

Univerza v Ljubljani

Fakulteta za Računalništvo in Informatiko

SEMINARSKA NALOGA - OMEN

# RAČUNALNIŠKA GRAFIKA

Avtorja:

Nina Plevnik, 63210251

Tilen Smrdel, 63210304

Predavatelj: prof. [Iztok Lebar Bajec](http://www.fri.uni-lj.si/si/iztok-lebar-bajec/default.html)

Asistenta: Žiga Lesar

Kazalo vsebine

[RAČUNALNIŠKA GRAFIKA 1](#_Toc123578878)

[Uvod 3](#_Toc123578879)

[Pristop izdelave 4](#_Toc123578880)

[1. Idejna zasnova 4](#_Toc123578881)

[2. Razdelitev nalog 4](#_Toc123578882)

[Izdelava 4](#_Toc123578883)

[Vizualizacija 4](#_Toc123578884)

[Igralec: 4](#_Toc123578885)

[Mapa: 5](#_Toc123578886)

[Programerski vidik 10](#_Toc123578887)

[Base game: 10](#_Toc123578888)

[Igra 13](#_Toc123578889)

[Mehanika: 13](#_Toc123578890)

[Potek igre: 13](#_Toc123578891)

[Zaključek 14](#_Toc123578892)

[Viri in sklici: 15](#_Toc123578893)

# Uvod

### Pristop izdelave

### Idejna zasnova

Kakor je bilo že napisano v najinem predlogu seminarja, sva bila ob pričetku polna idej, ki pa jih je bilo sprva težko nakopičiti in urediti v celoto. Veliko idej je preko postopka izdelave bilo prilagojenih in veliko dopolnjenih.

### Razdelitev nalog

Po končanih idejah sva si razdelila naloge, seveda imava področja v katerih se vsak od naju bolje izkaže, zato je pomembno, da si naloge razdeliva tako, da v najkrajšem času čim boljše opraviva delo. Z tem razlogom sva delo ločila na vizualni in programerski aspekt, kjer je večinsko delo opravil eden od naju, seveda pa sva na obeh področjih delovala oba.

## Izdelava

### Vizualizacija

Vizualni aspekt je v igrah vedno zelo pomemben, le ta ustvari atmosfero in želen ambient, ki v nas vzpodbudi željo po igranju in raziskovanju. Seveda je v večini primerov to tisto, kar igralca lahko na prvi pogled pritegne ali odvre od igranja.

Želela sva se izkazati tudi v tem, saj sva imela v grobem že nekaj izkušenj z programom Blender. Te izkušnje pa sprva niso bile dovolj, zato sva morala najina obzorja razširiti in se naučiti še veliko več.

### Igralec:

Začela sva z modelom igralca, pri katerem sva prišla že do prvih komplikacij. Model je sestavljen iz 5 kosov, to so: trup, leva in desna roka in noga. Prvi problem je nastal v točko rotacije, ki jo je bilo primerno prilagoditi za uporabo animacije, težavo sva rešila z premikom točke na stikališče trupa in uda.

Po izdelavi modela pa je prišel prvi večji izziv, teksture in UV mapping, ker na tem področju še nisva imela izkušenj sva sprva potrebovala veliko pomoči. V tem primeru sva si pomagala z orodjem pametno odvijanje, kjer sva kose ločila na takšen način, da se le te ne prekrivajo, kar je bilo v najinem primeru zadostujoče. Ko je bilo to urejeno sva objekt pobarvala.

Zanimiv problem, do katerega sva prišla v programerski fazi je bil težava, ki je nastala zaradi nepravilne izdelave ene od okončin. Leva roka je v animaciji delovala, desna pa ne. Ugotovili smo da je problem nastal, saj objekt ni bil pravilno prezrcaljen vendar le skaliran v drugo smer. Po popravkih je bil problem rešen.

Spodnja slika prikazuje objekt možic:



### Mapa:

Naslednji korak je bila izdelava okolja, kjer sva se prvič srečala v objektom takšne velikosti. Izdelala sva celotno mapo, nato pa kar hitro prišla do velikega problema, ki nama je spremenil pristop izdelave. No sva želela izdelati UV mapping in teksture, sva ugotovila da bi celoten postopek morala ponoviti, za vsak objekt, tudi če se ponavljajo, če pa bi zajela vse skupaj bi za vsak na novo dodan objekt imela enak problem. Tako sva prišla do ugotovitve, da morava izdelati assete.

#### Assets:

1. Stene*:*

Seveda sva želela, da bi bila igra čimbolj vizualno razgibana, izdelala sva 3 različice sten, ki se ne razlikujejo le po teksturi, temveč tudi v samem 3D objektu.

**Stena1:** Prvi asset z katerim sva pričela je bila stena1, zato sva pri tem objektu imela tudi največ problemov. Ugotovila sva da pristop z pametnim odvijanjem mappingom tukaj nebo deloval, saj le ta popači obliko in ni ugoden za risanje. Zato sva uporabila 2 metodi:

1. *označevanje robov z MEAN:* V tej metodi, izdelovalec sam začrta robove, na katerih loči stik, to pa sva uporabila predvsem za objekte oblik likov kvadrat in pravokotnik. Uporaba tega orodja, nam dopušča, da objekt odvijemo kot plašč.
2. *uporaba projekcijskega pogleda za odvijanje(unwraping):* Ta metoda se je izkazala za najbolj uporabno v najinem primeru, saj je pogled iz vsake strani dobro ločen. V tej metodi uporabnik zajeme plošče, ki jih želi odviti, ter se postavi točno na os iz katere želi da se oblike preslikajo. Metoda seveda ni le zelo hitra, vendar tudi zelo efektivna, saj ploščinam dobro ohrani proporce. Zanimivo dobo se je izkazalo tudi to, da v veliko primerih ni bilo potrebno ponavljanje teksture, saj je bilo potrebno le prekrivanje 2 kosov, kjer je eden prezrcaljen.

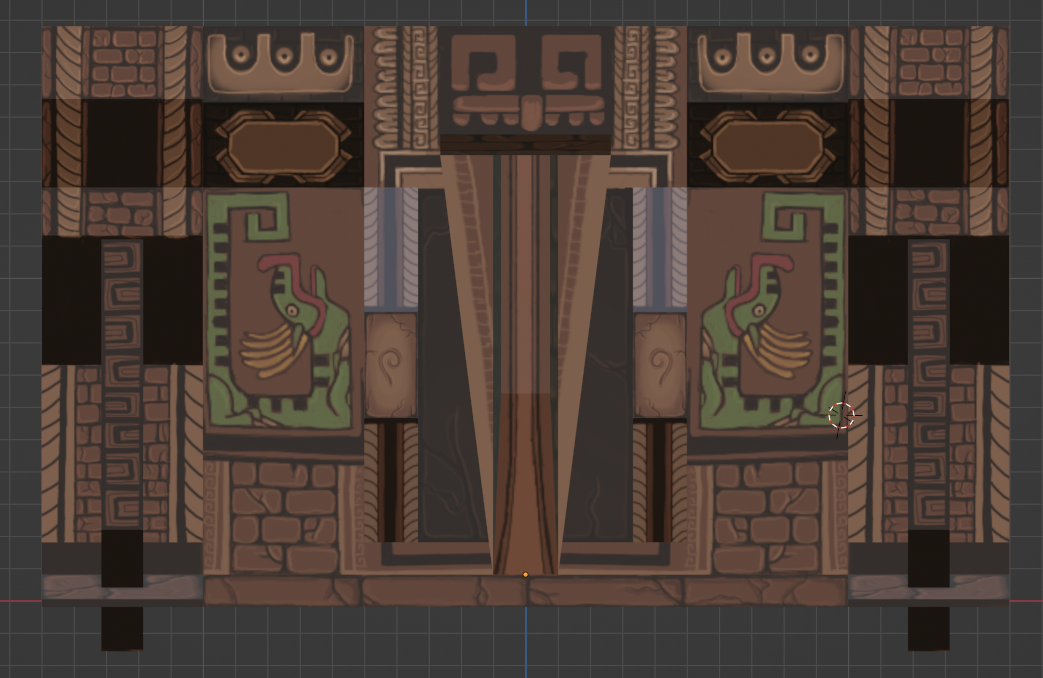
Po Uv mappingu je sledil kreativni del, risanje tekstur, teksture sva narisala direktno v programu Blender.

Spodnja slika prikazuje asset stena 1:



**Stena 2:** Objekt je bil izdelan na identični način, kot pri steni 1.

Spodnja slika prikazuje asset stena 2:



**Stena 3:** Ta objekt je bil sprva izdelan, kot 2 ločena objekta, kjer bi imela levi in desni rob stene. Zaradi pomankanja časa sva se odločila izdelati steno, ki se jo lahko doda na steno 1 in 2, ter prav tako lahko uporabi kot samostojno steno.

Spodnja slika prikazuje asset stena 3:



##### Talne površine:

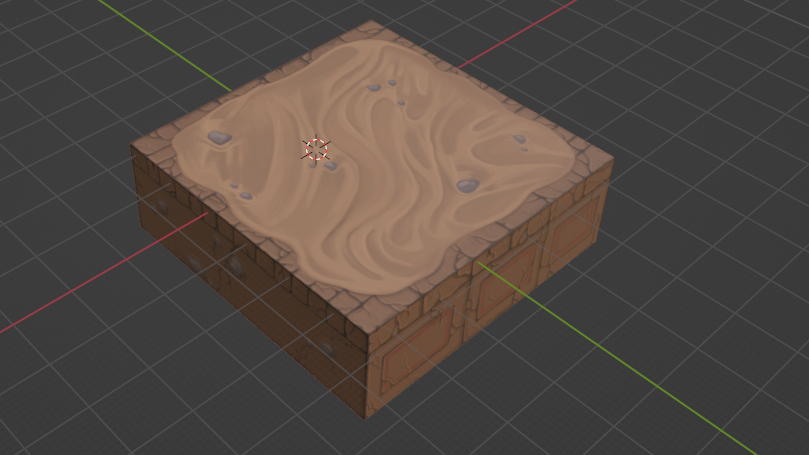
Talne površine so sestavljene iz 2 objektov, 1.objekt je teksturiran in predstavlja vizualno povezavo, v njem pa je skrit 2. objekt z lastnostmi: kolizija, možnost skoka in skrivanjem

Opis lastnosti:

* Kolizija: preprečevanje prehoda objekta preko drugega
* Skrivanje: objekta igralec ne vidi
* Možnost skoka: na površini je možen skok
* Smrt: ob stiku z objektom karakter izgubi življenje

**Tla kvadrat:** površina velikosti 30x30x10, za UV mapping je bila uporabljena metoda odvijanje z uporabo projekcijskega pogleda, saj je objekt preprost.

Spodnja slika prikazuje asset tla kvadrat:



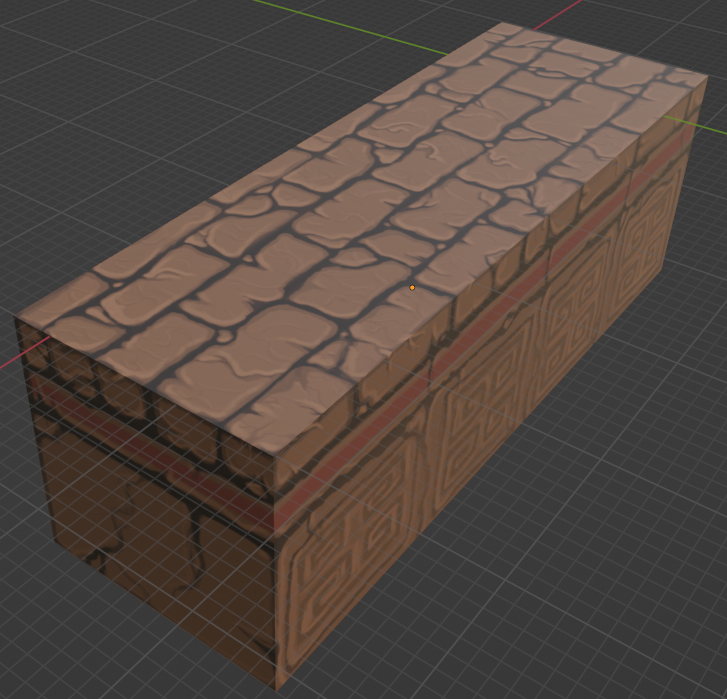
**Tla hodnik 1:** površina velikosti 30x10x10.

Spodnja slika prikazuje asset tla hodnik 1:



**Tla hodnik 2:** kopiran objekt tla hodnik 1, razlika je samo v vrhnji teksturi.

Spodnja slika prikazuje asset tla hodnik 2:



##### Platforme:

Platforme so uporabljene dekorativno, prav tako pa tudi kot objekt, ki predstavlja dinamiko igre. Platforme so lahko premikajoče ali statične, to pa je bilo urejeno programsko

**Platforma:** Pri tem objektu so teksture na vseh straneh, obratni stranici sta samo zrcaljeni.

Spodnja slika prikazuje asset platforma:



##### Dekoracija/Ostalo:

**Krogi:** objekt je bil eden najbolj zahtevnih za izdelavo UV mappa, saj sva imela težavo z ločitvijo celotnega kroga, ter ohranjanjem njegove oblike za lažje risanje. Tudi ta problem je bil rešen z uporabo odvijanja z projekcijo. Velik del odvijanja je bil narejen tudi z metodo mean.

Spodnja slika prikazuje asset krog:



**Špice:** model se ločuje na 2 objekta, kjer je prvi objekt dekorativen, objekt, ki pa je skrit označuje mejo, kjer igralec ob koliziji z njim izgubi življenje.

Spodnja slika prikazuje asset spice:

**Kamn:** dekorativen objekt, z kolizijo

Spodnja slika prikazuje asset kamen:

#### Izdelava mape:

Po izdelavi assetov je bila izdelava mape dokaj preprosta, odprla sva novo Blender datoteko in po potrebi v njo vnesla/pripela želene objekte. Začela sva z izdelavo poti po kateri se bo naš karakter premikal čez naloge, nato pa celotno sceno dekorirala.

### Programerski vidik

Igre sva se programersko lotila z naslednjimi datotekami:

### Base game:

#### 3d modeli:

Vsebina je opisana v točki vizualizacija. Vsebuje vse 3D modele in njihove teksture, parav tako pase tu nahajajo tudi vse JavaScript datoteke za animacije. Programi za animacijo so klicani v Omen.js.

#### Audio:

Datoteka vsebuje vse zvočne datoteke, uporabila sva vgrajene funkcionalnosti Java Scripta za delo z zvokom. Audio shranimo kot novo spremenljivko, ter upravljamo z zvokom z klici funkcij kot so .pause() , .play() itd.

#### Bin:

Server,js: datoteka server,js nam je bila podana, datoteke nisva spreminjala. Funkcionalnost te datoteke je klic ostalih Java Script datotek.

#### Engine:

Base: mapa zajema glavne funkcionalnosti brez katerih bi bila igra neuporabna. Sestavlja jo:

* *application.js:*

V tej datoteki inicializiramo program. Definairamo funkcijo .ubdate() in .resize().

* Char\_cont.js:

glavni kontroler za karakterja, pripravi vse lastnosti za nadaljno porabo. Tu deklariramo poslušanje tipk, kar ločimo na:

KeyUP, pritisk gumba.

KeyDOWN, gumb ni več pritisnjen.

KeyPRESSED, gumb smo pritisnili, vendar dejanje ni ponavljajoče, na primer skok in moči.

Ustvaimo tudi nekaj novih funkcij kot so:

Is\_moving() – ali se igralec premika

getCharRotation() – uporaba za Cam\_cont.js

* Cam\_cont.js:

sekundarni kontroler, podobn je Char\_cont.js.

Na začetku nastavimo pitch in yaw na CharRotation(), nato nastavimo vse potrebno, razen rotacije, saj se kamera ne rotira, le premika z karakterem

* Node.js:

program pregleda vse node iz Blender datotek, ter prebere vse potrebne lastnosti, da lahko le te uporabljamo.

* WebGL.js:

v tej datoteki pokličemo senčilnike in complier, prav tako pa kreiramo vse kar je potrebno, da se bodo stvari izrisale.

Ostalo:

* Abilities.js:

Po imenu je razvidno, da je to mapa v kateri je zapisana koda za moči. Koda deluje tako, da moč dobi objekt, kateremu nastavi lastnosti odvisno od moči.

* Accessor.js:

Dostopa do vseh elementov, ki so nam vili podani iz vaj.

* Bufferview.js:

Nastavi lastnosti byteOffset, byteLenth, byteStride. Dobljeno iz vaj.

* Camera.js:

Pripravi kamero.

* GLTFLoader.js:

Uporaba za nalaganje ostalih datotek iz knjižnice engine.

* Material.js:

Uporaba za lastnosti materialov.

* Mesh.js:

Uporaba za primitive.

* OMEN.js:

Glavna datoteka, znotraj nje prvo pokličemo vse 3d modele, prebereva sceno, ter prebereva kamero iz karakterja. Celotna vsebina iz Char\_cont.js se navezuje na to datoteko.

Animacija se ločuje na:

* igralec se ne premika: se dogaja idele\_animation, torej majhen premik rok,
* igralec se premika: roke in noge se premikajo kot hoja
* OrthographicCamera.js in PerspectiveCamera.js

Nanaša se na kamero.

* Physics.js:

Vsebuje kodo potrebno za kolizijo. Pomembo je tudi, da se prepričamo da objekt ne trči sam z sabo.

* Primitive.js:

Primitivi za Mesh.js.

* Rendrer.js:

Uporabi ostale datoteke za renderiranje.

* Sampler

Ustvari sampler.

* Scene

Z uporabo nodov vzame sceno iz Blender datoteke.

* Shaders.js:

In progress, zadovzeno za izgled, barve, luci

* Texture.js:

Vsebuje vse potrebno za teksture.

* Utils.js:

Vsebuje init() in clone()

#### Index.html:

#### Style.html:

## Igra

### Mehanika:

Med izdelavo premikanja sva se odločila za nov pristop, premikanje sva postavila na tipka WASD, ter se odločila da se bo našega karakterja rotiralo z uporabo miške. Ta sprememba je bila narejena ob ugotovitvi da bo takšen način imel boljšo efektivnost in prispeval k uporabniški izkušnji. Zaradi te odločitve sva tipke za moči prestavila na 1234.

Seveda pa se igre ne igra samo z tipkovnico in miško, velik del uporabnikov za igranje uporablja krmilnike igralnih konzol. Zato sva poleg tipkovnice implementirala tudi krmilnik.

### Potek igre:

## Zaključek

## Viri in sklici: