SVEUČILIŠTE U ZAGREBU

FAKULTET ELEKTROTEHNIKE I RAČUNARSTVA

ZAVRŠNI RAD br. 2323

Make Me Move -

didaktička računalna igra za učenje strukturne molekularne biologije

Igor Jerković

Zagreb, lipanj 2012.

Zahvaljujem Ivani Mihalek i Mili Šikiću na korisnim savjetima.

Sadržaj

1. Uvod 1

2. Make Me Move – didaktička računalna igra za učenje strukturne molekularne biologije 2

2.1. Opis problema 2

2.2. Edukativne računalne igre 2

2.3. Strukturna molekularna biologija u igri 3

2.4. Sastav igre 4

2.4.1. Nulta i prva razina 4

2.4.2. Druga razina 5

2.4.3. Treća i četvrta razina 5

3. Implementacija 7

3.1. iPhone aplikacija 7

3.1.1. iPhone igra 7

3.2. Objective C 8

3.3. Cocos2D 11

3.3.1. Sprite i grafičke performanse 12

3.3.2. Scene 17

3.3.3. Akcije i animacije 21

3.4. Organizacija igre MakeMeMove 22

4. Rezultati 24

5. Zaključak 27

6. Literatura 28

1. Uvod

Učenje kroz igru je jedan od efikasnijih načina usvajanja novih znanja. Pogotovo ako su igre napravljene na zanimljiv i jednostavan način kojim igrači lakše usvoje činjenice vezane za igru. Zadnjih desetak godina su ulaganja i napredak na tržištu računalnih igara bila ogromna, a ovo područje uvijek pomiče granice trenutno limitiranog sklopovlja kako bi igre bile bolje i realističnije. Razvijeno je i mnogo igara edukativnog karaktera u vidu pitalica, kvizova i simulacija s ciljem pristupačnijeg prenošenja novog znanja. Zbog pozitivnog utjecaja edukativnih računalnih igara u ovom radu će biti opisano stvaranje didaktičke računalne igre za učenje strukturne molekularne biologije – *Make Me Move*.

U radu će se pratiti razne faze nastanka edukativne računalne igre *Make Me Move*, od odabira samog programskog jezika u kojem će se igra implementirati, odabira radnog okvira prilagođenog razvijanju računalnih igara, odabira platforme, dizajniranja programa pod objektno orijentiranom paradigmom do samog igranja igre temeljene na molekularnoj biologiji.

Molekularna biologija teži razumijevanju i istraživanju interakcija između raličitih staničnih sustava, uključujući složene međuodnose DNK, RNK i proteinske sinteze te učenju kako su te interakcije regulirane. Pri tome valja istaknuti veliku međuvezu moderne molekularne biologije i računarskih znanosti, pogotovo u vidu bioinformatike. Uz bioinformatiku nekadašnji problemi prevelikih vremena za pojedina izračunavanja su znatno umanjena te je računarska znanost znatno doprijenela u rješavanju i otkrivanju odgovora na mnoga pitanja molekularne biologije. Ovaj rad je isto tako primjer interdisciplinarnosti ova dva područja, iako ne u istraživačkom nego edukativnom smislu.

2. Make Me Move – didaktička računalna igra za učenje strukturne molekularne biologije

2.1. Opis problema

Poznato je da igrači računalnih igara često pamte činjenice vezane uz igru. Cilj rada je primjeniti to saznanje u edukativne svrhe, Zadatak je razviti didaktičku aplikaciju koja igraču pomaže da savlada činjenice o kompliciranom biomolekularnom sustavu. Kao primjer sustava koristi se uloga aktina u pokretljivosti stanice. Potrebno je simulirati molekularne interakcije u stanici i njihov utjecaj na njeno kretanje. Korisnik igre treba ponoviti točan niz interakcija kako bi uspješno pokrenuo stanicu. Sve interakcije imaju poveznice na znanstvenu literaturu koja ih opisuje. Igra je razvijena za iPhone u radnom okviru cocos2D.

2.2. Edukativne računalne igre

Računalne igre edukativnog karaktera su dobar način usvajanja novih znanja. Ono što čini računalne igre dobrima kao sredstva pri učenju ima zapravo malo ili nikakve veze sa činjenicom što su to igre. Zapravo, igre koje su najzanimljivije za učenje nisu bilo kakve igre, razni tipovi igara mogu imati različite efekte. Igre u stilu *Tetrisa* mogu poboljšati igračeve sposobnosti u prepoznavnju uzoraka, igre poput *Učilice* mogu pomoći u učenju nekih činjenica. No, ovo vjerojatno nisu najzanimljivije računalne igre što se tiče učenja.

Vjerojatno najzanimljivije računalne igre su digitalne simulacije raznih svijetova u kojima igrač ima svog surogata (virtualnog predstavnika sebe u igri) kroz kojega može utjecati u tom virtualnom svijetu kroz složene simulacije i velik broj parametara. Igra kao što je *Civilization* u kojima je igrač zapravo dio kompleksnog sustava i može učiti duboke konceptualne principe o povijesti i društvenim znanostima su veoma zanimljive jer je igrač zapravo smještan u samoj simulaciji koja ima ogroman broj parametara. Odlična stvar kod simulacija je što se mogu prvo istestirati posljedice koje slijede iz akcija prije nego se pokuša nešto slično u stvarnom svijetu. Ispostavlja se da ovakve računalne igre u kojima se odvijaju složene simulacije mogu biti i dobra simulacija za ljudski um. Naime, za čovjeka je

efektivno razmišljanje često puta više simulacija složenih situacija u umu gdje se čovjek zamišlja kao surogatni lik, nego što su to apstraktne generalizacije. Čovjek zapravo najbolje uči kada se stavi u određenu situaciju, a igre u kojima se odvijaju složene simulacije su upravo odličan primjer toga. Iako su komercijalne računalne igre u kojima postoje ovakve simulacije složenih svijetova najčešće ratnog tipa u kojima su glavni likovi vojnici ili razbojnici, ne postoji razlog zašto se takvo nešto ne bi moglo primjeniti i u znanstvene svrhe. U dobrim računalnim igrama igrač ima priliku osim nekih zadanih ciljeva rješavati i ciljeve u smjeru u kojem to on smatra najbolje. Primjerice u igri *Grand Theft Auto* igrač može birati hoće li se ponašati kao zločinac ili dobrotvorac, može rješavati zadatke raznim poretkom, može uskočiti u vozilo hitno pomoći i pomagati ljudima, a može i započinjati ratove mafija, sve ovisi kako igrač razvije svog virtualnog surogata. Ovo može biti i dobar koncept za znanstvene igre (simulacije nekih složenih procesa), u kojima igrač također može stvarati drukčije rezultate ovisno o svojim postupcima. Računalne igre su između ostaloga dobre jer mogu kreirati virutalnu reprezentaciju igrača u kompleksnom simuliranom svijetu, mogu poticati na razvoj inteligencije, mogu razvijati poticaje za nove ideje i ono što je možda najvažnije, dopuštaju igraču da se stavi u kontekst tog složenog svijeta.

Igra *Make Me Move* nije toliko razvijena simulacija kompleksnog sustava, ali ima potencijala za razvoj u smjeru dobre simulacije procesa u molekularnoj biologiji.

2.3. Strukturna molekularna biologija u igri

Najmanji objekti koje ljudsko oko može zapaziti su reda veličine 0.1mm. To znači da u određenim uvjetima čovjek može vidjeti bez ikakvih optičkih pomagala primjerice amebu ili papučicu. Manje stanice su lako vidljive slabijim mikroskopom. Čak je moguće zapaziti strukture unutar stanice kao što su nukleus, mitohondriji ili kloroplasti. Što idemo prema jačim mikroskopima dobivamo uvid u sve složeniju strukturu stanice.

Igra je posložena na zanimljiv način u analogiji sa prethodnim primjerom zamjećivanja sve detaljnije strukture stanice i njenih dijelova. Naime zadatak ovog

rada je napraviti dvije razine, gdje razine predstavaljaju stupanj detaljnog opisa stanice, tj. uvećanja.

Cilj same igre je predočiti igraču ulogu aktina u pokretljivosti stanice. Monomeri proteina aktina se slažu u filamente koji guraju stanicu prema naprijed, a također mogu pomoću u međustaničnom transportu raznih patogena i mjehurića (engl. vesicle), ali i raznih ostalih struktura kao što su primjerice liposomi. Kompleks formiran od aktinski vezanih proteina 2 i 3 (Arp2/3) je nukleator aktinske polimerizacije, koja uzrokuje grananje filamenata. Svi ovi dijelovi stanice će biti korišteni u simulaciji pokretanja stanice u igri.

2.4. Sastav igre

Igra je raspodijeljena na dvije razine u kojima svaka predstavlja veće uvećanje i detaljniji prikaz stanice. Igra je napravljena tako da ima prostora za nadogradnju, no o tome će više biti govora kasnije.

2.4.1. Nulta i prva razina

Na prvoj razini igre koje se simbolično zove *Zoom level 0*, opisana je jako gruba skica stanice, bez ikakvih uvećanja, u kojem je potrebno simulirati kretanje stanice pravilnim izborom koraka: dodavanje lamelopodijuma, dodavanje točke adhezije u željenom smjeru kretanja, uklanjanjem točke adhezije sa prvotne pozicije, te ponovnim dodavanjem točke adhezije negdje u pozadinu stanice, tj. u smjeru iz kojega se stanica giba. Nakon toga prelazi se na jednu dinamičniju scenu u kojoj bakterija treba uloviti svoju hranu, u ovom slučaju neutrofil, dok igrač upravlja bakterijom zadavajući bakteriji smjer kretanja, na što se u željenom smjeru iscrtavaju lamelopodij i točke adhezije koje se pojavljuju i iščezavaju, kako se bakterija miče prema cilju. Kako bi ova razina bila zanimljivija, potezi igrača se boduju. Točan odgovor na ispravan poredak u micanju stanice donosi bodove, dok netočan odabir sljedećeg koraka oduzima dio dosadašnjih bodova i tjera igrača da igra pažljivije. U dijelu u kojem igrač upravlja bakterijom, cilj je što prije uhvatiti neutrofil jer svakom sekundom koja protekne oduzme se određeni dio dosadašnjih bodova.

2.4.2. Druga razina

Druga razina igre se naziva *Zoom level 10*, jer se na ovoj razini već može vidjeti detaljniji izgled stanice. Konkretno, na ovoj razini igrač nauči ulogu aktina u pokretljivosti stanice i kretanju stanice u smjeru podražaja, tj. u smjeru gdje se nalazi hrana. Igrač treba ispravnim redoslijedom birati dijelove stanice ponuđene u desnom izborniku kako bi uspio natjerati stanicu da prihvati podražaj i pokrene se u smjeru podražaja. Dijelovi stanice koje korisnik mora ispravnim redoslijedom odabrati i na točnu poziciju u stanici smjestiti su:

- receptor
- ARP 2/3
- adhezijski kompleks
- *cofilin*
- *vesicle*

Za svaki točan odabir dijela stanice iz desnog izbornika i klik na točnu poziciju te smještanje u stanici, igrač biva nagrađen bodovima i napreduje u cilju pokretanja stanice u smjeru podražaja. Dodatno, igrač treba otkriti kako između drugog i trećeg koraka treba dodati *cofilin* na odgovarajuću poziciju u stanici kako mu mjerilo razine aktina ne bi palo u nedozvoljene vrijednosti, te tako igrač gubi. Ukoliko igrač ne primjeni ovaj korak igra mu biva znatno teža jer će razina aktina brzo opasti i igra će završiti gubitkom. Radi postizanja još bolje mogućnosti učenja iz ove igre u svakom trenu je moguće zaustaviti igru i klikom na pojedini dio stanice dobiti popis poveznica na znanstvenu literaturu vezanu za taj dio stanice.

2.4.3. Treća i četvrta razina

Igra je napravljena tako da ima mogućnosti za proširenja i na veće razine povećanja gdje se približavamo sve manjim mikroskopskim veličinama. Tako je moguće u daljnjem razvoju napraviti *Zoom Level 100* i *Zoom Level 1000*, gdje bi *prava* molekularna biologija mogla doći na vidjelo. Cilj ovog rada je bio napraviti upravo te prve dvije razine, no i izgraditi igru sa inženjerske strane tako da bude

nadogradiva za treću i četvrtu razinu. U nastavku ćemo opširnije vidjeti o samom tijeku izrade aplikacije, o jeziku te radnom okviru u kojem je igra razvijena.

3. Implementacija

Slijedi nekoliko potpoglavlja u kojima se opisuje jezik u kojem je napravljena igra, uređaj za koji je izrađena igra, radni okvir koji je korišten pri izradi igre, te sam dizajn programske potpore kroz opis pojedinih klasa.

3.1. iPhone aplikacija

Pametni mobiteli polako preplavljuju svijet, a sa sve većim tržištem javlja se i sve veći broj aplikacija bez kojih pametni mobitel nije pametan. Što se tiče konkretno iOS uređaja, kada se lansira jedna aplikacija preko 250 milijuna ljudi sa iOS uređajima je u mogućnosti skinuti i koristiti tu aplikaciju. Ovdje se nalazi velik potencijal; zanimljiva i jedinstvena aplikacija bez obzira bila ona sportskog, političkog, glazbenog ili edukativnog karaktera može brzo postati proširena na velikom broj uređaja i korisnika. Potencijal ove, ali i drugih (npr. Android) platformi je u usponu i stoga je za platformu na kojoj će se računalna igra edukativnog karaktera o struturnoj molekularnoj biologiji razvijati odabrana iOS platforma, točnije za uređaj na kojem igra radi odabran je iPhone. Mobilna platforma je odabrana jer se tako aplikacija može koristiti praktički svugdje, mobitel ljudi mogu nositi sa sobom, puno su fleksibilniji nego sa recimo prijenosnim računalom i imaju širi spektar vremena za iskoristiti u korištenje aplikacije što je upravo i glavni razlog za odabir mobilne platforme u implementaciji igre.

3.1.1. iPhone igra

Razvijanje igara za iPhone je poprilično uzbudljivo i zabavno, to je jedna od rijetkih stvari koje ljudi mogu raditi a da bi se osjećali ponovno kao djeca s time da u ovom slučaju oni razvijaju igre za neku drugu djecu, kao što je nekad dok su oni bili djeca netko osmislio njihovu omiljenu igru.

Izrada igara pokriva neočekivano širok spektar područja računarske znanosti, ovdje su spomenute samo neke najvažnije:

- Grafika
- Glazba i zvukovni efekti

- Mrežno programiranje
- Objektno orijentirano programiranje

Srećom razvijeni su mnogi radni okviri (engl. framework) koji pomažu u bržoj izradi projekta. Tako je za izradu ovog rada korišten radni okvir cocos2D, koji je inače jedan od najpopularnijih za razvoj iPhone igara. Programsko okruženje (eng. IDE – integrated development environment) u kojem je projekt razvijan je Xcode, a sam jezik u kojem je igra implementirana je Objective C. U nastavku slijedi malo više o specifičnostima jezika i radnog okvira u kojem je igra razvijena.

3.2. Objective C

Objective C je objektivno orijentiran programski jezik koji se danas primarno koristi na Apple-ovoj Mac OS X i iOS platformi. Objective C je razinu iznad programskog jezika C, koji je strogi podskup jezika Objective C, tj. svaki program napisan u programskom jeziku C moguće je prevesti i prevoditeljem za programski jezike Objective C, i slobodno se svaki kod napisan u C-u može koristiti i kod pisanja Objective C koda. Objective C-ov model objektno orijentiranog programiranja temelji se na slanju poruka instancama objekta. U ovom jeziku ne pozivaju se jednostavno metode, nego se šalju poruke. Razlika između ta dva koncepta je u tome kako kod koji je referenciran imenom metode ili poruke biva izvršen. U Objective C-u ciljani objekt koji prima poruku se razrješava za vrijeme izvršavanja programa (engl. dynamic binding), s time da sam objekt interpretira dobivenu poruku. Posljedica toga je da sustav koji funkcionira na principu slanja poruka nema provjeru tipova. Objektu koji prima poruku, tj. primatelju se ne garantira da će znati odgovoriti na dobivenu poruku, i ako ne zna jednostavno stvori iznimku.

Primjer poziva metode u programskom jeziku C++:

```
obj->method(argument);
```

Primjer slanja poruke u programskom jeziku Objective C:

```
[obj method:argument];
```

Objective C zahtjeva da se sučelje i imeplementacija nalaze u dvije razdvojene datoteke. Sučelje se stavlja u tzv. header datoteku, a sam kod ide u

implementacijsku datoteku. Header datoteka ima sufiks .h, dok imeplementacijska datoteka ima sufiks .m. Ovako bi na primjer izgledali sučelje i implementacija u Objective C-u:

```
@interface classname : superclassname {
//instance variables
}
+classMethod1;
+(return_type)classMethod2;
+(return_type)classMethod3:(param1_type)param1_varName;
-(return_type)instanceMethod1      :(param1_type)param1_varName
                                   : (param2_type)param2_varName;
-(return_type)instanceMethod2WithParameter:
(param1_type)param1_varName                    andAnotherParameter:
(param2_type)param2_varName;
```

Povratni tipovi mogu biti bilo koji standardni C-ov tip, pokazivač na Objective C objekt, ili pokazivač na neki specifični tip objekta kao što je recimo NSImage* ili NSString*. Podrazumijevani povratni tip je id.

Sam jezik i metode, tj poruke su pisane tako da ako se čitaju s lijeva na desno, odaju osjećaj kao da se kod čita na engleskom jeziku. Iz tog razloga metode nisu kao u većini jezika gdje se prvo navede ime metode pa u zagradama argumenti, nego su argumenti unutar metode između dijelova imena razdvojeni dvotočkama.

```
-(void)setRangeStart: (int)start andEnd: (int)end;

-(void)shootSpaceship: (Spaceship *)aSpaceship atPosition: (CGPoint)aPosition;
```

Sučelja, odnosno .h datoteke nemaju nikakav kod zapravo, pa pogledajmo primjer jedne .m datoteke, tj. implementacijske datoteke:

```
@implementation classname
+ (id)classMethod
{
      return [Bacteria alloc];
}
```

```
-(Bacteria *)instanceMethod {
    Bacteria *greenBacteria = [[Bacteria alloc] initWithSize: CGSizeMake( 200, 300)
                                                    andColor: [UIColor greenColor]];
    // do some calculations
    return greenBacteria;
}
```

Jednom kad smo napravili Objective C klasu možemo ju i instancirati. Prvo treba alocirati neinicijaliziranu instancu klase te potom inicijalizirati taj objekt. I obično se ta dva koraka rade u jednoj liniji, odmah jedno iza drugog.

```
MyObject *o1 = [[MyObject alloc] init];
MyObject *o2 = [[MyObject alloc] initWithString:myString];
```

Objective C podržava dinamičko povezivanje, na primjer ukoliko pošaljemo neku poruku kolekciji raznih objekata, od kojih samo neki znaju odgovoriti, tj. primiti poruku, nema straha od stvaranja greške tijekom samog izvođenja programa. Također objekt ne mora biti definiran za vrijeme prevođenja, a i dalje mu možemo poslati poruku.

Važno je napomenuti da Objective C nema automatsko oslobađanje neiskorištene zauzete memorije (engl. garbbage collector) tako da svu memoriju koju se zauzme treba i osloboditi jer u suprotnom se događa curenje memorije, a treba imati na umu da je ovo razvoj aplikacije za mobitele, koji su prije svega jako ograničeni uređaji i memorijom i baterijom. Stoga je uvijek važno posebno paziti na memoriju, ali i pri korištenju nekih naprednijih, zahtjevnijih funkcionalnosti. Uređaji kao što je primjerice GPS ili kamera, ukoliko se pristupa takvim komponentama potrebno je paziti i na memoriju jer na kraju je najvažniji dojam koji aplikacija ostavlja na korisnika. Aplikacije koje prebrzo troše bateriju ili se učestalo ruše (a ruše se najčešće zbog curenja memorije) bit će pregažene od strane konkurencije, a i sami korisnici će brzo odbaciti takve aplikacije jer ih nije ugodno koristiti.

3.3. Cocos2D

Kao što je već spomenuto razvoj računalnih igara zahtjeva znanja i vještine iz više područja računarske znanosti a neke od važnijih su grafika, glazba, umrežavanje itd. No zahvaljujući ovom radnom okviru nije potrebno gledati i proučavati primjerice OpenGL ES, OpenAL i ostala niskorazinska aplikacijska sučelja (engl. API) pri izradi igre. Dakako ako je potrebna izrazita optimiziranost uvijek je bolje koristiti što nižu razinu jer će brže raditi. No cocos2D je već dugo razvijan i jako je optimiziran, tako da se i već dosta složenije igre mogu bez problema razvijati u ovom radnom okviru. A što se tiče igre koja je razvijena u okviru ovog rada cocos2D je bio i više nego dovoljan.

Cocos2D je zapravo Objective C radni okvir (engl. framework) otvorenog koda (engl. open-source) izrazito prilagođen stvaranju 2d igara za iOS platformu (iPhone, iPad i iPod touch) te za Apple-ova računala Mac OS X. Cocos2D koristi OpenGL ES za grafičko renderiranje, pružajući svu brzinu grafičkog procesora (GPU) uređaja na kojemu se igra izvršava. Najvažnija stvar zbog koje je Cocos2D popularan je činjenica da omogućuje programeru da posveti više vremena na stvaranje igre, a ne na proučavanje niskorazinskih biblioteka. Ovo je posebno važno jer je AppStore tržište vrlo fluidno i razvija se jako brzo. Izrada protipa i što prije kretanje u razvoj igre može biti ključ u pobjeđivanju konkurencije jer je iOS tržište visoko kompetitivno mjesto, a ne trošiti previše vremena kopajući po OpenGL ES ili OpenAL bibliotekama.

Cocos2D također ima mnogo ugrađenih klasa koje olakšavaju razvijanje igara, kao što je recimo TextureCache, koji automatski cach-ira grafiku, pružajući brži i lakši osjećaj kako stvari funkcioniraju na ekranu kod samog korisnika. TextureCache se vrti u pozadini i nije potrebno čak ni znati kako se koristi jer funkcije ove klase su potpuno transparentne prema programeru. Još neke od korisnih ugrađenih stvari su renderiranje fonta, sprite sheet-ova, robustnih audio sustava itd. Ovaj radni okvir je i odličan alat za stvaranje prototipova, te je moguće relativno brzo izgraditi jednostavnu igru.

Jedna od najjačih sastavnica Cocos2D-a su akcije. Akcije omogućuju pomicanje, skaliranje i manipuliranje sprite-ovima i ostalim objektima velikom

lakoćom. Kao primjer navedeno je kako pomaknuti jedan sprite, u ovom slučaju bakteriju za 200 piksela u desno kroz vrijeme od 3 sekunde.

```
CCAction *moveAction = [CCMoveBy actionWithDuration:5.0f position:CGPointMake( 400.0f, 0.0f )]
[bacteria runAction:moveAction];
```

To je sve što je potrebno za napraviti kako bi jedan objekt u igri bio animiran, u ovom slučaju pomaknut je na neku poziciju na ekranu. No postoji još i niz raznih mogućih akcija u Cocos2D koje su već ugrađene kao što su rotiranje, skaliranje, skakanje, blinkanje, zamagljivanje, i mnoge druge vrste animiranja koje olakšavaju posao i stvaraju igre živahnijim i u krajnjem cilju zanimljivim. Također je sadržan i impresivan sustav čestica kojima je lako dodati atraktivne efekte dima, vatre, kiše ili snijega u igru. Ugrađeni su i razni efekti pri prijelazima među raznim scenama u igri. Kao što je već spomenuto Cocos2D omogućuje lako korištenje biblioteke OpenAL bez potrebe učenja korištenja same biblioteke, tako da se pozadinska melodija ili bilo kakvi zvučni efekti mogu dodati praktički u jednoj liniji koda. Ugrađeno je i parallax skrolanje, čime se dobiva iluzija 3D dubine i 3D perspektive što može učiniti igru atraktivnijom. U nastavku će biti opisani neki važniji termini vezani za Cocos2D, ali ne samo uz taj radni okvir nego i općenito uz razvoj igara.

3.3.1. Sprite i grafičke performanse

U industriji razvoja igara pojam sprite se spominje svugdje. Sprite je slika koja se može kretati po ekranu neovisno o drugim slikama. Sprite može biti neki glavni heroj igre, ogromna slika pozadine u igri ili recimo u ovom radu dio stanice ili bakterija koja lovi hranu okolo po ekranu. U praksi su sprite-ovi najčešće png ili pvrtc slikovne datoteke. Jednom kad se učita u memoriju, sprite se konvertira u teksturu koja je posebno prilagođena iPhone-ovom grafičkom procesoru za renderiranje prikaza na ekran. Dakle jednom kad se sprite koji je zapravo obična slika konvertira u teksturu, učitava se u memoriju uređaja i sprema se kao nekompresirana tekstura. Zatim se ta datoteka kompresira i konvertira u format koji iPhone-ov grafički procesor može razumjeti i učitava se u radnu memoriju prije nego li se mogu koristiti. Učitana slika u radnoj memoriji je ono što se naziva teksturom. Grafički procesor nativno podržava nekoliko kompresiranih formata kao

što je na primjer pvrtc. Datoteke ostalih formata se moraju spremiti kao nekompresirani slikovni podaci. To je ono što OpenGL ES crta na ekran. Dekompresija slike je razlog zašto je sprite tj. png slika možda mala u flash memoriji uređaja ali u memoriji zauzima puno više prostora jer se dekompresira. Kako bi se uštedilo na memoriji i reducirala količina neiskorištenog prostora u teksturama, prikladno je puno tih malih sprite-ova, sličica glavnog lika igre izkombinirati i spremiti u jednu veliku teksturu koja se naziva teksturni atlas (engl. texture atlas). Teksturni atlas je jednostavno jedna velika tekstura, koja sadržava skupinu sličica koje se koriste u igri, a iz njega se lako mogu izrezati pojedini dijelovi tj. manje sličice koje su spremljene u atlas. Najjednostavnije je možda zamisliti jedan album u kojem su sve slike zalijepljene jedna pokraj druge. Iz albuma je vrlo jednostavno izrezati pojedinu fotografiju sa velike stranice albuma. Jedna od ključnih stvari pri postizanju boljih grafičkih performansi je predavanje OpenGL ES-u što manje tekstura, tako da se u jednoj teksturi zalijepi više sličica kombiniranih u teksturni atlas. Sve ovo je bitno jer je ključ u stvaranju skupine poziva (engl. batch calls) iscrtavanja jer su iscrtavanja slika ne ekran zapravo najskuplje operacije.

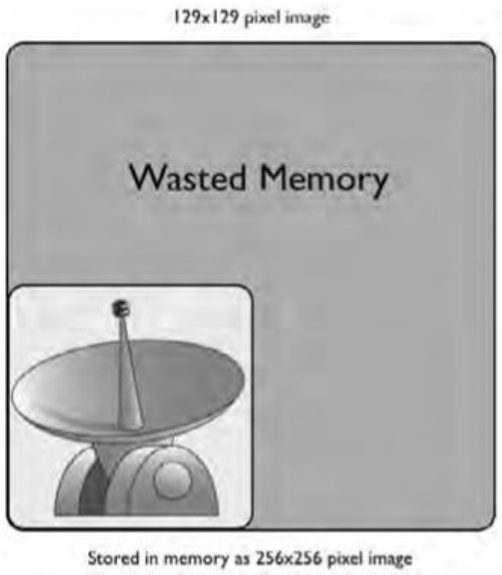
U mnogo stvari u životu postoje brojna ograničenja, tako je i u razvoju računalnih igara. Ustvari, razvoj računalnih igara uvijek traži najviše od limitiranog sklopovlja (engl. hardware) koje se trenutno nalazi na tržištu. Česti problem s kojim se programeri računalnih igara susreću je loša performansa kada se na ekranu nalazi velik broj sprite-ova. Čak i kada bi se maknula sva logika igre i samo ostavilo puno sprite-ova na ekranu da se kreću - performanse će biti i dalje loše što ukazuje koliko je zahtjevnije stalno iscrtavanje slikovnih elemenata po ekranu. Može biti više razlog lošim performansama, no jedan od najčešćih je prevelik broj izmjena konteksta tekstura zbog korištenja pojedinačnih sprite-ova, a ne skupine sprite-ova od jednom. U cocos2D bi ekvivalent bile klase CCSprite i CCSpriteBatchNode. Što se ustvari događa je previše OpenGL ES poziva za koje je već navedeno da su skupe operacije, a kada su sve slike razdvojene u zasebnim teksturama znatno je više posla i memorije nego što grafički procesor može obraditi u vremenu za vrijeme trajanja jednog okvira (engl. fps – frame per second - označava koliko često se cijeli ekran osvježi u jednoj sekundi). Naravno jedno od najjednostavnijih rješenja je korištenje teksturnog atlasa. Ukoliko se

detaljnije promatra klasa CCSprite u Cocos2D-u može se primjetiti da se metoda –(void)draw poziva na svako osvježenje zaslona (na svaki frame). Unutar te metode nalaze se zapravo OpenGL ES pozivi za crtanje sprite-a na ekran. Za svaki sprite, OpenGL ES se mora povezati sa teksturom tog sprite-a i onda nacrtati (ovaj korak se zove renderiranje) sprite na ekran. Postoji još jedan korak prije nego li se piksel pojavi na ekranu: OpenGL ES driver samog iOS operacijskog sustava konvertira OpenGL ES pozive u strojni jezik kako bi se mogle izvoditi operacije na grafičkom procesoru. Svaki OpenGL ES poziv ima cijenu nekoliko CPU ciklusa za OpenGL ES driver. Stoga će se igra izvršavati brže ukoliko se renderiranje izvršava sa što manje OpenGL ES poziva.

Evo primjera za vizualizirati sve ovo. Zamislite sebe kao da obavljate ulogu OpenGL ES-a i računalna igra traži od vas da nacrtate scenu plaže. Scena plaže ima pijesak, more i suncobrane. Ako igra koristi samo CCSprite-ove, tada vam prvo mora predati sliku za pijesak i pitati vas da nacrtate tu sliku, zatim dati sliku mora i pitati vas da ju nacrtate. Ovako bi se nastavilo dok se sve slike ne bi renderirale na ekranu što ubrzo postaje naporan i repetitivan posao, te se počinjete pitati zašto vam igra ne može dati odjednom sve slike. I na kraju ispada da se mogu teksture svih slika poslati OpenGL ES-u odjednom i način na koji se to radi u Cocos2d-u je CCSpriteBatchNode sa teksturnim atlasom.

CCSpriteBatchNode je posebna klasa u Cocos2D-u koja se može ponašati kao roditelj omotaču CCSprite-a. Nalazi se između podloge igre (engl. layer) i samih CCSprite-ova. Kao što se da naslutiti iz samog imena, ukoliko je korišten CCSpriteBatchNode, tada su svi pozivi za iscrtavanje CCSprite-ova ugrupirani i poslani odjednom zajedno. No velika ušteda i dobitak na performansama ne postiže se samo skupljanjem poziva na jednu hrpu nego i kombiniranjem manjih tekstura u jednu veliku teksturu s kojom se OpenGL ES onda mora povezati samo jednom. Ovo je znatna ušteda – svi pozivi povezivanja OpenGL ES-a sa teksturama svedeni su na jedan poziv. Primjerice ukoliko je 50 CCSprite-ova unutar CCSpriteBatchNode-a tada je potrebno samo jedno povezivanje OpenGL ES-a umjesto njih 50. Velika tekstura se naziva teksturni atlas (engl. texture atlas ili sprite sheet) i to je jednostavno tekstura dovoljno velika da u nju stanu sve manje teksture. Potrebno je prisjetiti se primjera sa plažom - to bi bio veliki papir sa kojeg bi se mogli izrezivati teksture pijeska, mora i suncobrana.

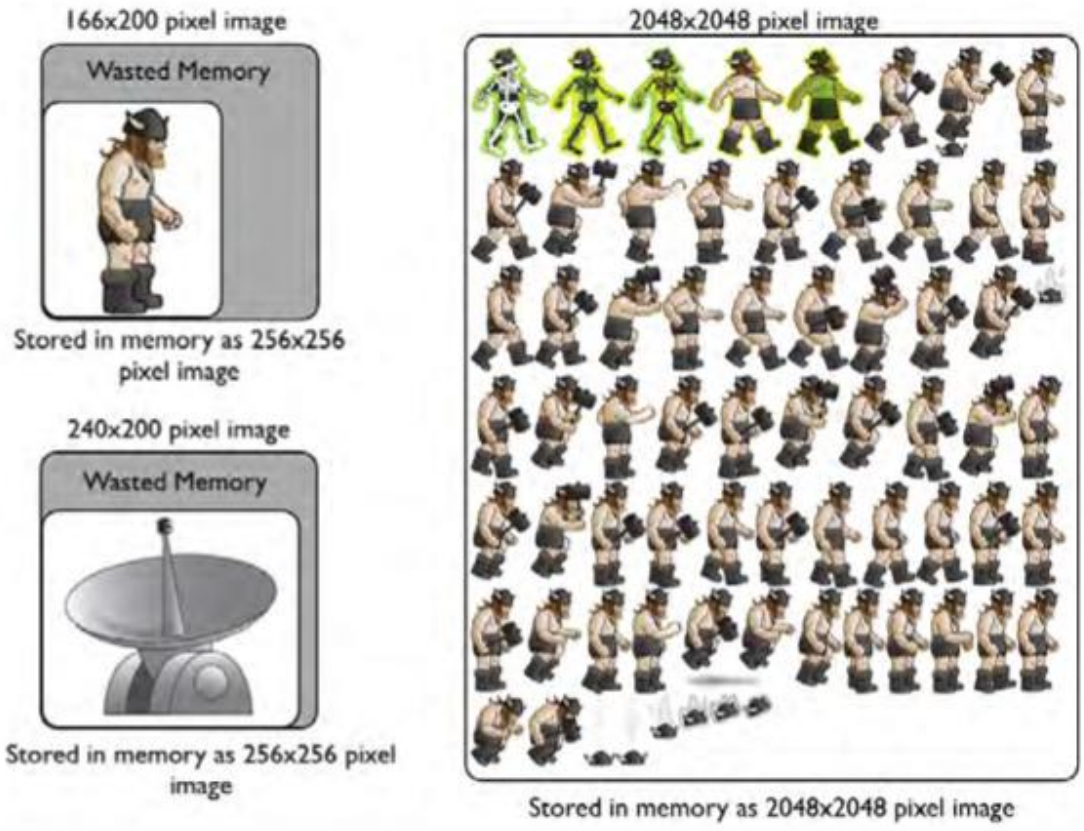
Memorija je izuzetno skup resurs na uređaju kao što je iPhone. Što se manje memorije zauzima to je bolja performansa igre. iPhone ima dijeljenu memoriju što znači da grafički procesor nema vlastitu memoriju, nego GPU koristi memoriju glavnog sistema za spremanje tekstura i geometrije. Dimenzije slikovnih tekstura mogu biti samo u veličinama eksponenata baze 2 po širini i dužini. Neki noviji iPhone uređaji mogu imati i različite dimenzije od eksponenata baze 2, no tada nastaju velike kazne što se tiče performansi. Kako bi bilo jasnije o čemu je riječ slijedi primjer. Potrebno je uzeti papir i nacrtati kvadrat i označiti ga tako da su mu širina 256 piksela i visina 256 piksela. Sada treba nacrtati još jedan kvadrat unutar ovoga ali tako da mu dimenzije budu 129x129 piksela, dakle malo vise od polovine dimenzija teksture u kojoj se nalazi ovaj manji kvadrat. Dakle 129x129 kvadrat predstavlja sliku spremljenu u teksturu dimenzija 256x256 piksela.



Slika 1. Neiskorištena memorija

Pogledom na crtež lako je zamjetiti da je skoro tri četvrtine teksturnog prostora prazno. Bez obzira na bitovnu dubinu slike, samo jedna četvrtina memorije koju ova slika zauzima se zapravo koristi pri prikazivanju tog slikovnog elementa a tri četvrtine je izgubljeno. Ovo se može razriješiti stisnutim pakiranjem sprite-ova na teksturni atlas. Svaki bajt memorije teksture se mora kopirati sa jedne lokacije u memoriji koju adresira CPU u područje koje koristi GPU. Korištenjem ove dodatne neiskorištene memorije, stvara se niz negativnih efekata, a ti efekti postaju sve gori i gori što se povećava broj sprite-ova na ekranu koji nisu grupirani u hrpi (engl.

nonbatched CCSprites). Zaključak je da kombinacija neiskorištene a zauzete memorije zbog individualnih tekstura i individualna povezivanja OpenGL ES-a i tekstura mogu znatno umanjiti performanse igre. Postoji još jedna prednost korištenja teksturnog atlasa. Često velik dio slika sadrži dosta transparentnog prostora, to je prostor koji ne sadrži nikakve podatke o boji piksela a zauzima prostor jednako kao da nije transparentan. Mnogi algoritmi koji spajaju hrpu sličica na jedan teksturni atlas uzimaju u obzir i transparentni dio slike, te Cocos2D automatski pokušava slike okolo zbiti tako da što manje praznog prostora ostane u teksturi.



Slika 2. Slika pokazuje neiskorištenost memorije kod individualnih slika nasparam korištenja teksturnog atlasa

Zaključak – glavni razlozi za korištenje teksturnog atlasa (engl. texture atlas ili sprite sheet):

- Smanjen broj OpenGL ES poziva povezivanja – što je više slika sadržano u teksturnom atlasu, smanjenje je veće
- Smanjen memorijski otisak za slike spremljene kao teksture u memoriji

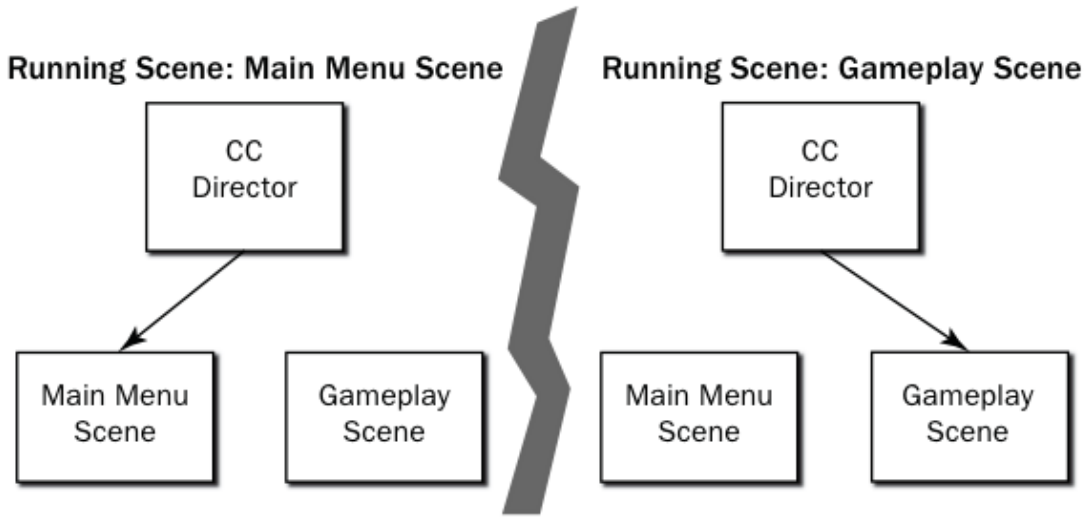
- Jednostavne metode odrezivanja praznog (transparentnog) prostora na slikama i na taj način ušteđa memorije jer ostale slike mogu zauzeti taj prostor koji bi inače ostao neiskorišten

Važno je napomenuti i da bi se postigle naglašavana poboljšanja što se tiče teksturnog atlasa potrebno je što bolje i što gušće popuniti atlas slikama. U ovom radu se nalazi poprilično mali broj sprite-ova – bakterija, stanica i nekoliko dijelova stanice, a ostalo su statične slike – tako da nije bilo potrebno koristit teksturni atlas jer se performanse ne bi osjetno popravile te su korišteni obični sprite-ovi, svaki sa svojom teksturom.

3.3.2. Scene

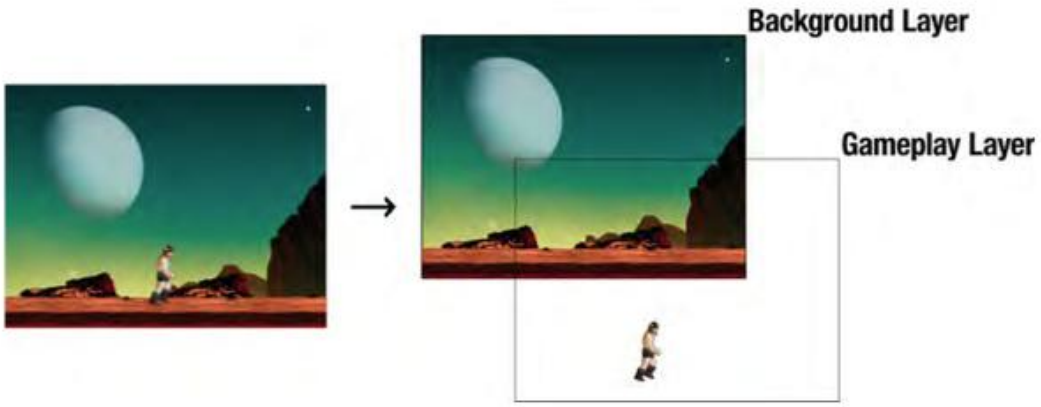
Kod sustava scena u Cocos2D-u potrebno je prvo vidjeti što je CCDirector. CCDirector je jedinstvena klasa (engl. singletone), posebna vrsta Objective C klase koja može imati samo jednu instancu. Kada se poziva singletone instanca uvijek se dobije samo jedna instanca te klase, bez obzira iz koje je klase pozvana. CCDirector je singletone koji je važan jer upravlja sustavom scena, preko njega se mijenjaju scene koje će trenutno biti prikazane na ekranu. Očito je da može biti samo jedan CCDirector jer odluka o tome koja scena se prikazuje treba biti jedinstvena, stoga je CCDirector i singletone.

Cocos2D igre su sastavljene od scena (CCScene), te direktora (CCDirector) koji je odgovoran za prikazivanje određene scene. Cocos2D direktor može djelovati samo sa jednom scenom u danom trenutku. Slika prikazuje kako u jednom trenu CCDirector odgovara za izvršavanje scene glavnog izbornika, a zatim se prebacuje na scenu igranja određenog dijela igre.



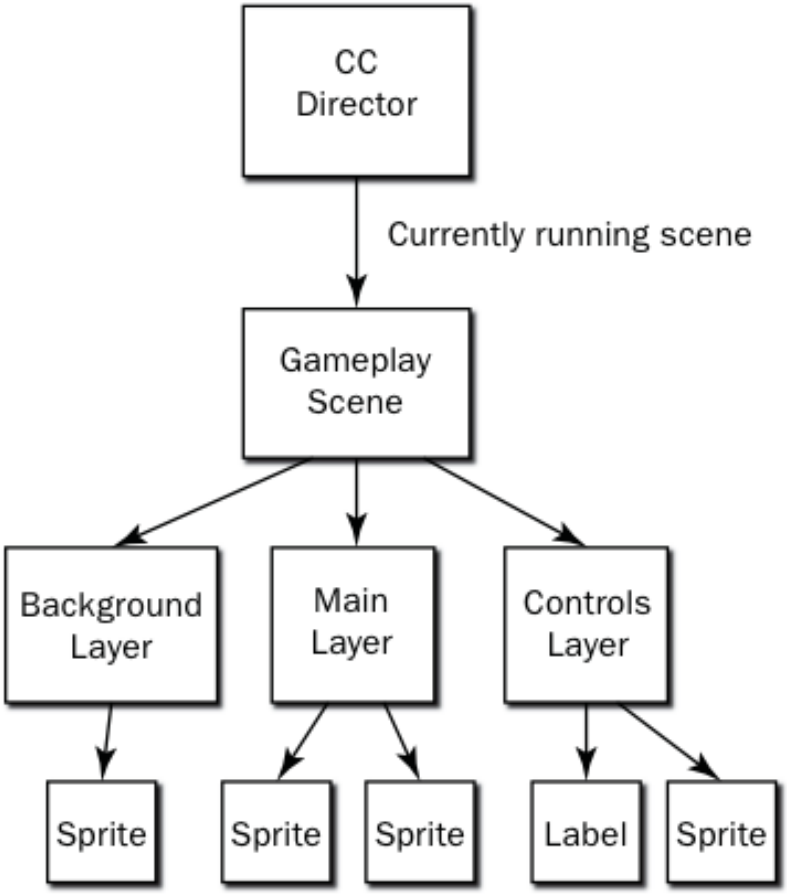
Slika 3. CCDirector prikazuje prvo scenu glavnog izbornika a zatim scenu igre

Svaka scena u Cocos2D igrama se sastoji od jedne ili više podloga (engl. layer), koji su posloženi tako da su jedni na drugome. Primjerice MainMenuScene u igri MakeMeMove ima samo jedan layer koji se naziva MainMenuLayer, zbog jednostavnosti ove scene nije potrebno imat više layera, no moglo se dodaji još jedan layer. Hipotetski BackgroundLayer koji bi bio čista statička pozadinska slika. Pošto se layeri nalaze jedan na drugome, samo bi se trebala postaviti z koordinata BackgroundLayer-a na manju vrijednost od z koordinate MainMenuLayer-a i tako bi se dobila scena sastavljena od više layera što je uobičajena praksa kada su scene složenije.



Slika 4. Više podloga u istoj sceni sa različitim z koordinatama

Jednom kada je stvoren layer na njega se dodaju pojedini objekti koji trebaju biti prikazani. Ovdje se prikazuju primjerice sprite-ovi i labele. Sve što se kreće zapravo je na layeru i također može imati namještenu z koordinatu čime se bira koji sprite će biti iznad kojega. Dakle scene su zapravo djeca direktora, a layeri su djeca scene, dok su svi sitniji elementi djeca layera, tj. mogu se dodati na layer.



Slika 5. Hijearhija scena, podloga (engl. layer) i raznih objekata na ekranu

Konkretno funkcija koja dodaje pojedini objekt svom roditelju je –(void)addChild: (CCNode *)aChild z: (NSInteger)z tag: (NSInteger)aTag. Preko metode addChild:z:tag dodaje se dijete roditelju pri čemu se može odmah i namjestiti z koordinata kako bi roditelj znao dodati dijete tako da bude u pravilnom poretku i da se vidi ili ne vidi u odnosu na neki drugi dodani objekta. CCNode je klasa koja je roditeljska svim klasama u Cocos2D-u, što znači da se svaki objekt može dodati nekom roditeljskom. Preko paramtera aTag može se namjestiti jedinstveni identifikator za pojedini objekt te mu se poslije lako može pristupiti funkcijom –(CCNode *)getChildByTag: (NSInteger)aTag. Ovo se koristi jako često,

jednostavno je i efikasno. Vraća objekt tipa CCNode* što znači da je potrebno napraviti cast u određeni tip, npr. u igri Make Me Move koristi se ova funkcija između ostalog i kod scene u kojoj bakterija lovi neutrofil. Kada se odlučuje o slučajnom smjeru kretanja neutrofila svakih nekoliko sekundi izvršava se dohvaćanje tog neutrofila upravo tom funkcijom.

```
CCSprite *neutrophil = (CCSprite *)[self getChildByTag:kNeutrophilTag];
```

Treba se primjetiti da je potrebno napraviti cast u CCSprite objekt, ali i jednostavnost dohvaćanja ovog sprite-a neutrofila bilo gdje unutar scene. U gornjoj naredbi se self odnosi na layer jer se u ovoj klasi ChasingBacteriaLayer sve odvija u tom layeru.

CCDirector također rješava i pitanja izlaska iz aplikacije ili prepoznavanja kada je aplikacija otvorena. Rješava i neke probleme koji bi zadavali muke da se trebaju vlastoručno implementirati kao što je recimo pauziranje aplikacije koje je prouzrokvalo operacijski sustav. Ovaj događaj se dešava kada primjerice korisnik zaključa ekran na svom iPhone-u dok je igrao igru ili kada igrač zaprima dolazni poziv i aplikacija odlazi u pozadinu. Sve te događaje rješava direktor i zaustavlja igru tako da kad će se igrač vratiti opet može nastaviti odakle je stao. Isto vrijedi i za suprotne događaje, na primjer kada korisnik otključa ekran i odluči nastaviti igru, CCDirector se pobrine da se igra jednostavno nastavi. Kada nastane krizna siituacija što se tiče memorije, CCDirector će izbaciti što više nepotrebnih tekstura sprite-ova koji se trenutno ne koriste. Taj poziv također izbacuje sve cach-irane teksture i bitmape fontova koje trenutno nisu na redu za renderiranje na ekran.

Kao što je već detaljnije opisano, sve slike se učitavaju u OpenGL ES teksture u formatu koji grafički procesor može razumjeti. Cocos2D sprite-ovi su zapravo poveznice ka ovim teksturama koje CCDirector i OpenGL ES renderiraju u igri. Cocos2D sadrži i teksturnog cache manager-a koji održava teksture cache-iranim u memoriji. Držanje tekstura u cachiranoj memoriji znatno povećava brzinu stvaranja novih sprite-ova koji će koristiti prijašnje korištene teksture. Negativna strana je višak memorije koji se stvori držanjem cachiranih tekstura u memoriji. Ako aplikacija zaprimi upozorenje o niskoj memoriji Cocos2D brzo miče iz memorije sve teksture koje se ne koriste aktivno. Stoga je potrebno zapamtiti

uvijek dealocirati sve scene i layere koji se više ne koriste kako bi memorijski otisak bio što manji.

CCDirector također brine o strukturi prozora koja se prikazuje na ekranu te postavlja delta vrijeme između posljednjeg poziva nekog događaja i trenutnog poziva na nulu. Ta metoda se poziva ukoliko je prošao znatan dio vremena između poziva događaja, primjerice kada se iPhonov sustav prilagođava na vraćanje vremena tijekom proljeća i jeseni ili pri mijenjanju vremenskih zona, te zbog varijacija vremena koje prikazuju telefonski tornjevi. Fizika i ostali izračuni unutar igre su osjetljivi na velike promjene vremena što bi moglo dati čudne rezultate i čudna ponašanja renderiranja, no o tome se brine CCDirector.

3.3.3. Akcije i animacije

Dodavanje animacija u Cocos2D-u je izrazito jednostavno i u tome se najviše demonstrira snaga ovog radnog okvira (engl. framework). Animacije u Cocos2D-u su kao poznati trik u kojem se nacrtaju stotinjak sličnih sličica i u svakoj sličici se lik malo pomakne, te kada se slike prezentiraju brzo izmjenjući jednu za drugom dobije se efekt kao da se slika miče. Otprilike tako rade i animacije u Cocos2D-u, animacije se postižu mijenjanjem slikovnih okvira više puta u sekundi. Tipično je za igre ova vrijednost 60 slikovnih okvira po sekundi (60fps – frame per second). To zapravo znači da sprite-ovi imaju kratke vremenske odgode između pokazivanja drukčijih tekstura svaki put kad se tekstura prikaže. Dva su koraka u kreiranju animacije u Cocos2D-u. Prvi je stvaranje objekta CCAnimation, kako bi se specificiralo koji skup tekstura se nalaze u animaciji, taj skup tekstura se nazvia okvrima (frames). Drugi korak je stvoriti akciju CCAnimate i pozvati ju da se izvrši na određenom sprite-u. CCAnimate zapravo specificira koji CCAnimation će se izvršiti na sprite-u pri postavljanju tekstura između kašnjenja kod izmjene okvira. Primjer iz rada je recimo kada naredimo da se bakterija pomakne u smjeru gdje igrač klikne na ekranu. Tada se poziva CCAnimation tipa CCMoveBy, koji pomiče sprite za određeni broj piksela po x i y koordinati čime se sprite miče u smjeru gdje je korisnik pritisnuo ekran svog iPhone-a.

```
CCAction *moveAction = [CCMoveBy actionWithDuration:1.0f position: CGPointMake(direction.x * 30, direction.y * 30)];
[bacteria runAction:moveAction];
// gdje je bacteria tipa CCSprite
```

3.4. Organizacija igre MakeMeMove

Sama igra MakeMeMove je organizirana kroz desetak klasa, od kojih su većina scene i podloge (engl. layers) gdje se odvija sva akcija vidljiva korisniku. Igra ima dvije singletone klase: AppDelegate i GameManager. U klasi AppDelegate već opisani CCDirector upravlja raznim događajima koji dolaze od operacijskog sustava kao što su upozorenja na nisku memoriju (engl. low memory warning) do izlaska iz aplikacije zbog viših prioriteta, primjerice uspostave dolaznog poziva (treba imati na umu da se ipak radi o mobitelu). Drugi singletone je znatno koncentriraniji na samu igru – GameManager. Klasa GameManger se brine o tome koja scena se trenutno prikazuje i o tome obavještava CCDirectora kako bi znao promijeniti scenu. Ova klasa je dostupna iz svih ostalih klasa projekta jer ovdje se nalaze i razne ostale stvari važne za igru koje su potrebne gotovo kroz cijelu igru kao što je primjerice broj dosad osvojenih bodova koje treba nekako prenijeti na sljedeći nivo, a GameManager se upravo brine o takvim sitnicama. Dakle GameManager zna za sve scene i sve scene znaju za njega, no same scene ne znaju ništa o ostalim scenama, znaju samo o sebi jer nema potreba ni za čim višim od toga. Tako se gradi i kod u skladu sa objektno orijentiranom paradigmom. Na ovaj način je svaka scena samo-sadržana (engl. self-contained) i nove scene se mogu dodavati bez brige o stvaranju međuovisnosti scena što je odlično svojstvo. Oblikovni obrazac jedinstvenog objekta (engl. singletone design pattern) je moćan oblikovni obrazac koji se u Cocos2D-u koristi i kod klasa CCDirector, CCTextureCache itd.

```
+(GameManager*)sharedGameManager {
    @synchronized([GameManager class])
    {
      if(!_sharedGameManager)
          [[self alloc] init];
      }
```

```
        return _sharedGameManager;
    return nil;
```

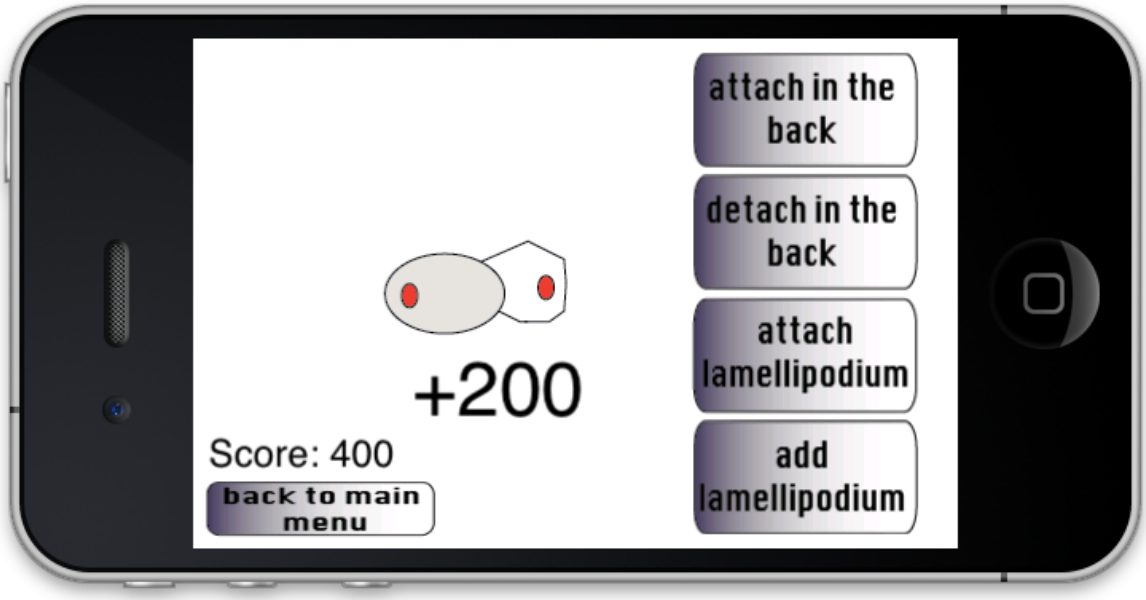
Gornji kod prikazuje (od @synchronized pa nadalje) kako je osigurano da čak ako i dva objekta u isto vrijeme žele dohvatiti GameManager-a, samo jedan to može i učiniti, ne mogu oba objekta u isto vrijeme pristupiti. Ovo je posebno važno jer tako se riješi problem višedretvenosti koji je kod igara zastupljen.

Nadalje na GameManager-u postoji metoda runSceneWithID kojom preko ovog objekta govorimo CCDirectoru koju scenu da prikaže. U ovom objektu su također sadržani podaci kao što je trenutni broj bodova ili razina do kuda je igrač trenutno došao. Obično se u ovom objektu pohranjuje i je li korisnik isključio ili uključio glazbu ili zvučne efekte itd.

Još jedna datoteka koju je važnu napomenutu u ovom projektu je i Constants.h u kojoj su pohranjene razne vrijednosti poput tipova scena u obliku enumeracija, razni tagovi s kojima se dohvaćaju pojedini sprite-ovi što je važno jer tako kod izgleda čitljivije nego da se umjesto definiranih konstanti svugdje po kodu pišu brojevi. Ovdje su također pripremljeni tipovi koji govore u kojoj je fazi igra (npr. kGameRunning, kGamePause i kGameFinished).

4. Rezultati

Kada se pokrene sama igra, prvo AppDelegate preko CCDirectora pokrene scenu glavnog izbornika (MainMenuLayer.h i MainMenuLayer.m datoteke). MainMenuLayer je jednostavna scena koja prikazuje dva gumba koji vode na dvije razine igre. Prije početka svake razine igre prvo se pokaže scena sa instrukcijama da bi igraču bilo lakše igrati dok se ne uhoda (InstructionsLayer). ZoomZeroLayer je zapravo nulta razina igre u kojoj igrač treba izabrati točne korake ponuđene u izborniku kako bi se stanica pokrenula.



Slika 6. Nulta razina igre u kojoj je potrebno odabrati točan redoslijed naredbi kako bi se stanica pomaknula

Nakon prelaska ovog dijela igre, pokreću se instrukcije za sljedeći zadatak, a to je lov bakterije na neutrofil. Ovdje prvi put se vidi prava akcija u igri. Igrač zadaje klikovima po ekranu željeni smjer kretanja bakterija na što se u tom smjeru iscrtava lamelopodij i točke adhezije koje se pojavljuju i iščezavaju kako se bakterija miče. Ovaj dio igre se zapravo jako nadovezuje na prethodnu savladanu razinu u kojoj je igrač trebao odrediti točan poredak naredbi u dodavanju i micanju lamelopodija i točaka adhezije, a ovdje sada korisnik ima priliku dobiti i upečatljiviju sliku kako se to zapravo događa kroz lov na neutrofil. Za prepoznavanje korisnikovih ulaza (engl. input) koji su zapravo dodiri prstima po ekranu koristi se funkcija Cocos2D-a ccTouchesEnded: touches withEvent: event

kojom se može prepoznati kada su dodiri prestali. Unutar same funkcije mogu se dohvatit pozicije dodira iz kojih se zatim računa u odnosu na trenutnu poziciju bakterije kuda dalje treba ići. Prepoznavanje kada je bakterija uhvatila neutrofil se odredi postupkom detektiranja kolizije. Naime u svakom trenu su poznate pozicije bakterije i neutrofila, te njihovi pravokutnici koji ih omeđuju (engl. bounding rectangle), te je jednostavno prepoznati kada je bakterija uhvatila neutrofil ispitivanjem funkcije CGRectIntersectsRect(CGRect rect1, CGRect rect2) koja govori dali se dva pravokutnika preklapaju.

Slika 7. Lov bakterije na neutrofil

Nakon uspješno ulovljenog neutrofila slijede instrukcije za složeniju scenu u kojoj se vidi stanica sa više detalja, a cilj je točno smjestit pojedine dijelove stanice na odgovarajuće pozicije kako bi se stanica pokrenula. Sve ovo se događa nad objektom Zoom10Layer koji dakako ima svoje sprite-ove i razne labele, te statičke slike dodane na raznim z koordinatama. Ovaj dio igre je organiziran kao stroj stanja. Naime u svakom trenu igra može biti u nekoliko stanja – može biti završena (pobjeda ili poraz), pauzirana (primjerice kada igrač želi pročitati detaljniju znanstvenu literaturu o pojedinom dijelu stanice, potrebno je kliknuti pauzu i ulazi se u način igre u kojem klikom na dijelove stanice više ih se ne može razmještati po stanici nego se otvaraju poveznice na odgovarajuću literaturu) ili igra može biti u tijeku (u igri implementirano kao kGameRunning). Ako je igra u tijeku dalje ima niz mogućih stanja, kao što su do kuda je igrač trenutno došao u

dodavanju dijelova stanice, te ako učini neki pogrešni korak vraća se na početnu scenu u kojoj je stanica "prazna". U svakom trenu potrebno je paziti na razinu aktina jer ako padne na nulu, igra je završena i igrač je izgubio. Stoga je potrebno znati iskoristiti kofilin između drugog i trećeg koraka kako bi porasla razina aktina u stanici.

Slika 8. Zoom10 level – detaljnije prikazani dijelovi stanice

U svakom trenutku, igrač može zaustaviti igru na što ulazi u način igranja u kojem klikom na bilo koji element na ekranu iskače novi prozor sa popisom poveznica na znanstvenu literaturu vezanu za pojedini pripadne pojam iz strukturne molekularne biologije.

5. Zaključak

Znanstvenici često koriste simulacije da bi testirali svoje hipoteze. Također koriste simulacije za ispitivanje složenih sustava u kojima je teško predvidjeti izlaz jer ovisi o brojnim parametrima. U tom slučaju postave razne parametre sustava, pokrenu simulaciju i jednostavno vide što će se dogoditi. Tada se traže objašnjenja za izlaze, stvaraju se nove teorije o složenim sustavima koji su simulirani, ponavljaju se simulacije kako bi se poboljšavala teorija itd. Igra *Make Me Move* ima na više načina analogiju se ovim pristupom. Igrač u ovoj igri treba podešavanjem raznih koraka vidjeti hoće li to doprinijeti cilju igre, da se stanica pomakne. Trenutno igra nije toliko složen sustav u kojem bi se mogle odvijati kompleksne simulacije, no ima potencijala da na spomenutima razinama Zoom100 i Zoom1000 bude više detalja u kojima će se moći namještati veći broj parametara i vidjeti što se tada događa u stanici. Ovakav način učenja zacijelo je upečatljiv, jer igrač ima priliku u igri naučiti kako pokrenuti stanicu, u jednom dijelu igre i upravljati kretanjem, dok se odvijaju animacije prethodno naučenih koraka. Nakon svega može i pročitati više znanstvene literature o pojedinim procesima i dijelovima stanice korištenim u igri. Na taj način igrač sigurno dobiva potpuniju sliku o molekularnim interakcijama unutar stanice i njihovog utjecaja na njeno kretanje.

6. Literatura

[1] R. Strougo, R. Wanderlich: *Learning Cosos2D,* 2011.

[2] Why are video games good for learning, URL *http://www.academiccolab.org/resources/documents/MacArthur.pdf*

[3] *Civilization nad its Discontents: Simulation, Subjectivity, and Space*, URL *http://game-research.com/index.php/articles/civilization-and-its-discontents-simulation-subjectivity-and-space/*

[4] C. Sykes, J. Plastino: *Actin filaments up against a wall*

[5] V. Achard, et al.: *A Primer-Based Mechanism Underlies Branched Actin Filament Network Formation and Motility*

[6] Thomas D., J. Cooper: *Actin, a Central Player in Cell Shape Movement*

[7] S. Kochen: *Programming in Objective-C 2.0*

[8] B. Geiger, J. Spatz, A. Bershadsky: *Environmental sensing through focal adhesion*

[9] J Victor, G. Resch: *The comings and goings of actin: coupling protrusion and retraction in cell motility*

[10] A. Ridley: *Life at the Leading Edge*

[11] J. Conway, A. Hillegass: *iPhone Programming: The Big Nerd Ranch Guide*

[12] J. LeMarche, D. Mark: *Beginning iPhone Development: Exploring the iPhone SDK*

[13] O'Reilly Media: *iPhone Game Development: Developing 2D & 3D games in Objective C*

Make Me Move – didaktička računalna igra za učenje strukturne molekularne biologije

Sažetak

Poznato je da igrači računalnih igara često pamte činjenice vezane uz igru. Cilj rada je primjeniti to saznanje u edukativne svrhe. Zadatak je razviti didaktičku aplikaciju koja igraču pomaže da savlada činjenice o kompliciranom biomolekularnom sustavu. Kao primjer sustava koristiti ulogu aktina u pokretljivosti stanice. Potrebno je simulirati molekularne interakcije u stanici i njihov utjecaj na njeno kretanje. Korisnik igre treba ponoviti točan niz interakcija kako bi uspješno pokrenuo stanicu. Sve interakcije trebaju imati poveznice na znanstvenu literaturu koja ih opisuje. Igru razviti za iPhone koristeći Cocos2D.

Make Me Move – didactic computer game for learning structural molecular biology

Abstract

It is well known that computer game players often memorize facts learnt from the game. The goal of this project is to develop a didactic application that helps the player to learn facts about complex biomolecular system. The role of actin in cell mobility is used as an example of a system. Molecular interactions in the cell and their impact on cell movement are being simulated in this application. The game user has to repeat the correct sequence of interactions in order to make the cell move. All interactions are linked to scientific literature that describes them. The game is developed for iPhone in the framework Cocos2D.