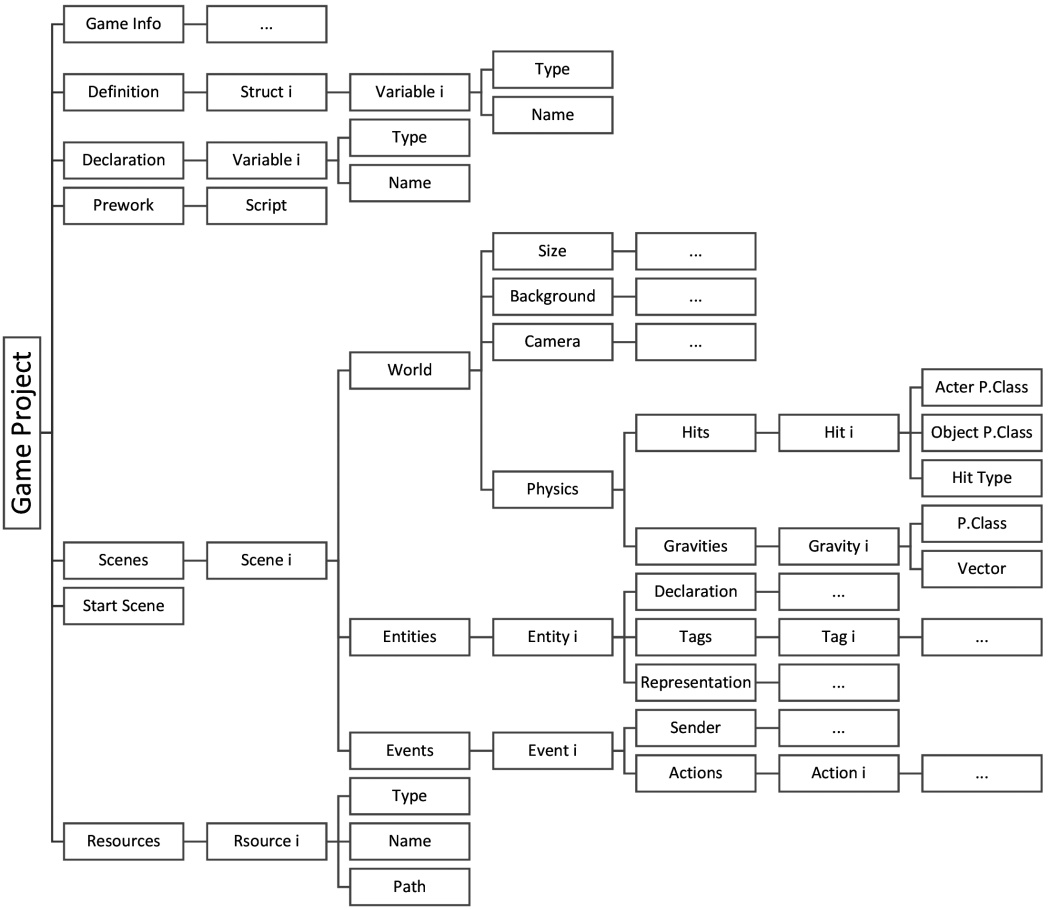
Component. Komponente su osobine entiteta (npr. masa, pozicija, brzina, itd.).

System. Skup svih funkcija koje obrađuju entitete u igri predstavlja sistem. Pri tome funkcije obrađuju samo entitete koje sadrže izabrane komponente.

Primer: Likovi, automobili, kamenje, drveće, svi objekti u igri uopšte, bi bili entiteti. Neke od komponenata entiteta lika bi bile npr. pozicija, brzina i masa. Neke od komponenata kamena bi bile npr. pozicija i masa, a neke od komponenata automobila bi bile pozicija, brzina, ubrzanje itd. Funkcija pomeri(pozicija, brzina) bi bila funkcija koja pomera sve entitete u igri koji imaju komponente pozicije i brzine i kao takva, funkcija pomeri bi bila deo sistema.

Struktura igre

Osmišljena je logička struktura igre sa 48 logičkih celina i napravljen je njen hijerarhijski model. Ovaj model je kompletna definicija projekta i smešta se u projektni XML fajl. Model je predstavljen šemom na slici 1.



Slika 1. Šema modela logičke strukture igre

Figure 1. Scheme of game logic model

U ovoj strukturi može se uočiti globalni deo i kolekcija scena (stanja) u igri. U globalnom delu se nalaze:

- definicije globalnih tipova (slogova) u vidu niza parova (naziv, tip),
- deklaracije globalnih promenljivih kao parova (naziv, tip),
- prework skripta (ono sto se događa nad globalnim promenljivama pre ulaska u glavnu petlju igre),
- definicije potrebnih resursa (slike, zvuci, sprajtovi) kao parova (relativna putanja, tip resursa).

Kolekcija scena je najsloženija podstruktura i svaka scena se sastoji iz 3 dela:

Svet – predstavlja opis sveta scene (pozadina u vidu slike ili boje, veličina sveta i zakoni fizike, kao što su gravitacija i sudari, koji vladaju nad objektima u sceni koji mogu pripadati različitim klasama fizike).

Entiteti – predstavljaju fizički objekat u sceni i obavezno imaju deklaraciju privatnih promenljivih (komponente), tagove (objašnjeno u odeljku „Entiteti, Filteri i ECS") i vizuelnu reprezentaciju (slika, sprajt ili primitivan oblik) u koju su spakovane i fizičke karakteristike objekta (masa i klasa fizike kojoj objekat pripada).

Događaji – predstavljaju elemente dinamike scene i cele igre i definisani su inicijatorom događaja i nizom akcija koje su uslovljene tim događajem.

Primer: Pomeraj miša bi bio inicijator, a akcija koja tada sledi bi bila pomeraj kamere; zajedno bi činili događaj. Sudar dva entiteta sa vizuelnom reprezentacijom (objekta u igri) različitih klasa fizike bi bio inicijator, a akcije koje tada slede bi bile uništenje aktera (jednog od ta dva entiteta) i reprodukovanje zvučnog efekta; zajedno bi činili događaj.

Međutim, ne može bilo koja akcija biti realizovana nakon bilo kog inicijatora. Moguće inicijatore i njima kompatibilne akcije opisuje tablica na slici 2.

Project Manager

Za manipulisanje hijerarhijskom strukturom logičkog modela igre napravljen je modul Project Manager. Ovaj modul predstavlja hijerarhijsku strukturu klasa analognu srukturi igre koja omogućava čitanje, menjanje i pisanje projektnog XML fajla. Glavna klasa ovog modula je Project koja sadrži objekte klasa podstruktura najvišeg nivoa, a ostale klase predstavljaju kontejnere (nizovi i hash mape) objekata klasa nižih hijerarhijskih nivoa. Za manipulaciju samim XML formatom upotrebljena je biblioteka PugiXML. Objekat klase Project je sastavni deo svih daljih procesa izrade igre.

ACTION	EVENT	Entity			Scene		Global	Time	HID	Physics
		Create	Destroy	Custom Condition	Create	Destroy	Custom Condition	Specified Interval	Keyboard & Mouse	Hit
Acter	Destroy									•
	Custom Script	•								
Chunk	Play	•		•	•		•	•	•	•
Song	Play, Pause, Stop	•		•	•		•	•	•	•
Entity	New	•		•	•		•	•	•	•
	Destroy				•		•	•	•	•
	Custom Script									
Global	Custom Script	•	•	•	•	•	•	•	•	•
Sprite	Change Sheet	•	•	•	•	•	•	•	•	•
System	Camera	•	•	•	•		•	•	•	•
	Change Scene	•	•	•			•	•	•	•
	End Game	•	•	•			•	•	•	•

Slika 2. Tablica kompatibilnosti inicijatora i akcija događaja

Figure 2. Event-Action compatibility table

Entiteti, Filteri i ECS

U pomenutoj strukturi entiteti imaju tagove i deklaracije komponenti. Postoje tri vrste događaja i tri vrste akcija koje se odnose na entitete. Da bi bio zadovoljen ECS (analogija: funkcija sistema – događaj ili akcija), uvedeni su filteri u događajima odnosno akcijama. Filter omogućava selekciju konkretnog entiteta na osnovu navedenog taga, komponente ili konkretnog naziva nekog tipa entiteta. Uzmimo proizvoljan tip entiteta za primer. Ako filter ima p tag-ova, sa T_i, $i = 1, \ldots, p$ označene su istinitosne vrednost iskaza koji navodi da izabrani tip entiteta ima tag indeksa i. Ako filter ima q komponenti, sa A_j, $j = 1, \ldots, q$ označene su istinitosne vrednosti iskaza koje tvrde da izabrani tip entiteta ima komponentu indeksa j. Ako filter ima r mogućih naziva tipa entiteta, sa N_k, $k = 1, \ldots, r$ označene su istinitosne vrednosti iskaza koji navodi da izabrani tip entiteta ima naziv indeksa k. Izabrani tip entiteta zadovoljava filter ako je istinit sledeći iskaz:

$$\left(T_1 \wedge T_2 \wedge \ldots \wedge T_p\right) \wedge \left(A_1 \wedge A_2 \wedge \ldots \wedge A_q\right) \wedge$$
$$\wedge \left(N_1 \vee N_2 \vee \ldots \vee N_r\right).$$

Ako je filter zadovoljen, izvršiće se događaj, odnosno akcija.

YANSL i prevodilac

Rečeno je da je cilj generisanje C++ koda igre koji se prosleđuje MinGW kompajleru radi dobijanja gotovog izvršnog fajla igre. Deo generisanog C++ koda će biti i pomenute skripte koje definišu ponašanje objekata u igri. Zbog velikih mogućnosti C++ jezika, javlja se problem velike slobode tih skripti. Takođe, za korisnika koji nije previše vičan programiranju je velika kompli-kacija mnoštvo tipova i oblika petlji koje C++ poseduje. To jednoznačno govori da C++ nije poželjan za upotrebu u funkciji skript jezika u ovom projektu i dovodi do razvoja posebnog skript jezika YANSL (Yet Another Nameless Script Language). Ovaj jezik predstavlja restrikciju C++ jezika prilagođenu mogućem sadržaju skripti u projektu igre i glavna odlika mu je jednostavnost. Restrikcija se vrši zabranjivanjem operatora i ključnih reči C++ jezika koje omogućavaju veliku kompleksnost i slobodu.

Korišćen je leksički parser Lex (FLEX) 3 kako bi se u skripti prepoznali definisani tokeni (ključne reči i operatori C++ jezika) i pomoću toga napisao jednostavan prevodilac. Ulaz za prevodilac je skripta u YANSL-u, a izlaz ekvivalentan C++ kod. Za svaki dobijeni token od Lex-a koji predstavlja zabranjenu ključnu reč ili operator, prevodilac dodaje prefix „ __” tako da se namerno izaziva greška pri pozivu MinGW kompajlera. Greške koje tada daje MinGW prolaze kroz proces obrade i korisniku se dostavljaju sa preciznim opisom skripte u kojoj je greška i pozicijom greške u skripti. Ostali tokeni (koji nisu zabranjeni) se jednostavno i jednoznačno prevode.

Pod pomenutom restrikcijom C++ jezika podrazumeva se sledeće:

- Izbacivanje tipova celih i realnih brojeva većeg ili manjeg kapaciteta i njihovu zamenu tipom „number” koji je 64-bitni double tip. Ostali dozvoljeni tipovi su boolean, char i string.
- Izbacivanje klasa, struktura, pokazivača, svih preprocesorskih direktiva i nizova. U jednom od primera koji su izrađeni korišćenjem ovog rada objašnjeno je na koji način je realizovana alternativa za nizove (primer u odeljku „Demonstracija, rezultati i diskusija“).
- Izbačena je mogućnost pisanja funkcija.
- Umesto 3 vrste petlji, moguće je koristiti samo jednu petlju koja se naziva spin i koja je analogna for petlji i u nju se prevodi.

Biblioteke

C++ predstavlja jezik opšte namene i u standardnoj biblioteci ne sadrži podršku za geometriju, grafiku i medijske funkcije. Pošto je za razvoj igre neophodno imati takve funkcionalnosti, one su u ovaj projekat dodate iz posebnih biblioteka. Neke od tih biblioteka razvijene su posebno za ovaj projekat a neke su nastale proširenjem postojećih biblioteka preuzetih sa interneta. Biblioteke koje su razvijene posebno za ovaj projekat su:

Geo2D. Geo2D je biblioteka za geometriju u 2D igri. Sadrži klase koje predstavljaju vektor, duž, primitivne oblike, poligon i operacije nad njima. Najbitnija klasa – Vector je generička klasa koja ima i generičke operatore kako bi operacije sa vektorima mogle da se izvršavaju nad vektorima različitih generičkih tipova. Sve ostale klase su zasnovane na upotrebi ove klase.

Graph2D. Graph2D je biblioteka za grafiku u 2D igri. Interno koristi SDL (Simple Directmedia Layer – Sam Lantinga) biblioteku i prateće biblioteke (SDL_image, SDL_ttf – Jon Atkins i SDL_bilinear – Mai Mairel) za svu grafiku, a biblioteku SFML (Simple and Fast Multimedia Library – Laurent Gomila) za kontrolu vremena. Za kontrolu vremena prvobitno je korišćen ctime iz standardne biblioteke, ali se SFML pokazao bolje i preciznije u merenju vremena. Bitne klase ove biblioteke su:

- Picture – predstavlja sliku, operacije nad slikom (skaliranje, rotacija, manipulacija pikselima, itd.) i ima podršku za iscrtavanje primitivnih oblika iz Geo2D,
- Graphics – inicijalizuje grafiku i otvara prozor čiji je sadržaj predstavljen klasom Picture,
- Font – od TTF fonta pravi objekat klase Picture tj. tekst pretvara u sliku,
- Timer – meri vreme,
- Sprite – sprajt tj. skup nizova slika koji predstavljaju animacije različitih stanja animiranog objekta u igri (npr. čovečuljak je animirani objekat, a njegova stanja su trčanje, hodanje i skok). Sprajt je fizički smešten u ZIP fajl koji sadrži sliku providne pozadine na kojoj su spakovani pomenuti nizovi slika i konfiguracioni fajl u kome piše koji deo slike predstavlja koji od pomenutih nizova.

Media2D. Media2D je biblioteka za medije – zvuk i video. Ova biblioteka je implementirana samo delom, onoliko koliko je bilo neophodno za razvoj ovog projekta. Trenutno ima klase Song i Chunk koje predstavljaju pesmu i zvučni efekat. Razlika je u tome što se u klasi Song zvuk učitava baferom, a u klasi Chunk se smatra da zvučni fajl može da se ceo smesti u memoriju. U planu je dodavanje klase Video koja će moći da prikazuje video format OGG na objektu klase Graph-2D::Picture.

Physics2D. Physics2D je biblioteka za simulaciju fizičkih sistema u igri. To je uprošćena fizika čvrstih tela sa više vrsta podržanih sudara. Ova bibiloteka bi mogla da se zameni nekom gotovom bibliotekom za simulaciju fizike (npr. Box2D) jer je to manje bitan deo ovog projekta. Klase ove biblioteke su Object i World.

Klasa Object predstavlja objekat u igri i može biti konstruisana od bilo kog primitivnog oblika iz Geo2D i od slike ili sprajta iz Graph2D. Tu su definisani i fizički parametri objekata – masa i vektori pozicije, brzine i ubrzanja. Kolizije objekata se detektuju piksel po piksel u klasi Graph2D::Picture, a da bi se generalizovala ta provera sudara, primitivni objekti se iscrtavaju pa se potom proveravaju sudari. Ovaj vid detekcije kolizije je prilično spor, naročito za geometrijske primitivne objekte, ali se njime postiže opštost i jednostavnost dok se efikasna detekcija kolizije ostavlja kao zasebna tema koja nije deo ovog projekta. Svaki objekat u igri ima definisanu i klasu fizike kojoj pripada. Klase fizike su skupovi objekata u igri za koje važe ista pravila gravitacije i sudara. Sudari mogu biti definisani kao posmatrani i to: elastični (ellastic), propustljivi (pass), telo manje mase nastavlja sa telom veće mase (wb) ili neposmatrani (none). Razlika između posmatranih i neposmatranih sudara je u tome što posmatrani sudari mogu biti inicijatori događaja i svi osim propustljivog utiču na fizičko stanje sveta scene.

Klasa World predstavlja svet igre i sadrži skup objekata u njemu nad kojim može da vrši operacije ažuriranja (pomeranje, povećanje brzine za ubrzanje, detekcija sudara i rekacija na njih, itd.). U ovoj klasi je definisana i kamera koja daje objekat klase Picture tj. sliku dela sveta igre koji se posmatra.

Generator

Generator je modul koji generiše C++ kod igre koristeći objekat klase ProjectManager::Project i jedan template C++ fajl. U jednom prolasku template fajla generator nailazi na komentare koji mu govore da na tom mestu treba da upiše određeni deo koda igre. Takvi su komentari za generisanje globalnih definicija tipova (struktura), globalnih deklaracija promenljivih, za definiciju, alokaciju i oslobađanje resursa, za

postavljanje startne scene i najkompleksnije, za generisanje klasa scena. Ispred svake klase scene se nalaze i klase svih entiteta u toj sceni. Vrlo je važno da se u projektnom XML fajlu scene navode u redosledu suprotnom od redosleda pozivanja, tj. da scene od kojih zavisi neka druga scena budu iznad nje jer je to i redosled generisanja klasa scena. Ako to ne bi bilo ispoštovano, nastala bi greška pri kompajliranju. Uz template fajl se nalaze još 2 para fajlova (header i source) u kojima su definisane bazne apstraktne klase – bazna klasa entiteta i bazna klasa scene.

Generisane klase scena nasleđuju pomenutu baznu klasu scene koja je uključena u template fajlu. Ove klase prate objektni Singleton pattern 4 što omogućava laku implementaciju konačnog automata za promenu scena u glavnoj petlji igre prateći objektni State pattern 5. Bazna klasa scene ima apstraktne metode za ažuriranje i obradu događaja (OnKeyDown, OnKeyUp, OnMouseMove, OnMouseLDown, OnMouseLUp, itd.) sa ulaza koje se pozivaju iz glavne petlje. Promena scene (promena pokazivača na objekat bazne klase scene u State pattern-u) je jedna od mogućih akcija kompatibilna nekim događajima.

Generisane klase entiteta nasleđuju baznu apstraktnu klasu entiteta koja pored metoda za ažuriranje takođe ima apstraktne metode za obradu događaja sa ulaza analogne onima u klasi scene. Prilikom događaja sa tastature, metod za obradu tog događaja u sceni poziva odgovarajući metod za obradu događaja u svakom od svojih entiteta. Ako je neki od događaja ulaza ima akciju koja se odnosi na neki tip entiteta, u klasi tog entiteta u metodu koji nadjačava nasleđeni apstraktni metod za obradu tog događaja će biti izgenerisan potreban kod.

Kako se u konfiguracionom XML fajlu tj. u projektu igre nalaze i skripte pisane u YANSL-u, generator pri dolasku do takvih skripti, poziva eksterni program (pomenuti prevodilac) koristeći ShellExecute (Windows API) i kao ulaz mu prosleđuje YANSL skriptu, a kao izlaz dobija C++ prevod skripte koji piše u izlazni fajl.

Pri kompaliranju generisanog C++ koda MinGW kompajlerom, mogu se javiti greške (pretpostavka je da uzrok tih grešaka može da bude samo u samoj skripti). Radi lakog pronalaženja i otklanjanja greške, korisno je da bude prijavljeno u kojoj skripti i na kojoj liniji se javlja greška. Generator u liniji iznad samog generisanog koda upisuje komentar koji naznačava koja je skripta u pitanju tako što napiše u kojoj je sceni i u kom događaju ili u kojoj akciji kog događaja. Ako je to događaj sa ulaza, piše se na koji uređaj, detaljnu radnju i dugme (po potrebi) se odnosi. Ako je to događaj ili akcija koja se odnosi na entitet, precizira se i detaljan filter, a ako je skripta prework, komentar je „prework".

Demonstracija, rezultati i diskusija

Kao rezultat projekta, generisano je nekoliko igara i u nastavku će biti izložena iskustva stečena prilikom izrade jedne od njih. Diskusija o tim iskustvima je značajna jer pruža uvid u prednosti i mane pristupa na kojem je zasnovan ovaj rad.

PRIMER: Brick Breaker

Glavna scena igre ima entitete brick i ball sa slikom kao vizuelnom reprezentacijom i liner, border i end sa pravougaonikom kao vizuelnom reprezentacijom. Postoje 4 fizičke klase.

Nultu klasu čine border (providni nepopunjeni pravougaonik koji okružuje ceo svet scene) i liner (pločica od koje se lopta odbija).

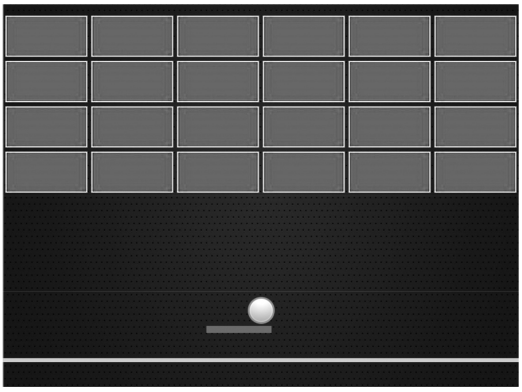
Prvu čini ball (lopta), a sudar između ove i nulte klase fizike je definisan kao elastičan. Kako lopta ima zanemarljivo malu masu u odnosu na entitete nulte klase, entiteti nulte klase ostaju nepomični nakon sudara.

Drugu klasu fizike čine cigle. Sudar između njih i lopte (između ove i prve klase fizike) je takođe elastičan i akcija pri događaju tog sudara je brisanje cigle (Acter:Destroy).

Treću klasu fizike čini samo end (crveni popunjeni pravougaonik u dnu). Tip sudara sa drugom klasom (loptom) nije važan, a akcija na događaj takvog sudara je promena scene na game_over (System:Change Scene) u kojoj se izlazi iz igre. Definisan je događaj pri ispunjenju globalnog uslova (Global:Custom Condition) da je broj preostalih cigli 0, a akcija na takav događaj je promena scene na you_won u kojoj se takođe izlazi iz igre.

Alternativno rešenje problema koji zahteva upotrebu niza

Kako u YANSL-u nema nizova, korišćen je alternativni način generisanja cigli upotrebom događaja. Definisan je događaj čiji je inicijator globalni uslov da je broj cigli (brojač za inicijalizaciju) manji od željenog (24), a akcija je dodavanje cigle u igru (Entity:New). Na taj način, događaj će se izvršiti onoliko puta koliko je potrebno da se scena u igri popuni ciglama. Definisani su i brojači vrste i kolone kao globalne promenljive, tako da događaj kreiranja entiteta (Entity:Create) za filter koji ispunjava cigla ima akciju u vidu skripte (Acter:Script) koja postavlja vektor pozicije i menja brojač vrste i brojač kolone. Ovime se dobija i animacija popunjavanja jer se cigle popunjavaju dinamički, a ne istovremeno (slika 3).



Slika 3. Generisana igra: Brick Breaker

Figure 3. Generated game: Brick Breaker

Zaključak

U generisanoj igri kojom je uspešno obuhvaćen ceo opisani proces, jasno se vidi da je pokazani pojednostavljeni model primenljiv i svrsishodan. Generisana igra je igriva i u njoj nema vidljivih zastoja, seckanja i sličnih problema. Najvažnije je da je projektni XML fajl za ovakvu igru relativno jednostavan i pregledan i korisniku je mnogo jednostavnije da razvija takav projekat u odnosu na programiranje igre od početka. Razvoj igre u okruženju koje je implementirano ovim projektom fokusira se na odnos objekata u igri i opisivanje načina funkcionisanja igre, dok klasično programiranje igara prvenstveno zahteva dobro poznavanje programiranja i programskih biblioteka. U narednom odeljku su istaknute trenutne generalne mane i potencijalni pravci budućeg kretanja projekta.

Nedostaci i dalji razvoj

Grafički korisnički interfejs za editovanje projektnog XML-a

Dodavanje grafičkog korisničkog interfejsa omogućilo bi da se projektni XML edituje u okruženju koje poznaje njegovu semantiku. Grafički interfejs omogućio bi da se većina sadržaja XML-a automatski generiše na osnovu vrednosti koje korisnik unosi u različitim formama za opisivanje igre. Samo bi se YANSL skripte pisale ručno. To znači da bi mesto mogućeg pojavljivanja greške bilo svedeno samo na skripte. Druga prednost GUI-a je što je moguće obraditi imena promenljivih koja se navode u XML-u. Ovo je bitno jer prevodilac koji je implementiran za YANSL pogrešno prepoznaje tokene i unutar celih reči tj. može da prepozna neku od zabranjenih ključnih reči C++ jezika u delu naziva neke promenljive. Na primer ako imamo globalnu promenljivu sa nazivom “redovan“ i istu promenljivu koristimo u nekoj skripti, nakon prevođenja i generisanja C++ koda, deklaracija promenljive će izgledati isto („redovan”) dok će pojavljivanje promenljive u skripti biti izmenjeno („re__dovan“). Razlog leži u prevenciji upotrebe nedozvoljenih ključnih reči koja je prethodno opisana u radu. Implementacija korisničkog interfejsa za editovanje projektnog XML-a omogućila bi da se takav unos spreči.

Grafički korisnički interfejs u igri

Često se u igri javlja potreba za uvođenjem objekata koji predstavljaju elemente korisničkog interfejsa kao što su statični tekst, dugmad, polja za unos teksta i slično. Biblioteka Graph2D koja je već implementirana bila bi uzeta za temelj nove biblioteke GUI2D koja bi pružala opisanu funkcionalnost. Integracija toga u ovakav projekat bi bila jednostavna. Scena bi imala konfi-

guraciju GUI elemenata u njoj koji nisu deo sveta i ne nestaju pri pomeranju kamere, već predstavljaju zaseban i fiksiran deo, ali događaji interakcije korisnika sa ovim elementima bi bili standardni događaji (tablica događaja u odeljku „Struktura igre" bi dobila još jednu grupu događaja i još jednu grupu akcija).

Izrada realnije simulacije fizičkih sistema

Kao što je ranije napomenuto, biblioteka za fiziku (Physics2D) je nesavršena, pa se npr. dešavaju neprirodna odbijanja pri elastičnom sudaru. Vektor brzine objekta nakon sudara ima pravac prave koja prolazi kroz centar minimalnog obuhvatajućeg pravougaonika za taj objekat i kroz tačku koja je aritmetička sredina svih presečnih tačaka kolizije. Prolaženje kroz sve tačke kolizije je sporo, ali bi svakako bilo potrebno za određivanje pravca objekta nakon odbijanja. Ograničenje je i što nema rotacije nakon sudara, pa bi i to bila funkcionalnost koja bi se mogla dodati u realnijoj simulaciji fizičkih sistema. Umesto izrade ovakve biblioteke mogla bi se upotrebiti i neka od gotovih biblioteka za fiziku (npr. Box2D).

Dalji razvoj YANSL-a

Pomenuti vid definisanja skript jezika (restrikcija) je bio najjednostavniji za postizanje cilja, ali dalji rad će se kretati u smeru preciznog definisanja gramatike YANSL-a i pravljenja prevodioca koji će imati suprotan pristup tj. neće posmatrati YANSL kao restrikciju C++ jezika, već kao klasičan jezik definisan od nule koji se po svojim pravilima prevodi u C++.

Literatura

Bilas S. 2002. A Data – Driven Game Object System: http://scottbilas.com/files/2002/gdc_san_jose/game_objects_slides_with_notes.pdf

Chady M. 2009. Theory and Practice of Game Object Component Architecture: http://www.gdcvault.com/play/1911/Theory-and-Practice-of-the

DeLoura A. M. 2000. *Game Programming Gems (Volume 1)*. Rockland: Charles River Media

---

Viktor Slavković

ECS 2D Scriptable Game Maker

Game development usually requires a high level of programming skills and a good knowledge of various libraries used for that purpose (e.g. graphics, sound, physics, keyboard and mouse control libraries). The goal of this project is to develop a simple generalized model for 2D game development. This model would require minimal programming efforts and would bring game development to a level of describing in-game objects interactions. The model promoted by this research is based on Entity Component System (ECS) architecture and is event-driven. This model also contains a pre-modeled hierarchy of logical units that is used as a base for 2D game development.

At the first level of the base model are global variables, configuration script, resources and scenes used in a game. Scenes are the most complicated substructure and each one consists of world definition (background, physics, and camera), entities and events. Entities are in-game objects which can be visualized as pictures, sprites or geometric primitives. The purpose of the events in this structure is to describe relations between in-game objects.

Each event has its initiator and its action queue. There are several types of event initiators and several types of actions, depending on the event type. If an event describes a complex relation between objects, it can be written as a script in YANSL (Yet Another Nameless Script Language). This script language was specially designed for this project and is a restriction of C++ language, in order to be as simple as possible. It is translated to C++ code, rather than compiled.

The whole structure of the game is stored in a XML project file since XML format is a natural container of hierarchical structures. The module "Project Manager" has been developed for the purpose of reading, editing and saving this pro-

ject file. When the project XML is done it is being passed to the Generator module. It first translates YANSL scripts to C++ and then populates the pre-made C++ template with the code snippets from the project. This way, the game is now fully written in the C++ language. Finally, the generated code is passed to the MinGW (Minimalist GNU for Windows) compiler (deployed with the project) and linked against custom written libraries for physics, graphics, geometry and media. The result is a fully functional executable version of the game.

Several different 2D games were developed using tools made in this research. XML projects used to develop those games show that the goal has been achieved – all the programming has been limited to the scripts describing in-game objects behavior and scripting the game scenarios.

Future directions of development are:

- Editor GUI (semantically-aware XML project editing),
- In-game GUI elements (buttons, text boxes, list views etc.),
- More realistic physics engine (for example: integration of Box2D library),
- YANSL compiler instead of YANSL to C++ translator.