SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN

Hrvoje Lesar

IZRADA PLATFORME ZA PRAĆENJE BITKE U IGRI IGRANJA ULOGA

DIPLOMSKI RAD

Varaždin, 2025.

SVEUČILIŠTE U ZAGREBU

FAKULTET ORGANIZACIJE I INFORMATIKE

V A R A Ž D I N

Hrvoje Lesar

Matični broj: 0016133479

Studij: Organizacija poslovnih sustava

IZRADA PLATFORME ZA PRAĆENJE BITKE U IGRI IGRANJA ULOGA

DIPLOMSKI RAD

Mentor:

Prof. dr. sc. Markus Schatten

Varaždin, rujan 2025.

Hrvoje Lesar

Izjava o izvornosti

Izjavljujem da je ovaj diplomski rad izvorni rezultat mojeg rada te da se u izradi istoga nisam koristio drugim izvorima osim onima koji su u njemu navedeni. Za izradu rada su korištene etički prikladne i prihvatljive metode i tehnike rada.

Autor potvrdio prihvaćanjem odredbi u sustavu FOI Radovi

Sažetak

U ovom radu prikazan je razvoj i funkcionalnosti platforme za praćenje bitke u igri igranja uloga s glavnim fokusom na implementaciju pravila igre Dungeons & Dragons. Teorijski dio prolazi kroz povijest razvoja igara igranja uloga, od korijena u obuci časnika u zapovijadanju i planiranju bitke, do današnjih igara. Nadalje opisuje sustav Dungeons & Dragons kao jedan od najpoznatijih sustava za igranje igara uloga. Prolazi kroz mnoga najbitnija pravila za razumijevanje igre.

Praktični dio se bavi prikazom implementacije umrežene aplikacije. Opisana je implementacija klijent i poslužitelj dijelova aplikacije te uz primjere koda prikazani su neki zanimljivi dijelovi implementacije i arhitekture aplikacije.

Ključne riječi: Računalne igre; Dungeons & Dragons; Role playing game; Web aplikacija; Razvoj aplikacija; Mrežne igre

Sadržaj

1. Uvod

Igre igranja uloga (engl. *Role playing games*, skraćeno RPG), doživjele su značajan porast popularnosti u posljednjem desetljeću, kako u klasičnim stolnim varijantama tako i u digitalnom obliku. Iako se ove igre temelje na mašti i zajedničkom pripovijedanju, njihova mehanika često uključuje kompleksne sustave pravila, praćenje brojnih parametara likova te precizno vođenje borbenih susreta. Tradicionalni način igranja uz papir, olovku i minijature sve se češće nadopunjuje ili zamjenjuje digitalnim alatima koji olakšavaju administraciju igre.

Suvremene softverske tehnologije omogućuju razvoj specijaliziranih platformi koje olakšavaju upravljanje i praćenje borbe u igrama igranja uloga. Takve platforme mogu automatizirati izračune, osigurati konzistentnu primjenu pravila te unaprijediti korisničko iskustvo sudionika igre. Unatoč postojećim rješenjima, često postoji potreba za prilagodljivim sustavima koji su usmjereni na specifične zahtjeve određenog sustava igre ili načina igranja.

Cilj ovog rada je razvoj i prikaz implementacije umrežene platforme za praćenje bitke u igrama igranja uloga, s primarnim fokusom na implementaciju pravila iz sustava Dungeons & Dragons. U sklopu rada analizirani su funkcionalni zahtjevi sustava, odabrane su odgovarajuće tehnologije za implementaciju, kako bi aplikacija bila što više prenosiva i dostupna na različitim uređajima. Implementirano rješenje podržava osnovne i napredne elemente borbe.

Prvi dio rada posvećen je teorijskoj obradi igara igranja uloga i upoznavanju sa sustavom Dungeons & Dragons. Sustav je dosta opsežan te se u ovom radu uglavnom spominju i objašnjavaju osnovna i najbitnija terminologija i mehanike igre.

Praktični dio rada prikazuje tehnologije korištene za implementaciju i dizajn platforme, opisuje kako i zašto su određene tehnologije odabrane i neke zanimljive implementacijske detalje. U aplikaciji je također djelomično implementiran sustav Dungeons & Dragons te je primjer postavljanja scena i bitke prikazan na kraju rada.

2. Igre igranja uloga

Igre igranja uloga predstavljaju jedan od najsloženijih i kreativnih spojeva priče, izvedbe i igre u suvremenoj kulturi. Ovakva igra poziva igrače da zajednički stvaraju i nastanjuju zamišljeni svijet, vođeni kombinacijom strukturiranih pravila i spontanog pripovijedanja. Svaka sesija je istovremeno i igra i priča, a njezin ishod oblikuju mašta igrača i slučajnost ishoda bacanja kockica.

Igre igranja uloga nemaju jednu definiciju i možemo ih definirati s različitih gledišta:

- Situacija igranja uloga definirana kao situacija u kojoj se od pojedinca izričito traži da preuzme ulogu koja inače nije njegova, ili ako jest njegova, onda u okruženju koje nije uobičajeno za izvođenje te uloge. [1]
- Igranje uloga nije jedinstvena jasno definirana aktivnost, već čitav niz aktivnosti okupljenih pod pogodnim nazivom. Na jednom kraju spektra nalazi se intenzivno odražavanje osobnih emocija, dok se na drugom kraju nalazi situacija u kojoj je preuzimanje uloge bliže konceptu zagovaranja. [2]
- Igranje uloga je umjetnost iskustva, a stvaranje igranja uloga znači kreiranje novih iskustava. [2]
- Igranje uloga definira se kao bilo koji čin u kojem se istovremeno stvara, dodaje i promatra imaginarna stvarnost. [2]
- Igra igranja uloga mora se sastojati od interaktivnog pripovijedanja: sposobnosti likova i razrješenja radnji, definirani su brojevima ili količinama kojima se manipulira prema određenim pravilima. Donošenje odluka igrača pokreće i pomiče priču unaprijed. Uz skupinu koja djeluje kao autor, priča organski raste i odigrava se, bivajući doživljena od svojih stvaratelja. [2]

2.1. Ratne igre

Početak i inspiracija za igre igranja uloga dolazi iz ratnih igara. Nasljeđe ratnih igara seže od šaha, budući da su prve igre unutar posebne kategorije ratnih igara uvelike posuđivale ploče, figure i mehanike upravo iz šaha. Georg von Reißwitz se smarta ocem ratnih igara, jer je razvio prvi sustav ratnih igara koji je široko korišten kao ozbiljan alat za obuku i istraživanje. Razvijenu igru su nazvali tzv. `Das Kriegsspiel` te je igra zadovoljila dugo prepoznatu potrebu za jeftinim i lako ponovljivim sredstvom za obuku časnika u zapovijedanju i planiranju bitke. Vojne ratne igre gotovo se uvijek fokusiranju na sadašnjost, na ondašnje vojske, tehnologiju i države u vrijeme njihova nastanka. Veliki dio razvoja ratnih igara osamnaestog i devetnaestog stoljeća održava neprestana poboljšanja u sredstvima i provedbi ratovanja te posljedičnu potrebu da se ratne igre stalno usklađuju s realnošću. [3]

Tek kad su se hobisti počeli poigravati ovim sustavima, uspjeli su osloboditi ratne igre ograničenja suvremenog konteksta i istražiti povijesna razdoblja, buduće moguće svjetove, pa

čak i nemoguće fantastične svjetove. Hobisti su također odbacili strogo reproduciranje stvarnih uvjeta na bojištu u korist više uravnoteženijeg pristupa koji je kombinirao realističnost i igrivost. Do 1960-ih, ovi zaigrani ljubitelji ratnih igara transformirali su ih iz sredstva za vojnu obuku u znatno maštovitiju aktivnost, onu koja je mogla poslužiti kao temelj za modeliranje događaja u igri poput Dungeons & Dragons (skraćeno D&D). [3]

2.2. Nastanak sustava Dungeons & Dragons

Dungeons & Dragons (kraće poznato kao D&D), igra nastala 1974. godine predstavlja ključni trenutak u povijesti igara uloga. Igru su razvili Gary Gygax i Dave Arneson, koji su težili proširenju mogućnosti stolnih ratnih igara prema novim narativnim i imaginarnim domenama. Prije D&D-a, Gygax je već imao utjecaj u zajednici igranja ratnih igara (dalje u tekstu: wargaming), osobito kroz svoj rad na igri `Chainmail`, srednjovjekovnoj minijaturnoj ratnoj igri koju je razvio i bio jedan od autora 1971. godine. `Chainmail` je izvorno zamišljen kao skup pravila za simulaciju srednjovjekovnih bitaka s minijaturama, no uključivao je i fantastični dodatak koji je omogućava igračima da u igru unesu mistična bića i čarobne elemente. Ova dopuna održavala je Gygaxovu fascinaciju srednjovjekovnom literaturom i fantastičnim narativima, posebno djelima J.R.R Tolkeina i Roberta E. Howarda. [4]

Dave Arneson je eksperimentirao s narativnim pristupom u svojoj kampanji Blackmoor, u kojoj su se pojedinačni likovi mogli razvijati kroz više sesija i aktivno utjecati na tijek priče, za razliku od tradicionalnog wargaminga u koje igrači kontroliraju cijele jedinice ili vojske. Arnesonove inovacije, u kombinaciji s Gygaxovim strukturiranim sustavom pravila iz Chainmaila, stvorile su temelje za mehaniku i narativni potencijal igre D&D. [4]

Prvo izdanje D&D-a je izdano 1974. godine. Bilo je u početku skromno po opsegu, no uvelo je revolucionarne koncepte; igrači su mogli preuzeti uloge različitih likova istraživati fantastične tamnice, sudjelovati u zadacima vođenim od strane Dungeon Mastera (DM-a), koji moderira svijet igre. Pravila D&D-a uključivala su elemente minijaturnog wargaminga, vjerojatnosnih mehanika s kockama i suradničkog pripovijedanja, stvarajući novi oblik interaktivne zabave koji je spajao strategiju, maštu i narativ. Tijekom vremena igra se razvila kroz više izdanja, od kojih je svako unaprijedilo pravila i proširilo mogućnosti, no osnovni princip suradničkog pripovijedanja i igre vođene likovima je ostao nepromijenjen.

3. Opis pravila igre - Dungeons & Dragons

Pravila igre D&D razvila su se od početne koncepcije igre 70-tih godina 20. stoljeća. Poglavlje će biti usredotočeno na pravila definirana u `Player's Handbook 2014`. Svakim novim izdanjem igre pravila se suptilno nagorađuju, poboljšavaju i razvijaju te se dodaju nove mehanike igre.

Dungeons & Dragons je igra igranja uloga posvećena pripovijedanju u svjetovima mača i čarolije. Dijeli elemente s dječjim igrama pretvaranja. Kao i te igre, D&D pokreće mašta. Radi se o zamišljanju visokog dvorca pod olujnim noćnim nebom i predočavanju kako bi pustolov mogao reagirati na izazove koje taj prizor predstavlja. Za razliku od igra pretvaranja, D&D pričama daje strukturu, način određivanja posljedica za akcije igrača. Igrači bacaju kockice kako bi razriješili jesu li njihovi napadi pogodili ili promašili, ili mogu li se njihovi pustolovi popeti na liticu, izmaknuti se udaru čarobne munje ili izvesti neki drugi opasan pothvat. Sve je moguće, ali kockice čine neke ishode vjerojatnijima od drugih. [5]

Jedan igrač preuzima ulogu Dungeon Mastera, glavnog pripovjedača i suca igre. DM stvara pustolovine za likove, koji se kreću kroz njihove opasnosti i odlučuje koje će putove istražiti. Više o DM-u će biti u potpoglavlju 3.2.

Svaki igrač stvara pustolova (koji se naziva i likom) i udružuje se s drugim pustolovima. Radeći zajedno, grupa može istraživati mračnu tamnicu, ukleti dvorac, izgubljeni hram... Pustolovi mogu rješavati zagonetke, razgovarati s drugim likovima, boriti se protiv čudovišta i otkrivati čudesne čarobne predmete i blago.

3.1. Lista pojmova

Ovo poglavlje služi za definiranje terminologije i kratica koje će se koristiti kroz rad. Svaki termin će biti kraće opisan i potencijalno imati engleski naziv (koji je uglavnom više prepoznatljiv nego hrvatski prijevod), no cijeli kontekst ne mora biti razumljiv u spisu, već će biti opisan u nekom od sljedećih poglavlja.

- **D&D** Dungeons & Dragons
- **DM** Dungeon Master
- **NPC** Non-Player Character
- **D20** Kockica s dvadeset strana
- **D2, D4, D6, D8, D10, D12** Kockice s određenim brojem strana
- **AC** Klasa oklopa, armour class
- **DC** Težina zadatka, difficulty class
- **HP** Životni bodovi, hit points

- **XP** Bodovi iskustva, Experience points
- **Ability score** Sposobnosti, rezultat sposobnosti
- **Hit die** Kockica za životne bodove

3.2. Dungeon Master

Dungeon master je kreativna sila iza D&D sesije. DM stvara svijet koji ostali igrači istražuju te također osmišljava i vodi pustolovine koje pokreću priču. Pustolovina se obično temelji na uspješnom dovršetku neke potrage i može trajati kratko poput jedne sesije ili se protezati kroz više sesija. Duže pustolovine mogu uvući igrače u velike sukobe za čije je rješavanje potrebno više sesija. Kada se povežu u niz pustolovine čine trajnu priču nazvanu kampanja. D&D kampanja može uključivati desetke pustolovina i trajati jednu sesiju, mjesecima ili pa ćak i godinama. [6]

Dungeon master preuzima mnogo različitih uloga. Kao arhitekt kampanje, DM stvara pustolovine postavljajući pozicije čudovišta, zamka i blaga koje likovi drugih igrača mogu otkriti. Kao pripovjedač, DM pomaže ostalim igračima vizualizirati što se događa oko njih, improvizirajući kada pustolovi učine nešto ili odu negdje neočekivano. Kao glumac, DM igra uloge čudovišta i sporednih likova, time im daje život. A kao sudac, DM tumači pravila i odlučuje kada ih se treba pridržavati, a kada ih promijeniti. Svaki DM pristupa ulogama izmišljanja, pisanju, pripovijedanju, improvizaciji, glumi i suđenju drugačije i vjerojatno će igrači i sam DM u nekim uživati više nego u drugima.

D&D pravila pomažu DM-u i ostalim igračima da se dobro zabave, ali pravila nisu glavni autoritet. DM je glavni i upravlja igrom te smije "kršiti" ili mijenjati definirana pravila. Ipak, cilj DM-a nije pobiti pustolove, već stvoriti svijet kampanje koji se vrti oko njihovih djela i odluka te osigurati da se igrači vraćaju po još.

Uspjeh D&D sesije ovisi o sposobnosti DM-a da zabavi ostale igrače za stolom, dok je njihova uloga stvaranje likova, udahnuti im život i pomoći usmjeravati kampanju kroz postupke svojih likova. DM treba održati igrače (i sebe) zainteresiranima i uživljenima u svijet koji je stvorio te omogućiti njihovim likovima da čine nevjerojatne stvari. Poznavanje u kojim dijelovima D&D-a igrači najviše uživaju pomaže DM-u stvoriti i voditi avanture u kojima će uživati i kojih će se sjećati. [6]

Aktivnosti na koje se DM može fokusirati za bolje zadovoljstvo igrača [6]:

- Gluma; igrači koji uživaju u glumi vole ulaziti u ulogu svog lika i govoriti njihovim glasom. Kao igrači uloge u duši, oni uživaju u društvenim interakcijama s NPC-ovima, čudovištima i ostalim članovima družine.
- Istraživanje; igrači koji žude za istraživanjem žele iskusiti čuda koja nudi fantastični svijet. Žele znati što se nalazi iza sljedećeg ugla ili brda, također vole pronalaziti skrivene tragove i blago.

- Poticanje akcije; neki igrači vole poticati akciju, željni su da se stvari događaju čak i ako to znači preuzimanje opasnih rizika. Rađe će srljati u opasnost i suočiti se s posljedicama nego se suočiti s dosadom.
- Borba; drugi igrači uživaju u fantastičnoj borbi poput premlaćivanja zlikovaca i čudovišta. Traže bilo kakav izgovor da započne borba, preferirajući akciju nad pažljivim promišljanjem.
- Optimizacija lika; igrači koji uživaju u optimizaciji sposobnosti svojih likova vole podešavati svoje likove za vrhunske borbene performanse stjecanjem razina, novih značajki i čarobnih predmeta, rado će prihvatiti svaku priliku da pokažu nadmoć svojih likova.
- Rješavanje problema; igrači koji žele rješavati probleme vole proučavati motivacije drugih likova, razmrsiti spletke zlikovaca, rješavati zagonetke i smišljati planove.
- Pripovijedanje; igrači koji vole pripovijedanje žele doprinijeti priči. Sviđa im se kada su njihovi likovi uključeni u priču koja se razvija i uživaju u susretima koji su vezani s glavnom radnjom i proširuju je.

3.3. Sustav D20

Kockica s dvadeset strana najčešće je korištena za određivanje uspjeha ili neuspjeha radnji u D&D-u i tu kockicu nazivom D20. Ponekad se koriste dvanaesterostrane, osmerostrane, šesterostrane, četverostrane kockice i novčić, one također imaju nazive ovisno o broju strana, D12, D10, D8, D6, D4 i D2.

Svi likovi i čudovišta u igri definirani su kroz šest osnovnih sposobnosti: snaga (engl. *strength*), spretnost (engl. *dexterity*), izdržljivost (engl. *constitution*), inteligencija (engl. *intelligence*), mudrost (engl. *wisdom*) i karizma (engl. *charisma*). Kod većine pustolova vrijednosti tih sposobnosti kreću se od 3 do 18, dok kod čudovišta mogu biti niske poput 1 ili visoke do 30. Te vrijednosti, kao i modifikatori koji iz njih proizlaze, temelj su gotovo svakog bacanja D20 kockice

Tri glavne vrste bacanja čine srž pravila igre: provjera sposobnosti, bacanje za napad i bacanje za spas (engl. *saving throw*). Sve tri radnje slijede nekoliko koraka:

1. bacanje kockice i dodavanje modifikatora. Na rezultat bačene D20 kockice dodajemo odgovarajući modifikator. Najčešće je to modifikator jedne od šest sposobnosti, a ponekad uključuje i bonus stručnosti koji odražava specifičnu vještinu lika.
2. uračunavanje dodatnih bonusa i kazni. Značajke klase, čarolije, specifične okolnosti ili neki drugi efekti mogu dodati bonus na rezultat ili ga umanjiti.
3. usporedba rezultata s ciljanim brojem. Ako je ukupan zbroj jednak ili veći od ciljnog broja, radnja (provjera, napad, spašavanje) je uspješna, u suprotnom, nije uspjela.

DM obično određuje ciljne brojeve i govori igračima jesi li uspjeli odraditi radnju. DM ne treba otkriti igračima ciljani broj za uspjeh.

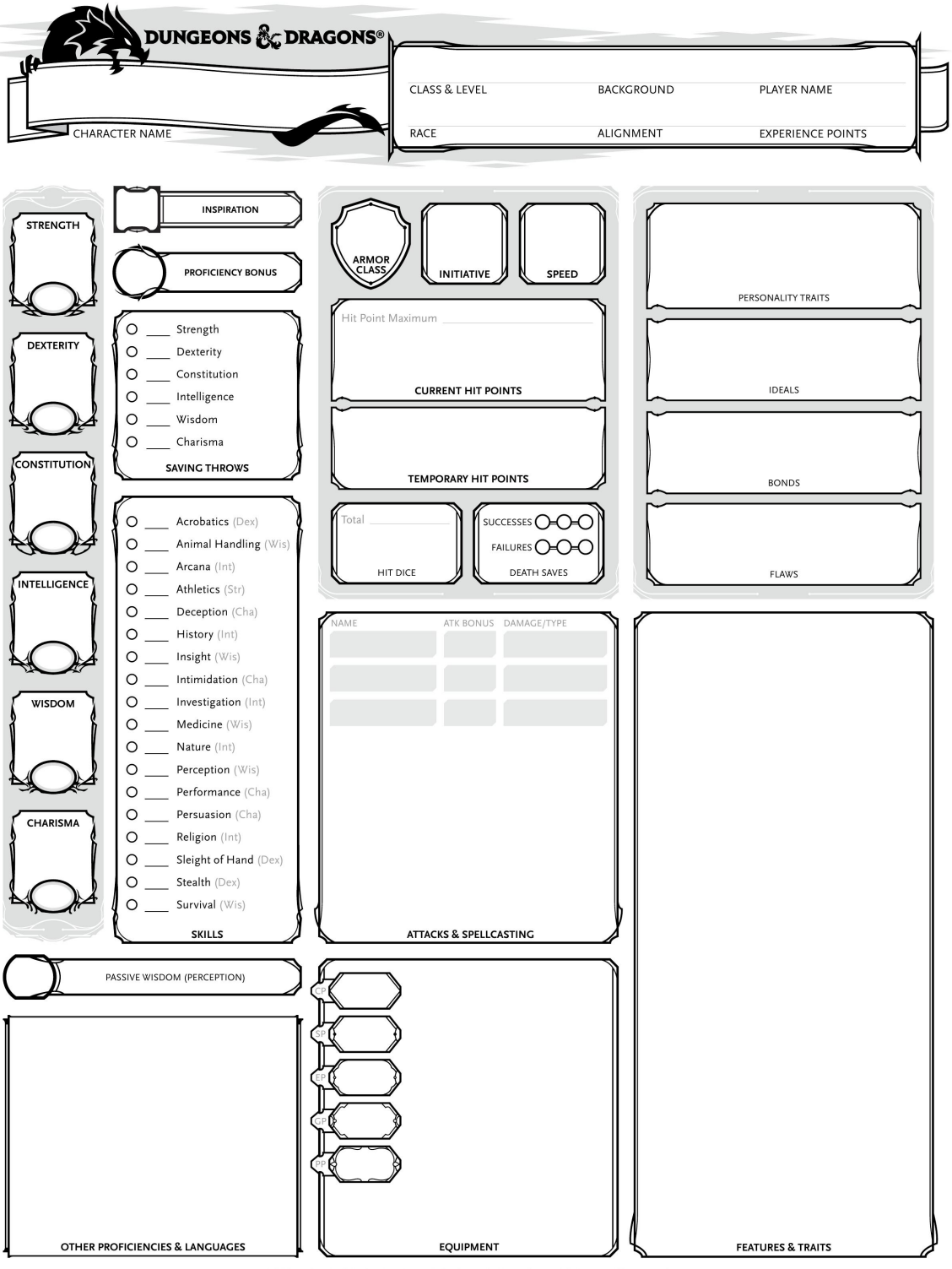
Ciljni broj za provjere sposobnosti i spas naziva se težina zadatka ili difficulty class (DC), dok se ciljni broj za napad naziva klasa oklopa ili Armour Class (AC). Ovo jednostavno pravilo temelj je rješavanja većine situacija u D&D-u.

Kod nekih provjera sposobnosti, bacanja za napad ili bacanja za spas bacanje kockice se modificira posebnom situacijom koju nazivamo prednost (engl. *advantage*) ili odsutnost prednosti (engl. *disadvantage*). U ovoj situaciji kod bacanja D20 kockice, baca se dodatna D20 i ovisno o situaciji, ako je prednost ili odsutnost prednosti odabire se kockica s najvećom vrijednošću ili kockica s najmanjom vrijednošću.

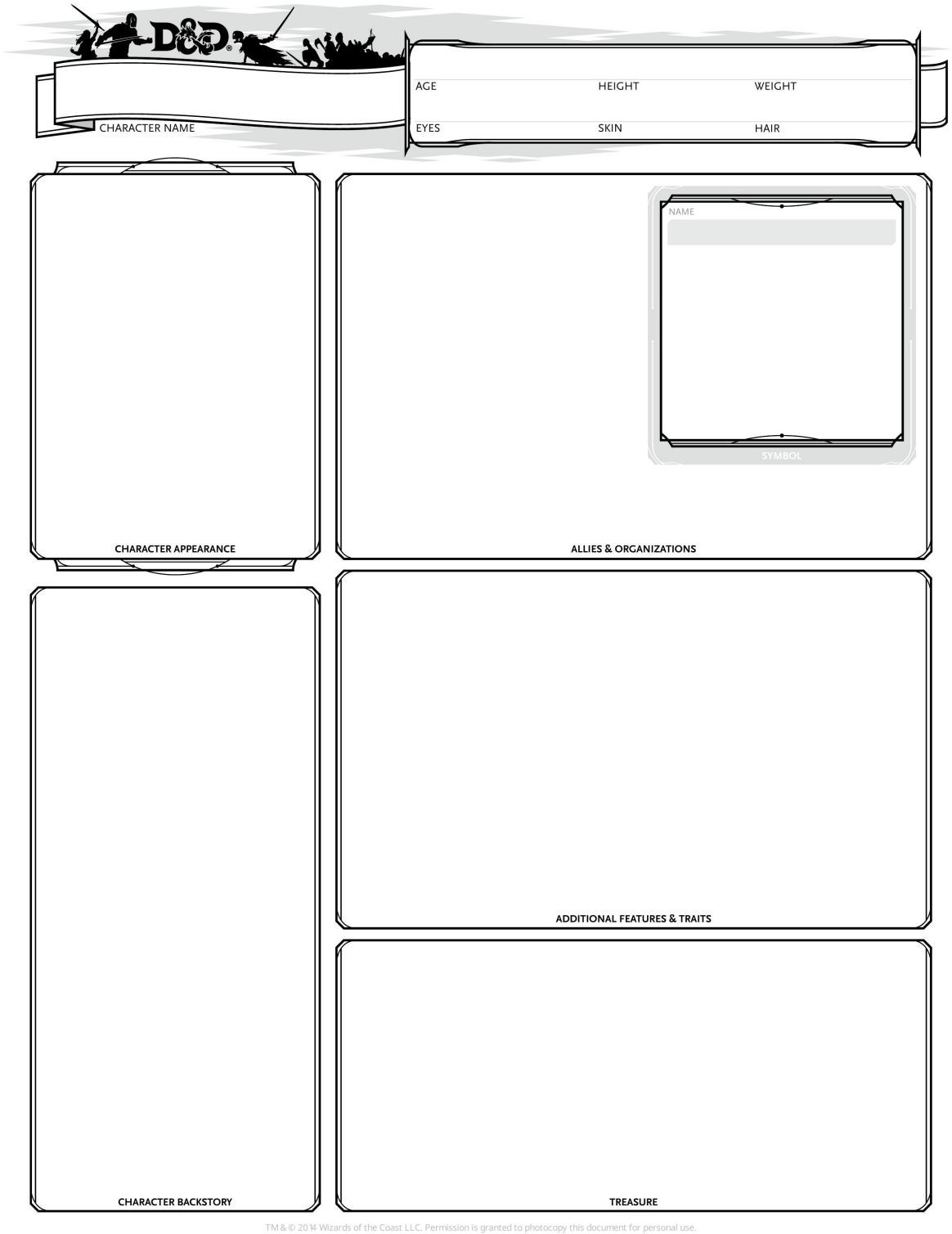
3.4. Kreacija lika

Kreiranje lika predstavlja odabir rase i klase te ostalih pojedinosti koje likovi time dobivaju kao i postupak kojim se dobivaju početne sposobnosti lika. Na slikama ispod nalaze se primjeri stranica o karakteristikama lika (engl. *character sheet*). Character sheet koristi igračima za zapisivanje i praćenje karakteristika njihovih likova.

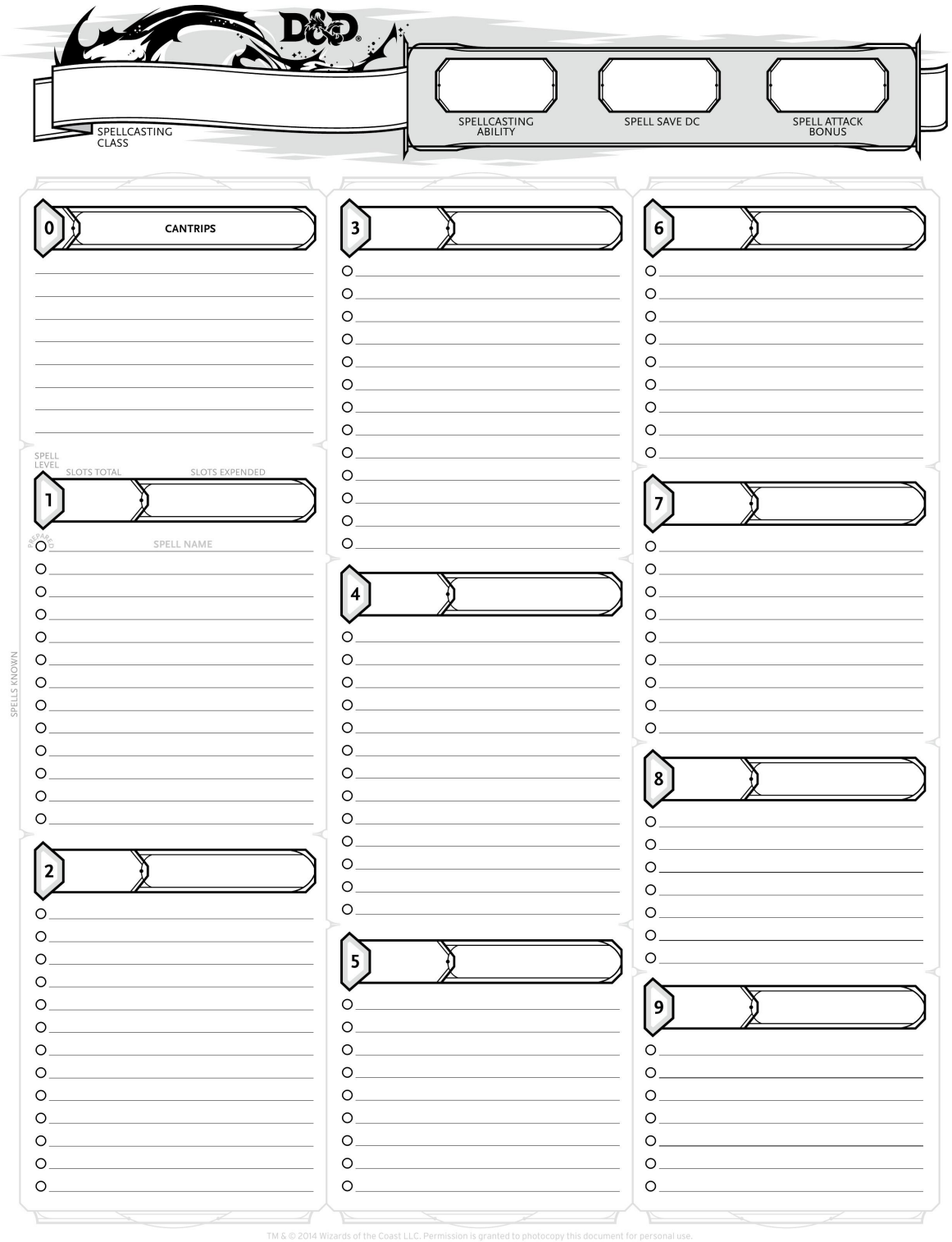
Character sheet se sastoji od tri stranice, prva stranica (Slika 1) sadrži podatke o sposobnostima lika, njegovu klasu, rasu, vještine, trenutnu opremu, životne bodove. Druga stranica (Slika 2) ima podatke o izgledu lika, starosti, visini, pozadinskoj priči, blagu. Omogućuje igraču detaljan opis svog lika, lakše preuzimanje uloge lika i gledanje fantastičnog svijeta kroz oči lika. Dok treća stranica (Slika 3) sadrži podatke o čarolijama pripremljenim i znanim čarolijama te razinama (engl. *levels*) i broj čarolija koje lik može baciti prije potrebnog odmora.



Slika 1: Character sheet - Prva stranica [7]



Slika 2: Character sheet - Druga stranica [7]



Slika 3: Character sheet - Treća stranica [7]

3.4.1. Rase

Svaki lik pripada rasi, jednoj od mnogih rasa dostupnih u D&D-u. Najčešće rase likova su patuljci, vilenjaci i ljudi. Neke rase također imaju podrase, kao što su planinski patuljci ili

šumski vilenjaci. Odabrana rasa doprinosi identitetu lika, uspostavlja opći izgled, pruža prirodne talente stečene kulturom i precima. Rasa daje određene rasne osobine, kao što su posebna osjetila, vještine s određenim oružjem ili alatima, stručnost u jednoj ili više vještina ili sposobnost korištenja manjih čarolija. Ove osobine se ponekad podudaraju i nadopunjuju sposobnostima određenih klasa. Najčešće rasne osobine su uvećanje jedne ili više sposobnosti, dob lika, moralni kompas, veličina, brzina, jezici koje lik razumije i kojima se može služiti i razgovarati te podrase; članovi podrase imaju osobine matične rase uz dodatne osobine definirane podrasom.

U nastavku će biti definirane različite rase, njihove podrase, dob, veličina, brzina i ostale sposobnosti koje se dobivaju odabirom pojedine rase. Kao primjer će za patuljke biti navedeni detaljni podaci o rasi, dok će druge rase imati skraćene podatke, razumijevanje svih različitih rasnih osobina nije nužno za implementaciju ni za razumijevanje igre:

- Patuljak (engl. *dwarf*) [5]
  - Povećanje sposobnosti: izdržljivost se povećava za 2.
  - Dob: patuljci sazrijevaju istom brzinom kao i ljudi, ali se smatraju mladima do 50. godine. Žive oko 350 godina.
  - Veličina: Srednja, patuljci su visoki između 4 i 5 stopa.
  - Brzina: brzina hoda je 25 stopa i ne smanjuje se nošenjem teškog oklopa
  - Dodatna svojstva rase:
    - Vid u mraku (engl. *darkvision*): patuljci vide u mraku i svijetlostno prigušenim uvjetima do 60 stopa.
    - Patuljačka otpornost (engl. *dwarven resilience*): prednost pri bacanju spasa protiv otrova i otpornost na otrov.
    - Patuljačka borbena obuka (engl. *dwarven combat training*): stručnost u korištenju bojne sjekire (engl. *battleaxe*), sjekirice (engl. *handaxe*), lakog čekića (engl. *throwing hammer*) i ratnog čekića (engl. *warhammer*).
    - Stručnost s alatom (engl. *tool proficiency*): stručnost u korištenju jedne vrsta alata: kovački alat, pivarski pribor, zidarski alat
    - Poznavanje kamena (engl. *stonecunning*): na sve provjere inteligencije (vezane uz vještinu: povijest), dodaje se dvostruki bonus stručnosti na provjeru.
  - Jezici: patuljci govore, pišu i čitaju zajednički (engl. *common*) i patuljački
  - Podrase patuljaka: brdski patuljak; povećanje mudrosti za 1 i maksimalni broj životnih bodova povećava se za 1 po razini. Planinski patuljak; povećanje snage za 2 i stručnost s lakim i srednjim oklopom.

- Vilenjak (engl. *elf*)
  - Povećanje sposobnosti: spretnost se povećava za 2.
  - Dob: vilenjaci sazrijevaju istom brzinom kao i ljudi, ali se smatraju odraslima tek oko 100. godine. Žive do 750 godina.
  - Veličina: Srednja, vilenjaci su visoki od ispod 5 do preko 6 stopa.
  - Brzina: brzina hoda je 30 stopa.
  - Tri podrase: visoki vilenjak (engl. *high elf*); Šumski vilenjak (engl. *wood elf*); Tamni vilenjak (engl. *drow*);
- Halfling
  - Povećanje sposobnosti: spretnost se povećava za 2.
  - Dob: halflinzi dosežu odraslu dob s 20 godina i žive do oko 150 godina.
  - Veličina: Mala, halflinzi su visoki oko 3 stope.
  - Brzina: brzina hoda je 25 stopa.
  - Jezici: govore, pišu i čitaju zajednički i halflingški (engl. *halfling*).
- Čovjek (engl. *human*)
  - Povećanje sposobnosti: sve sposobnosti se povećavaju za 1.
  - Dob: ljudi dosežu odraslu dob u kasnim tinejdžerskim godinama i žive manje od stoljeća.
  - Veličina: Srednja, ljudi znatno variraju u visini i građi.
  - Brzina: brzina hoda je 30 stopa.
  - Jezici: ljudi govore, pišu i čitaju zajednički i jedan dodatni jezik po izboru.
- Dragonborn
  - Povećanje sposobnosti: snaga se povećava za 2, a karizma za 1.
  - Dob: brzo rastu, odrasli su s 15 godina i žive oko 80 godina.
  - Veličina: Srednja, viši su i teži od ljudi, često preko 6 stopa.
  - Brzina: brzina hoda je 30 stopa.
  - Jezici: govore, pišu i čitaju zajednički i zmajski (engl. *draconic*).
- Gnom (engl. *gnome*)
  - Povećanje sposobnosti: inteligencija se povećava za 2.
  - Dob: gnomovi odrastaju oko 40. godine i žive 350 do 500 godina.
  - Veličina: Mala, visoki su između 3 i 4 stope.
  - Brzina: brzina hoda je 25 stopa.

  - Jezici: gnomovi govore, pišu i čitaju zajednički i gnomski (engl. *gnomish*).
  - Podvrste: šumski gnom (engl. *forest gnome*) Kameni gnom (engl. *rock gnome*).

- Poluvilenjak (engl. *half-elf*)
  - Povećanje sposobnosti: karizma se povećava za 2 i dvije druge sposobnosti po izboru za 1.
  - Dob: sazrijevaju kao ljudi, odrasli s 20 godina, žive preko 180 godina.
  - Veličina: Srednja, otprilike iste veličine kao ljudi.
  - Brzina: brzina hoda je 30 stopa.
  - Jezici: poluvilenjaci govore, pišu i čitaju zajednički, vilinski (engl. *elvish*) i jedan dodatni jezik po izboru.

- Poluork (engl. *half-orc*)
  - Povećanje sposobnosti: snaga se povećava za 2, a izdržljivost za 1.
  - Dob: sazrijevaju brže od ljudi s oko 14 godina, žive do 75 godina.
  - Veličina: Srednja, krupniji su od ljudi.
  - Brzina: brzina hoda je 30 stopa.
  - Jezici: poluorkovi govore, pišu i čitaju zajednički i orkovski (engl. *orc*).

- Tiefling
  - Povećanje sposobnosti: inteligencija se povećava za 1, a karizma za 2.
  - Dob: sazrijevaju kao ljudi, žive malo duže.
  - Veličina: Srednja, iste veličine i građe kao ljudi.
  - Brzina: brzina hoda je 30 stopa.
  - Jezici: tieflinzi govore, pišu i čitaju zajednički i pakleni (engl. *infernal*).

3.4.2. Klase

Klasa je temelj onoga što likovi mogu učiniti. Ona je više od pukog zanimanja; ona je životni poziv lika. Klasa lika oblikuje način na koji igrači doživljavaju svijet i kako s njim komuniciraju. Klasa daje raznoliko posebne značajke, poput borčeve vještine s oružjem i oklopom ili magovim čarolijama. Na nižim razinama, klasa nudi tek dvije ili tri značajke, no kako lik napreduje, stječe nove, a postojeće se često poboljšavaju [5]. U tablici ispod se nalaze nazivi klasa, opisi, kockice za životne bodove koje svaka klasa koristi kod određivanja životnih bodova po razini, primarne sposobnosti klase, sposobnosti koje se koriste kod bacanja za spas te stručnosti s oklopom i oružjem. Osim ovih sposobnosti svaka klasa ima specifične značajke no kroz njih nećemo prolaziti u ovom radu i dostupne su u `Player's Handbook`-u [5].

Tablica 1: Tablica klasa

Klasa	**Opis**	**Vrsta kockica za životne bodove**	**Primarna sposobnost**	**Bacanja za spas**	**Stručnosti s oklopom i oružjem**
Barbarin	Žestoki ratnik primitivnog podrijetla koji može ući u bojni bijes	D12	Snaga	Snaga, izdržljivost	Laki i srednji oklopi, štitovi, jednostavno i borbeno oružje
Bard	Nadahnjujući mađioničar čija moć odjekuje glazbom stvaranja	D8	Karizma	Spretnost, karizma	Laki oklop, jednostavno oružje, ručni samostreli, dugi mačevi, rapiri, kratki mačevi
Klerik	Svećenički prvak koji upravlja božanskom magijom u službi više sile	D8	Mudrost	Mudrost, karizma	Laki i srednji oklopi, štitovi, jednostavno oružje
Druid	Svećenik "Stare Vjere", upravlja moćima prirode i poprima životinjske oblike	D8	Mudrost	Inteligencija, mudrost	Laki i srednji oklopi (nemetalni), štitovi (nemetalni), toljage, bodeži, strelice, koplja, mlatovi, štapovi, sablje, srpovi, praćke
Borac	Majstor borbe, vješt s raznim oružjem i oklopima	D10	Snaga ili spretnost	Snaga, izdržljivost	Svi oklopi, štitovi, jednostavno i borbeno oružje
Redovnik	Majstor borilačkih vještina koji koristi snagu tijela za fizičko i duhovno savršenstvo	D8	Spretnost, mudrost	Snaga, spretnost	Jednostavno oružje, kratki mačevi
Paladin	Sveti ratnik vezan svetom zakletvom	D10	Snaga, karizma	Mudrost, karizma	Svi oklopi, štitovi, jednostavno i borbeno oružje
Rendžer	Ratnik koji koristi borilačku vještinu i magiju prirode za borbu protiv prijetnji	D10	Spretnost, mudrost	Snaga, spretnost	Laki i srednji oklopi, štitovi, jednostavno i borbeno oružje
Lupež	Nitkov koji koristi skrivanje i trikove kako bi svladao prepreke i neprijatelje	D8	Spretnost	Spretnost, inteligencija	Laki oklop, jednostavno oružje, ručni samostreli, dugi mačevi, rapiri, kratki mačevi
Čarobnjak	Bacač čarolija koji crpi urođenu magiju iz dara ili krvne loze	D6	Karizma	Izdržljivost, karizma	Bodeži, strelice, praćke, štapovi, laki samostreli
Vještac	Korisnik magije koja proizlazi iz pogodbe s izvanplanetarnim entitetom	D8	Karizma	Mudrost, karizma	Laki oklop, jednostavno oružje
Mag	Učeni korisnik magije sposoban manipulirati strukturama stvarnosti	D6	Inteligencija	Inteligencija, mudrost	Bodeži, strelice, praćke, štapovi, laki samostreli

(Izvor: Player's Handbook [5])

3.4.3. Ostale značajke

U ostale značajke lika ubrajamo ime, spol, visinu i težinu, opredjeljenje, jezike, osobne karakteristike (osobine, ideale, veze, mane) i pozadinu. Značajke opisuju način na koji bi se lik mogao ponašati tijekom igre. Značajke kao ime, spol, visinu, težinu i osobne karakteristike igrač sam kreira kako bi pobliže opisao svog lika i njegovo ponašanje u svijetu, dok se značajke kao što su opredjeljenje, pozadina mogu birati (ili igrači/DM mogu kreirati pozadine) te će u nastavku ukratko biti opisani.

Opredjeljenje u opisuje likove moralne i osobne stavove te je kombinacija dvaju faktora: jedan identificira moralnost (dobar, zao, neutralan), a drugi opisuje odnos prema društvu i redu (zakonit, kaotičan, neutralan). Svaka kombinacija ovih dva faktora ukupno definira devet mogućih kombinacija. Ovih devet opredjeljenja opisuju tipično ponašanje s tim opredjeljenjem, pojedinci mogu značajno odstupati od tog tipičnog ponašanja [5]:

- Zakonito dobar (engl. *Lawful good*) - pojedinci će učiniti ispravnu stvar onako kako to društvo očekuje
- Neutralno dobar (engl. *Neutral good*) - čine najbolje što mogu kako bi pomogli drugima u skaldu sa svojim potrebama
- Kaotično dobar (engl. *Chaotic good*) - djeluju kako im savjest nalaže, s malo obzira prema onome što drugi očekuju
- Zakonito neutralan (engl. *Lawful neutral*) - pojedinci djeluju u skaldu sa zakonom, tradicijom ili osobnim kodeksima
- Neutralan (engl. *Neutral*) - opredjeljenje onih koji radije izbjegavaju moralna pitanja i ne zauzimaju strane, čineći ono što se u tom trenutku čini najboljim
- Kaotično neutralan (engl. *Chaotic neutral*) - slijede svoju volju, držeći svoju osobnu slobodu iznad svega
- Zakonito zao (engl. *Lawful evil*) - uzimaju ono što žele, unutar granica kodeksa tradicije, lojalnosti ili reda
- Neutralno zao (engl. *Neutral evil*) - opredjeljenje onih koji čine sve što im može proći nekažnjeno, bez suosjećanja ili grižnje savjesti
- Kaotično zao (engl. *Chaotic evil*) - djeluju s proizvoljnim nasiljem, potaknuti pohlepom, mržnjom ili željom za krvlju

Svaka priča ima neki početak te pozadina lika otkriva odakle lik dolazi, kako je postao pustolov i koje je njegovo mjesto u svijetu. Odabir pozadine pruža važne smjernice o identitetu lika te daje liku dodatne značajke kao stručnost u nekoj vještini ili znanje nekog jezika. U samom primjeru izrade lika će biti detaljno opisana jedna pozadina te sve dodatne značajke koje lik dobiva i opremu s kojom počne. Kratki popis pozadina [5]:

- Akolit (engl. *Acolyte*) - osoba koja je provela život u službi hrama određenog boga, djelujući kao posrednik između svetog i smrtnog svijeta.
- Šarlatan (engl. *Charlatan*) - osoba koja zna manipulirati ljudima i koristi trikove, prevare i lažne identitete za vlastitu korist.
- Kriminalac (engl. *Criminal*) - iskusni prijestupnik s poviješću kršenja zakona i s kontaktima u podzemlju.
- Zabavljač (engl. *Entertainer*) - osoba koja živi za nastup pred publikom i može ih zabaviti, očarati ili inspirirati.
- Narodni heroj (engl. *Folk hero*) - osoba iz skromnog društvenog položaja, kojeg ljudi smatraju svojim prvakom protiv tiranije i čudovišta.
- Pustinjak (engl. *Hermit*) - osoba koja živi u samoći, pronalazeći tišinu i odgovore daleko od društva.
- Plemić (engl. *Noble*) - osoba koja nosi plemićku titulu, posjeduje zemlju ili ima politički utjecaj i razumije bogatstvo i moć.
- Mudrac (engl. *Sage*) - osoba koja je provela godine učeći o znanju svijeta i proučavajući rukopise.
- Mornar (engl. *Sailor*) - osoba koja je godinama plovila na morskom plovilu, suočavajući se s olujama i čudovištima.
- Vojnik (engl. *Soldier*) - osoba kojoj je rat bio život, obučena u borbi i preživljavanju na bojnom polju.
- Uličar (engl. *Urchin*) - osoba koja je odrasla sama na ulici, siromašna i bez roditelja, preživljavajući zahvaljujući svojoj snalažljivosti.

3.5. Primjer lika

U nastavku prolazimo kroz kreaciju lika po pravilima definiranim u prijašnjim poglavljima. Lik će početi na prvoj razini.

Kreacija počinje s odabirom rase. Za rasu odabiremo poluorka, što znači da će lik imati povećanu snagu i izdržljivost, snagu za dva boda, izdržljivost za jedan, brzinu od 30 stopa te dobiva osobine vid u mraku (engl. *darkvision*) i prijeteću prisutnost (engl. *menacing*). Prijeteća prisutnost omogućava stručnost u vještini zastrašivanja/prijetnje (engl. *intimidation*). Još dodatne dvije sposobnosti koje daje poluork rasa su nemilosrdna izdržljivost (engl. *relentless endurance*); kad je liku broj životnih bodova smanjen na nulu, nije odmah ubijen, već pada na jedan životni bod i ne može opet koristiti ovu sposobnost dok ne završi dugi odmor. Druga je divljački napadi (engl. *savage attacks*); kad lik postigne kritični pogodak napadom oružja u bliskoj borbi, može bacati još jednu dodatnu kockicu od kockica štete oružja i dodati vrijednost šteti kritičnog pogotka.

Za klasu odabiremo barbarina, što znači da će lik koristiti kockicu D12 za akcije vezane uz njegove životne bodove, npr. kod prelaska na višu razinu, ili kod kratkog/dužeg odmora. Lik dobiva stručnosti s lakim i srednjim oklopima, štitovima, jednostavnim i borbenim oružjima. Kod bacanja spasa koristi snagu i izdržljivost te odabira stručnosti u dvije vještine iz područja rukovanja životinjama (engl. *animal handling*), atletika (engl. *athletics*), zastrašivanja, prirode (engl. *nature*), percepcije (engl. *perception*) i preživljavanja (engl. *survival*). Za ovog lika odabiremo stručnost u vještinama rukovanja životinjama i percepcije te je lik također vješt u zastrašivanju zbog odabrane rase. Jedinstvena sposobnost barbarina je bijes. Tokom bitke barbarin može pobjesniti korištenjem bonus akcije. Dok bjesni dobiva dodatne pogodnosti, ako ne nosi teški oklop; ima prednost na provjere snage i bacanja za spašavanje snage; kad napada oružjem za blisku borbu, koje koristi snagu, dobiva bonus na bacanje štete; ima otpornost na neke vrste štete, otpornost na tučnjavu (engl. *bludgeoning*), na probijanje (engl. *piercing*) i na sječenje (engl. *slashing*). Zadnja sposobnost koju lik dobiva od klase s prvom razinom je neoklopljena obrana (engl. *armored defense*); dok lik ne nosi oklop, klasa oklopa (AC) je jednaka 10 + modifikator spretnosti + modifikator izdržljivosti [5]. Na višim razinama klase lik dobiva još više mogućnosti no ovdje nećemo proći kroz njih.

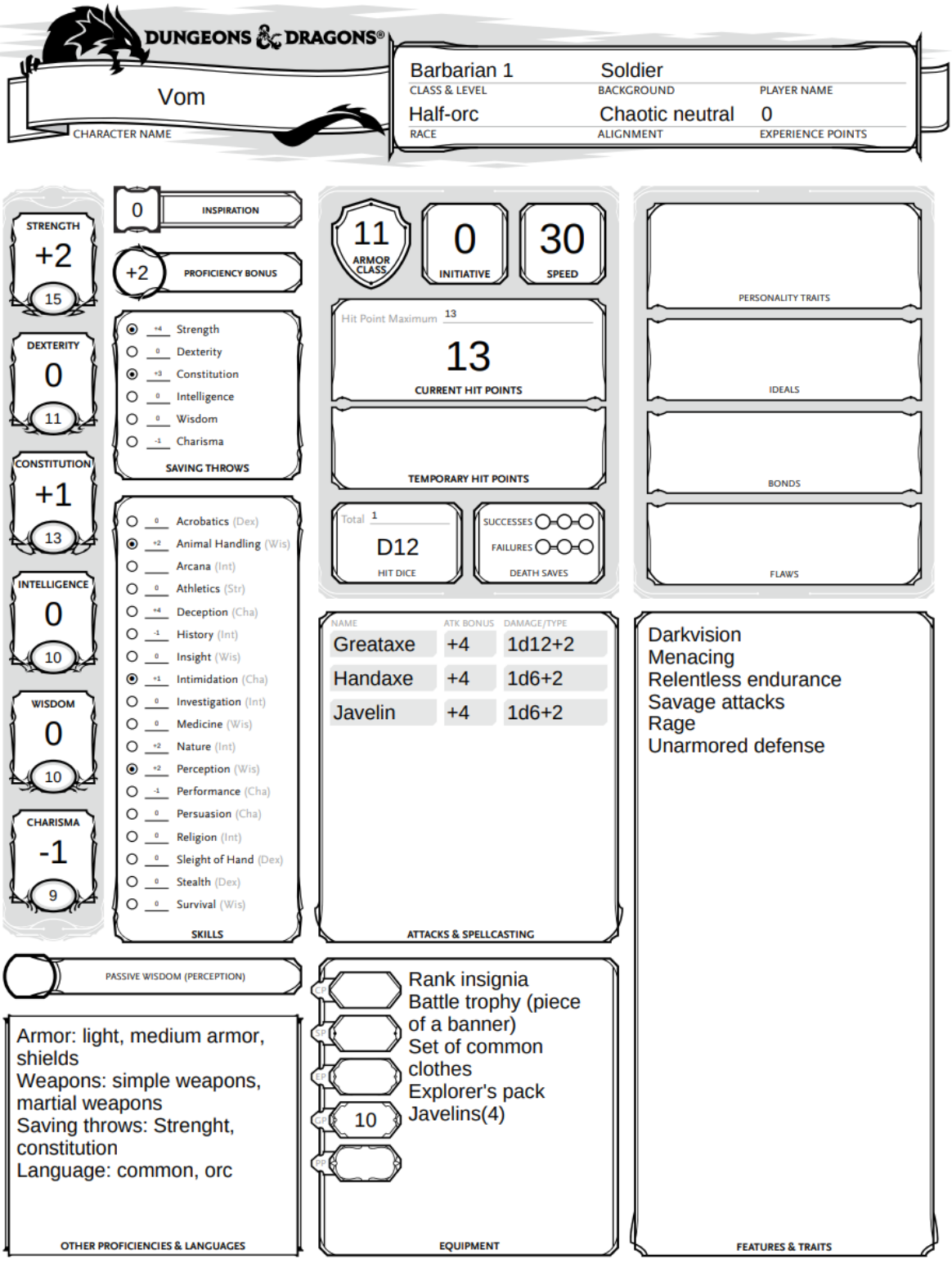
Slijedeće je potrebno odrediti vrijednost i modifikator svake sposobnosti i maksimum životnih bodova. Postoji više načina na koji se mogu odrediti početne vrijednosti za sposobnosti, no u *Player's Handbook-u* je preporučeno da se šest puta bace četiri D6 kockice te se kod svakog bacanja izbaci najmanja kockica. Zbroj triju kockica određuje vrijednost jedne od sposobnosti te nakon šest bacanja igrač može proizvoljno rasporediti vrijednosti po sposobnostima. Modifikatori za sposobnosti se isčitavaju iz slijedeće tablice [5]:

Tablica 2: Vrijednosti sposobnosti i modifikatori

Vrijednost	**Modifikator**	**Vrijednost**	**Modifikator**
1	-5	16-17	+3
2-3	-4	18-19	+4
4-5	-3	20-21	+5
6-7	-2	22-23	+6
8-9	-1	24-25	+7
10-11	0	26-27	+8
12-13	+1	28-29	+9
14-15	+2	30	+10

(Izvor: Player's Handbook [5])

Maksimalni broj životnih bodova na prvoj razini određuje vrsta klase, pošto je ovaj lik barbarin, broj životnih bodova će iznositi 12 + modifikator izdržljivosti. Ispod se nalazi slika s primjerom popunjenog character sheet-a za ovog lika, neki dijelovi su popunjeni, a nisu ovdje opisani, pošto su manje važni za razumijevanje igre i direktno se isčitavaju iz pravila (npr. oprema koju lik dobiva).



Slika 4: Primjer popunjenog character sheet-a

3.6. Borba

Ovo poglavlje pruža pravila koja su potrebna kako bi likovi i čudovišta sudjelovali u borbi, bilo da se radi o kratkom okršaju ili produženom sukobu. Kroz cijelo poglavlje pravila se odnose na igrača ili DM-a. DM kontrolira sva čudovišta i likove koji nisu igrači, a uključeni su u borbu, drugi igrači kontroliraju svoje likove.

Tipičan sukob je između dvije strane i organiziran je u ciklus rundi i poteza. Runda predstavlja oko šest sekundi u svijetu igre. Tijekom runde, svaki sudionik u bitci ima svoj potez. Redoslijed poteza određuje se na početku borbenog susreta, kada svi bacaju inicijativu. Nakon što svi odrade potez, borba se nastavlja u slijedeću rundu ako nijedna strana nije pobijedila drugu.

Prije početka borbe DM određuje tko bi mogao biti iznenađen na početku borbe. Ako nijedna strana ne pokušava biti prikrivena automatski primjećuju jedna drugu i ovaj korak se preskače, no ako se jedna strana uspješno prikrade ostvaruje prednost. Strana koja je iznenađena, tj. lik ili čudovište koje je iznenađeno se ne može pomaknuti ili poduzeti akciju u svom prvom potezu borbe, također ne može poduzeti reakciju dok taj prvi potez ne završi. Član grupe može biti iznenađen čak i ako ostali članovi nisu [5].

Inicijativa određuje redoslijed poteza tokom borbe. Kada borba počne svaki sudionik radi provjeru spretnosti (engl. *dexterity check*) kako bi odredio svoje mjesto u poretku inicijative. Baca D20 kockicu i na dobivenu vrijednost dodaje modifikator spretnosti. DM određuje inicijativu za sve likove, čudovišta koje ne kontroliraju drugi igrači. Nakon dobivenih vrijednosti za inicijativu, DM ih rangira od najviše do najmanje vrijednosti te određuje redoslijed poteza. U slučajevima kad igrači imaju istu inicijativu mogu međusobno odrediti tko nastupa prvi, ako čudovište i igrač imaju jednaku inicijativu DM određuje redoslijed.

3.6.1. Igračev potez

Kad je igrač na potezu može pomicati svog lika do udaljenosti određenom brzinom lika i odraditi jednu akciju. U pravilu svako polje je veličine pet sa pet stopa, što bi značilo da se lik s brzinom od trideset stopa može pomaknuti maksimalno šest polja. Igrač odlučuje da li će se prvo kretati ili odraditi akciju. U poglavlju 3.6.3 će biti opisane najčešće akcije koje likovi mogu odraditi, mnoge značajke različitih klasa i druge sposobnosti koje pružaju dodatne opcije za akcije.

Razne značajke klase, čarolije i druge sposobnosti omogućavaju poduzimanje dodatne akcije u trenutnom potezu. Dodatna akcija se može jedino poduzeti u slučaju kad značajka klase, čarolija ili neka druga sposobnost navodi da se nešto može odraditi kao dodatna akcija. Po potezu, jednako kao s akcijom, može se odraditi samo jedna dodatna akcija te ako lik ima dostupno više dodatnih akcija potrebno je odrediti koju je najbolje iskoristiti u tom slučaju [5, str. 189].

Kao dio poteza lik može stupiti u interakciju s jednim predmetom ili značajkom okoline, ta interakcija se može dogoditi tijekom kretanja ili akcije. Na primjer, otvaranje vrata dok se

lik kreće prema neprijatelju ili izvlačenje oružja kao dio iste akcije za napad bi spadale pod interakciju s okolinom.

Određene posebne sposobnosti, čarolije i situacije omogućuju poduzimanje posebne akcije zvane reakcija. Reakcija je trenutni odgovor na okidač neke vrste, koji se može dogoditi tokom poteza igrača ili tuđeg poteza. Napad prilike (engl. *Opportunity attack*) je najčešća vrsta reakcije i više je opisana u poglavlju 3.6.4.

3.6.2. Kretanje i položaj

Kad je lik na potezu može se pomaknuti do udaljenosti definirane brzinom lika. Po potezu lik može potrošiti proizvoljnu količinu brzine, ne mora potrošiti svu ili uopće iskoristiti brzinu. Kretanje se može razdijeliti kroz cijeli potez. Moguće je koristiti dio brzine prije i poslije akcije. Na primjer, ako je brzina lika trideset stopa, može se kretati deset stopa, iskoristiti neku akciju, a zatim se kretati još dvadeset stopa. Ako akcija uključuje više od jednog napada oružjem, kretanje se može dodatno razdvojiti između tih napada.

Borbe se rijetko odvijaju u praznim sobama. Špilje pune kamenja i guste šume su tipična mjesta odvijanja borbe i sadrže težak teren (engl. *difficult terrain*). Za svako kretanje kroz ovakav teren potrebno je utrošiti jednu dodatnu stopu za svaku stopu koju lik želi preći. Na primjer, ako se radi o udaljenosti od pet stopa, za prolaz kroz težak teren potrebno je utrošiti deset stopa brzine [5, str. 190].

Likovi se mogu kretati kroz prostor prijateljskih stvorenja, ali ne mogu završiti potez ili zaustaviti se na istom polju. Kroz prostor neprijateljskih stvorenja mogu se kretati samo ako je stvorenje barem dvije veličine veće ili manje od lika koji se kreće, no nije dozvoljeno završavanje kretanja u prostoru koji zauzima to stvorenje [5, str. 191].

U tablici ispod se nalaze veličine stvorenja i različite količine prostora koje zauzimaju, tokom borbe veća stvorenja kontroliraju veću količinu prostora, također objekti u okolini ponekad mogu koristiti kategorije veličine. Prostor stvorenja je područje u stopama koje to stvorenje kontrolira tokom borbe, a nije izraz njihovih fizičkih dimenzija.

Tablica 3: Kategorije veličine

Veličina	**Prostor**
Sićušan	2.5 sa 2.5 stopa
Mala	5 sa 5 stopa
Srednja	5 sa 5 stopa
Velika	10 sa 10 stop
Ogromna	15 sa 15 stopa
Gigantska	20 sa 20 stopa ili više

(Izvor: Player's Handbook [5, str. 191])

3.6.3. Akcije

Kad je lik na potezu i želi odraditi neku akciju, osim akcija koje dobiva od klase ili nekih drugih posebnih značajki uvijek može odraditi neku od slijedećih akcija ili improvizirati akciju. Ako je akcija improvizirana DM odlučuje ako je moguća, koju vještinu lik može iskoristiti za akciju i kakvo bacanje kockice će odrediti uspješnost akcije.

Napad je najčešća akcija u borbi, bilo da se zamahuje mačem, ispaljuje strijela iz luka ili tučnjava šakama, ovom akcijom izvodi se jedan napad izbliza ili napad na daljinu [5, str. 192]. Napadi će biti detaljnije opisani u poglavlju 3.6.4.

Bacanje čarolija, mnoge vrste čudovišta i likova imaju pristup čarolijama i mogu ih koristiti u borbi. Svaka čarolija ima definirano vrijeme bacanja (engl. *cast time*), koje određuje mora li bacač koristiti akciju, reakciju, minute ili sate vremena da baci čaroliju. U nekim slučajevima, kao kad čarolija ima vrijeme bacanja od više sati, bacanje čarolije nije nužno jedna akcija. Većina čarolija ima vrijeme bacanja od jedne akcije, pa je često takva čarolija korištena tokom borbe [5, str. 192].

Trk (engl. *dash*), omogućava liku da potroši akciju kako bi se mogao na istom potezu dodatno kretati. Povećanje je jednako brzini lika, nakon što se primjene bilo kakvi modifikatori brzine. Na primjer, s brzinom od trideset stopa, na istom potezu moguće je pomaknuti se šestdeset stopa ako se iskoristi ova akcija. Bilo kakve promjene na brzinu također imaju utjecaj na dodatnu brzinu dobivenu ovom akcijom, na primjer ako je lik usporen i ima smanjenu brzinu s trideset na petnaest, korištenjem ove akcije će dobiti dodatnih petnaest brzine [5, str. 192].

Isključivanje iz borbe (engl. *disengage*), akcija omogućuje kretanje bez da neprijatelji dobivaju napad prilike kad se lik kreće oko njih [5, str. 192].

Izmicanje (engl. *dodge*), akcija omogućava usredotočenost na izbjegavanje napada. Do početka slijedećeg poteza, svako bacanje napada protiv lika koji je iskoristio ovu akciju, ima odsutnost prednosti ako lik vidi napadača, a sva bacanja za spas na spretnost se izvode s prednošću [5, str. 192].

Pomoć, likovi mogu pružiti pomoć drugom liku u izvršavanju nekog zadatka. Korištenjem ove akcije drugi lik dobiva prednost na slijedeću provjeru sposobnosti koju izvrši za obavljanje zadatka u kojem dobiva pomoć, pod uvjetom da napravi tu radnju prije nego pomagač opet dođe na potez. Također moguće je pomoći prijateljskom stvorenju u napadu na neprijatelja unutar pet stopa, napadač na svom potezu dobiva prednost na prvo bacanje za napad [5, str. 192].

Pretraga, akcija može biti iskorištena za pretragu okoline i pronalaženju nečega. Ovisno o prirodi pretrage, DM može zatražiti da se napravi provjera vještine percepcija ili istraga [5, str. 193].

3.6.4. Napadi

Napade možemo podijeliti u nekoliko vrsta, napadi na blizinu, na daljinu i napadi prilike. Svaka vrsta napada se ima razlike no svi imaju jednaku strukturu i način na koji se izvode [5, str. 193-194]:

1. Odabir mete; potrebno je odabrati metu unutar dometa napada, meta može biti stvorenje, predmet ili lokacija. Na primjer, lokacija se može odabrati ako pokušavamo napasti nevidljivog neprijatelja ili ako čarolija koju bacamo ima utjecaj na veće područje.

2. Određivanje modifikatora; DM određuje ako se meta skriva iza zaklona i ako napadač ima prednost ili nema prednost protiv mete. Uz to, čarolije, posebne sposobnosti i drugi učinci mogu imati utjecaja.

3. Razriješivanje napada; igrač radi bacanje za napad (engl. *attack roll*), ovo bacanje određuje ako je meta uspješno pogođena napadom ili je napad promašio. U slučaju pogotka određuje se šteta koja je napravljena meti, također bacanjem kockice, ovisno o napadu koriste se različite kockice, neki napadi uzrokuju posebne učinke uz štetu ili umjesto nje.

Za izvođenje bacanja za napad koristi se D20 kockica te se na dobivenu vrijednost dodaju određeni modifikatori. Ako je ukupan zbroj bacanja i modifikatora jednak ili veći od klase oklopa mete, u tom slučaju napad pogađa. Dva najčešća modifikatora koji se dodaju su modifikator sposobnosti (engl. *ability modifier*) i bonus stručnosti lika. Modifikator sposobnosti odgovara sposobnosti koja se koristi za napad, npr. može biti snaga ili spretnost ovisno o vrsti oružja. Bonus stručnosti se dodaje ako lik ima stručnost u korištenju oružja kojim napada.

U rjeđim slučajevima kao rezultat bacanja D20 moguće je dobiti 1 ili 20. Ako je dobivena vrijednost 20, napad je uvijek pogodak, neovisno o bilo kakvim modifikatorima ili klasi oklopa mete, također ovakav napad je uvijek kritičan pogodak. Ako je dobivena vrijednost 1, napad uvijek promašuje i ne dodaje se modifikator ni ne provjerava klasa oklopa mete. Slična funkcija je kod ostalih bacanja D20, vrijednost od 20 je skoro uvijek uspjeh dok je vrijednost 1 neuspjeh.

3.6.5. Šteta i iscjeljivanje

Ozljede i rizik od smrti stalni su pratitelji onih koji istražuju svjetove D&D-a. Životni bodovi predstavljaju kombinaciju fizičke i mentalne izdržljivosti. Stvorenje s više životnih bodova teže je ubiti ili onesvijestiti. Ona s manje životnih bodova su krhkija. Trenutni životni bodovi stvorenju mogu biti bilo koji broj između nule i maksimuma životnih bodova. Kad stvorenje pretrpi štetu, ta se šteta oduzima od životnih bodova, gubitak životnih bodova nema utjecaj na sposobnosti stvorenja sve dok je vrijednost veća od nule.

Ako šteta ne rezultira smrću stvorenja, tada nije trajna. Likovi mogu vratiti životne bodove odmorom, korištenjem čarolija ili ispijanjem ljekovitog napitka. Kada stvorenje primi iscje-

ljivanje bilo koje vrste, vraćeni životni bodovi dodaju se trenutnim životnim bodovima i ne mogu premašiti maksimum životnih bodova tog lika ili stvorenja.

Postoji nekoliko situacija koje se mogu dogoditi kad neki lik ili stvorenje padne na nula životnih bodova. Manje važna stvorenja, čudovišta uglavnom umiru odmah nakon što im životni bodovi padnu na nulu, dok važniji likovi mogu pasti u nesvijest ili imati nekoliko pokušaja da se spase od smrti i stabiliziraju svoje stanje. U slučaju kad količina štete svede lika na nula životnih bodova i ostatak štete je veći ili jednak broju maksimalnih životnih bodova lika, lik umire u tom trenutku. Padom u nesvijest tijekom borbe znači da lik ili stvorenje na svom potezu mora izvršiti bacanje za spas od smrti (engl. *death save*). Za spas se koristi D20 i sve vrijednosti jednake ili veće od deset približavaju lika životu, dok sve ispod deset smrti. Na treći uspjeh lik postaje stabilan te se više ne mora spašavati od smrti, dok na treći neuspjeh lik umire. Bacanje dvadeset broji se kao dva uspijeha, a bacanje jedan se broji kao dva neuspjeha. Stabilizirano stvorenje ima nula životnih bodova i onesviješteno je, odmorom se životni bodovi mogu povratiti, no odmor traje neko vrijeme i nije moguć tokom borbe. Tokom borbe iscjeljivanje na barem jedan životni bod je jedini način za osvijestiti lika ili stvorenje.

4. Prikaz implementacije

U ovom poglavlju će biti opisani alati i tehnologije korištene za implementaciju platforme. Korišteno je više različitih tehnologija, fokus je na odabir i korištenje tehnologija koje su što više prikladne za implementacije web aplikacija. Ovime se osigurava da je aplikacija dostupna, funkcionalna i prenosiva preko više različitih operativnih sustava i web preglednika.

Platforma je implementirana na način da može biti korištena kao zasebna web aplikacija koja se spaja na udaljeni poslužitelj ili kao desktop aplikacija koja automatski u pozadini pokreće poslužitelj i omogućava korisniku korištenje aplikacije bez posebnog pokretanja i postavljanja poslužitelja. Desktop i mobilna integracija je moguća kroz korištenje tauri programskog okvira.

Tauri je programski okvir za izgradnju malih, brzih programa za sve desktop i mobilne platforme. Aplikacija može integrirati bilo koji frontend okvir koji se kompajlira u HTML, javascript, css i koristiti ove tehnologije za razvoj korisničkog sučelja, a istovremeno može koristiti jezike poput rusta, swifta, jave i kotlina za backend logiku i interakciju s hardverom [8]. U ovom projektu tauri služi za pakiranje projekta u jednu izvršnu datoteku te jednostavno postavljanje i pokretanje poslužitelja i jednog klijenta.

Aplikacija se sastoji od dva odvojena dijela, to su poslužitelj i klijent. Poslužitelj služi za spremanje podataka, spajanje i sinkronizaciju podataka između povezanih klijenata te učitavanje slika, kako bi bile dostupne klijentima preko web API-a, te kreiranje manjih sličica (engl. *thumbnail*) iz učitanih slika, da klijenti ne bi trebali preuzimati cijele slike koje mogu biti veće rezolucije, a nikada neće biti potrebna takva rezolucija. Klijent sadrži svo korisničko sučelje i logiku igre, komunicira kroz poslužitelja s ostalim klijentima te prima promjene o stanju igre od poslužitelja.

Cijela aplikacija je umrežena, sve promjene koje se događaju kod jednog klijenta budu sinkronizirane i prikazane kod ostalih klijenata koji su povezani u istu sesiju.

4.1. Web aplikacije

Web aplikacija je program klijent-poslužitelj arhitekture koji koristi web preglednik kao svog klijenta. Web aplikacija obavlja interaktivnu uslugu povezivanja s poslužiteljem putem interneta ili intraneta [9].

Razlikuje se od tradicionalne web stranice na slijedeće načine [9]:

- **Dinamičko naspram statičkog**: dok web stranica, u većini slučajeva isporučuje sadržaj statičkih datoteka, web aplikacija prikazuje dinamički prilagođen sadržaj na temelju parametara zahtjeva, korisničke sesije.
- **Interaktivnost**: web aplikacije omogućuju korisnicima obavljanje specifičnih zadataka, poput kupnje robe ili upravljanje zalihama.

Web aplikacije se temelje na distribuiranoj arhitekturi koja se sastoji od tri glavna ele-

menta. Prvi element je klijent još nekad i nazivan frontend; to je obično web preglednik instaliran na korisnikovom računalu. Preglednik je odgovoran za prikazivanje sadržaja i rukovanje interakcijama korisnika. Drugi element je poslužitelj, ponekad zvan backend; ovaj element sluša dolazne zahtjeve od klijenata i generira odgovore. Obrađuje zadatke poput posluživanja statičkih datoteka, autentifikacije i prosljeđivanja dinamičkih zahtjeva aplikacijskim programima. Treći element je baza podataka; web aplikacije često zahtijevaju pristup trajnim podacima. Povezuju se s bazama podataka kako bi pohranile i dohvaćale informacije. Arhitektura aplikacije uključuje specifičnu logiku za omogućavanje pristupa podacima i upravljanje transakcijama [10].

4.2. Poslužitelj

Poslužitelj je implementiran u programskom jeziku rust. Rust je sistemski programski jezik sa statičnim tipovima podataka, dizajniran za performanse i sigurnost. Razvijen od strane Mozilla Research, stvoren je kao odgovor na dugogodišnju napetost između kontrole na niskoj razini koju pruže jezici poput C i C++-a i sigurnosti jezika više razine poput jave ili pythona. Rust sigurnost postiže sustavom koji se zove vlasništvo (engl. *ownership*) i provjeravanjem vlasništva nad varijablama tokom kompajliranja programa sustavom zvanim provjeravač posuđenog (engl. *borrow checker*). Korištenjem ova dva sustava osigurava korektan pristup i dijeljenje memorije, izbjegava probleme s korištenjem izbrisane memorije, čitanja ili pisanje u memoriju koju više dretva istovremeno koristi, a pristup memoriji nije sinkroniziran [11].

Budući da rust nema sakupljača smeća (engl. *garbage collector*) i ima minimalan runtime, nevjerojatno je brz i memorijski učinkovit. Kod kompajliranja programa koristi apstrakcije bez troška, što znači da programer može koristiti koncepte programiranje više razine, poput iteratora ili generičnih funkcija, bez da računalo obavlja dodatni posao tijekom izvođenja, svi koncepti više razine tokom kompajliranja su pretvoreni u nižu razinu, na primjer iteratori su pretvoreni u jednostavne for petlje, dok su generične funkcije pretvorene u više različitih funkcija koje kompajlirani program poziva [11].

4.2.1. Protokoli za prijenos podataka između poslužitelja i klijenta

Implementirani poslužitelj koristi http i websockete za komunikaciju s klijentima. HTTP je protokol na kojim se temelji world wide web, dizajniran je za prijenos specijaliziranih poruka putem mreže. Koristi TCP kao svoj temeljni transportni sloj, što osigurava da su svi podaci preneseni u točnom redoslijedu i primljeni. Komunikacija http-a koristi model zahtjeva i odgovora. HTTP klijent poput web preglednika započinje komunikaciju slanje poruke zahtjeva prema http poslužitelju. Poslužitelj zatim zaprima zahtjev, obrađuje ga i generira poruku odgovora koju šalje natrag klijentu. Na ovaj način klijent može zatražiti bilo kakvu vrstu resursa od poslužitelja [10].

HTTP zahtjev se sastoji od nekoliko dijelova i počinje s metodom zahtjeva, putanju do resursa i broj verzije http-a, slijedeći redovi su popis zaglavlja, nakon zaglavlja je prazni redak iza kojeg se u poruku dodaje tijelo. Tijelo je opcionalno i ne mora biti prisutno u poruci. Ispod

je primjer zahtjeva, korištena je metoda GET, pokušava se dohvatiti resurs index.html, verzija HTTP-a je 1.1 i u zahtjev uključeno je jedno zaglavlje, tijelo je prazno. Ovakvi zahtjevi su uglavnom slani od klijenta prema poslužitelju, no nije neubičajeno da dva poslužitelja komuniciraju preko http-a te u ovom slučaju jedan poslužitelj preuzima ulogu klijenta.

```
GET /index.html HTTP/1.1
User-Agent: Mozilla/5.0(X11;Linux x86_64) Chrome/143.0.0.0
```

Klijent od poslužitelja prima odgovor na zahtjev. Struktura odgovora počinje statusnom linijom, koja sadrži verziju http-a, troznamenkasti statusni kod i kratko objašnjenje poslanog koda, npr. OK. U odgovoru, jednako kao kod zahtjeva, nakon prve linije dolazi lista zaglavlja koja su poslana te nakon zaglavlja prazni redak i tijelo odgovora. U slučaju ispod tijelo je popunjeno i klijent prima html dokument koji može prikazati u web pregledniku.

```
HTTP/1.1 200 OK
Content-Type: text/html
Content-Lenght: 40

<html><h1>Hello World!</h1></html>
```

HTTP protokol je protokol bez stanja, što znači da je životni vijek veze između klijenta i poslužitelja ograničen na jednu razmjenu zahtjeva i odgovora. Poslužitelj ne održava stanje veze tijekom prijenosa uzastopnih naredbi. Ne može pamtiti niz interakcija niti grupirati zahtjeve zajedno. Budući da protokol ne održava informacije o sesiji, web aplikacije moraju koristiti druge mehanizme, poput kolačića za uspostavljanje trajnih sesija ili stanja kroz višestruke zahtjeve [10].

Websocket je protokol koji omogućuje dvosmjernu komunikaciju između klijenta i poslužitelja preko jedne, dugotrajne TCP veze. Za razliku od klasičnog http-a koji koristi model zahtjev odgovor, websocket omogućuje da obje strane šalju poruke u bilo kojem trenutku, bez potrebe za novim zahtjevima [12]. Ova karakteristika je idealna za aplikacije koje zahtijevaju ažuriranja u stvarnom vremenu i možemo nazvati websocket kao protokol koji održava stanje, za razliku od http-a koji je bez stanja. Uspostava websocket veze počinje kao običan http zahtjev koji je nadograđen na websocket protokol. Klijent šalje posebni http GET zahtjev s zaglavljem `Upgrade: websocket`. Poslužitelj odgovara statusom `101 Switching Protocols` ako prihvaća zahtjev. Nakon ovoga, veza prelazi na websocket protokol. Poruke se šalju u obliku okvira i postoji nekoliko vrsta okvira, to su tekst okviri, za slanje tekstualnih poruka, binarni okviri, za slanje biranih poruka i kontrolni okviri, za slanje keep-alive poruka i zatvaranje veze [13]. U kasnijim poglavljima su prikazani primjeri gdje i kako se oba protokola koriste.

4.2.2. Axum

Axum je okvir za izradu web aplikacija u ekosustavu programskog jezika rust, dizajniran za izgradnju asinkronih http usluga s naglaskom na ergonomiju, modularnost i kompozabilnost.

Pruža visokorazinsku apstrakciju za usmjeravanje, obradu zahtjeva i generiranje odgovora, dok koristi nižerazinske komponente iz asinkronog rusta. Naglašava tipksi sigurno usmjeravanje bez korištenja makro komandi i mehanizme ekstrakcije, omogućujući deklarativno parsiranje zahtjeva putem ekstraktora, na primjer pruža ekstraktore za parametre puta ili parsiranje JSON tijela zahtjeva [14].

Axum se u aplikaciji koristi kao web poslužitelj, odgovoran je za povezivanje klijenata, spajanje s bazom podataka, pohranjivanje podataka o igrama te upravljanje i serviranje slika koje klijenti pohrane za neku igru. Kroz nekoliko primjera koda ćemo proči glavne značajke axum-a i način na koji se koristi kroz aplikaciju.

```rust
pub fn get_router<T: AppStateTrait>(state: T) -> axum::Router {
    axum::Router::new()
        .route("/assets/upload", routing::post(upload::upload))
        .route(
            "/assets/thumbnails/{image_id}",
            routing::get(serve::thumbnails),
        )
        .route("/assets/{filename}", routing::get(serve::serve_file))
        .layer(DefaultBodyLimit::disable())
        .layer(RequestBodyLimitLayer::new(FILE_SIZE_LIMIT))
        .with_state(state.clone())
}
```

Isječak koda 1: Kreiranje axum ruta za učitavanje i serviranje slika

U isječku koda 1 je prikazana funkcija koja kreira axum `Router` objekt s globalnim stanjem aplikacije `state`. U funkciji su definirane tri različite rute, detalji o rutama su u poglavlju 4.2.5. Funkcija `route` kao argumente prima putanju na koju će web poslužitelj reagirati ako je zatražena od klijenta, drugi argument registrira http metodu i funkciju koja se poziva za generiranje odgovora na klijentov zahtjev. Na primjer `route("/assets/upload", routing::post(upload::upload))` registrira rutu `"/assets/upload"`, definira da prima samo zahtjev za `POST` metodu, te ako klijent napravi ispravan zahtjev na ovu rutu poziva funkciju `upload::upload`. Funkcija `with_state` prima dijeljeno globalno stanje aplikacije i omogućava funkcijama koje generiraju odgovor klijentu pristup tome stanju, na primjer stanje može sadržavati vezu na bazu podataka. U slijedećem isječku je prikazana struktura globalnog dijeljenog stanja.

```rust
pub struct AppStateConfig<F>
where
    F: Adapter,
{
    pub file_system_handler: F,
    pub database: DatabaseConnection,
    pub entity_queue: Arc<Mutex<EntityQueue>>,
}

impl AppStateConfig<local_adapter::Local> {
    pub async fn get_default_config() -> AppStateConfig<local_adapter::Local> {
        let database = Self::get_database().await;
        let entity_queue = Arc::new(Mutex::new(EntityQueue::new(database.clone())));
        Self {
            file_system_handler: Self::get_fs_handler_from_config(),
            database,
            entity_queue,
        }
    }
}

impl<F> AppStateTrait for AppState<F> where F: Adapter
{
    type FsHandler = F;

    fn get_fs_handler(&self) -> Self::FsHandler { self.fs_handler.clone() }
    fn get_db(&self) -> DatabaseConnection { self.database.clone() }
    fn get_entity_queue(&self) -> Arc<Mutex<EntityQueue>> {
    ↪  self.entity_queue.clone() }
    fn get_scheduler(&self) -> Scheduler { self.scheduler.clone() }
}
```

Isječak koda 2: Implementacija dijeljenog globalnog stanja

Zbog jednostavnosti aplikacija koristi jedno globalno stanje, iako axum podržava različito stanje za svaki `Router` objekt. Globalno stanje se sastoji od objekta za upravljanje datotekama te u ovom slučaju stanje koristi implementaciju za pristup datotekama na lokalnom računalu tj. pristup datoteka na računalu koje pokreće poslužitelj. Stanje se također sastoji od veze s bazom reda čekanja entiteta `EntityQueue`. Red čekanja entiteta će biti detaljnije objašnjen u kasnijem poglavlju, no za sad je samo bitno znati da postoji u globalnom stanju i bude dostupan kroz ekstraktore u funkcijama za generiranje odgovora na zahtjeve klijenta.

```rust
pub async fn upload<F: Adapter>(
    asset_manager: AssetManager<F>,
    conn: DbConn,
    multipart: extract::Multipart,
) -> Result<Json<UploadResponse>> {
    let file = UploadedFile::from_multipart(multipart).await?;

    let transaction = conn.begin().await?;

    let asset = asset_manager
        .create(
            &transaction,
            file.name.to_string(),
            &file.data,
            AssetType::File,
        )
        .await?;

    let thumbnails = if infer::is_image(&file.data) {
        asset_manager
            .create_thumbnail_assets(&transaction, &asset, Some(&file.data))
            .await?
    } else { vec![] };

    transaction.commit().await?;

    Ok(Json(UploadResponse {
        id: asset.id,
        thumbnails: thumbnails
            .into_iter()
            .map(|t| gen_partial_asset_url(&t.name))
            .collect(),
        url: gen_partial_asset_url(&asset.name),
        filename: asset.name,
        original_filename: asset.original_filename,
    }))
}
```

Isječak koda 3: Implementacija upload funkcije za učitavanje datoteka na poslužitelja

Isječak 3 prikazuje cijelu implementaciju funkcije za učitavanje datoteke s klijenta na poslužitelja. U isječku 2 definirana je struktura globalnog stanja koju aplikacija koristi, a u ovoj funkciji se koriste dva objekta izvedena iz tog stanja. Ti objekti su `AssetManager<F>` i `DbConn`, dok axum pruža implementaciju za `extract::Multipart`. Funkciju ni u jednom trenutku ne pozivamo ručno, već kroz registraciju rute i korištenjem rust kompajlera axum generira poziv na definiranu funkciju s točnim parametrima. To radi na način da svi parametri funkcije moraju implementirati trait `FromRequestParts`, što omogućava axum-u mogućnost izvlačenja, postavljanja i prilagođavanje parametara.

Nadalje u funkciji se axum-ov ekstraktor `extract::Multipart` koristi za dobivanje podataka o datoteci koja se šalje poslužitelju, nakon izvlačenja podataka pokreće se transakcija

na razini baze podataka. Transakcija osigurava da će se u bazu zapisati podaci o učitanom dokumentu samo ako sve prođe bez greške. Ako se u bilo kojem trenutku kod pisanja podataka u bazu ili kasnijeg generiranja sličica originalne datoteke dogodi greška od koje se aplikacija ne može oporaviti, transkacija će biti obustavljena i podaci neće biti zapisani u bazu podataka. Kod završetka funkcije generira se odgovor koji se šalje klijentu. Odgovor je u JSON formatu, format je vidljiv kroz tip podataka koji funkcija vraća `Json<UploadResponse>`.

```rust
pub struct DbConn(DatabaseConnection);
impl<S> FromRequestParts<S> for DbConn
where
    S: AppStateTrait,
{
    type Rejection = Error;

    async fn from_request_parts(
        _parts: &mut Parts,
        state: &S,
    ) -> core::result::Result<Self, Self::Rejection> {
        Ok(state.get_db().into())
    }
}
```

Isječak koda 4: Implementacija ekstraktora za vezu s bazom podataka

Isječak iznad prikazuje implementaciju traita `FromRequestParts` koji omogućava axum-u te na kraju programeru korištenje prilagođenih ekstraktora u funkcijama za generiranje odgovora na klijentove zahtjeve. U isječku je definirano da varijabla `state` mora implementirati `AppStateTrait` (implementacija vidljiva u isječku 2) što garantira varijabli `state` posjedovanje metode `get_db` i sposobnost kreiranja veze s bazom podataka. Tu vezu dalje axum može proslijediti u funkcije koje koriste ekstraktor `DbConn`.

4.2.3. Socketioxide i Socket.IO

Socketioxide je implementacija socket.io poslužitelja [15] u rust-u. Biblioteka je prigodna za aplikaciju jer ima direktnu integraciju s axum-om. Socket.IO je biblioteka koja omogućava koja omogućuje događanjima vođenu dvosmjernu komunikaciju u stvarnom vremenu između web klijenata, obično web preglednika i poslužitelja. Namijenjen je aplikacijama koje zahtijevaju trenutno slanje podataka između klijenta i poslužitelja, s minimalnim kašnjenjem. Za potrebe aplikacije socket.io je korišten za uspostavljanje websocket veze između klijenata i poslužitelja te za prosljeđivanje podataka s jednog klijenta na drugi, bez da ti klijenti moraju biti direktno povezani jedni s drugima.

Komunikacija kroz socket.io je bazirana na slanju događaja između klijenta i poslužitelja. Da bi se neki dio koda izvršio primitkom određenog događaja, klijent ili poslužitelj trebaju imati registriran slušatelj (engl. *listener*) za taj događaj. Na primjer, u isječku koda 5 postoji linija `socket.on(JOIN_EVENT, join_handler::<T>);` koja označava da će poslužitelj reagirati na događaj `JOIN_EVENT` koji dobije od klijenta.

Isječak koda ispod prikazuje način na koji se socketioxide integrira s axum-om, kako se kreira namespace i ruta na koji se klijenti mogu spajati, te dodatne postavke vezane za spajanje i održavanje veze s klijentom, također opet je vidljivo globalno stanje aplikacije `AppStateTrait` koje se prosljeđuje u socketioxide kontekst kako bi i socketioxide imao pristup globalnom stanju aplikacije. U poglavlju 4.2.6 je konkretno objašnjeno gdje se, u ovom kontekstu, globalno stanje koristi. Prije registracije nove rute potrebno je kreirati i konfigurirati socketioxide uslugu koja se može proslijediti u rutu. Usluga ima postavljeno na šezdeset sekundi vremena čekanja za `ping` i `ack` događanja. Ovo su posebna događanja koja socketioxide koristi za provjeru veze s klijentom i uspješno dostavljenih poruka/događaja. Poslužitelj u nekom intervalu šalje ping događaj klijentu te klijent ima šezdeset sekundi za poslati pong odgovor, u suprotnom veza s klijentom bude isključena. Na sličan način funkcionira ack događaj, kod ovog događaja klijent ima određeno vrijeme za slanje odgovora primitka nekog događaja od poslužitelja. Ovim mehanizmima socket.io garantira da su klijenti aktivni te ako nisu može isključiti vezu s njima i obustaviti slanje događaja prema neaktivnim klijentima.

`io.ns("/", on_connect::<T>);` definira da će se pozvati `on_connect` funkcija kad se klijent spoji na namespace `/`. Socket.io koristi namespace kao jedan od načina za grupiranje povezanih klijenata i odabir klijenata kojima će se slati događaji. Drugi način su sobe, poslužitelj odlučuje ako i u koju sobu će pridružiti klijenta. Sobe su koncept jedino dostupan poslužitelju, klijenti nemaju pristup popisu soba kojima su pridruženi. Spajanjem klijenta poziva se funkcija `on_connect` koja služi da se povezanom klijentu registrira povratni poziv funkcije u slučaju ako se klijent isključi te postavlja vezu u određeno stanje. Drugi registrirani povratni poziv funkcije se koristi za povezivanje klijenta u određenu sobu i slanje podataka o igri tom klijentu te povezivanje tog klijenta s drugima u istoj sobi. Sve dok klijent ne pošalje odgovarajući `JOIN_EVENT` događaj neće primati nikakve događaje od poslužitelja, ni poslužitelj neće primati nikakve druge događaje od klijenta. Više o povezivanju između klijenata i poslužitelja u poglavlju 4.2.6.

```rust
pub fn get_router<T: AppStateTrait>(state: T) -> axum::Router {
    let (service, io) = SocketIoBuilder::new()
        .with_state(state.clone())
        .ping_timeout(Duration::from_secs(60))
        .ack_timeout(Duration::from_secs(60))
        .connect_timeout(Duration::from_secs(60))
        .build_svc();

    io.ns("/", on_connect::<T>.with(auth_middleware::<T>));

    axum::Router::new().route_service(
        "/socket.io/",
        ServiceBuilder::new()
            .layer(CorsLayer::permissive())
            .service(service),
    )
}

pub fn on_connect<T: AppStateTrait>(socket: SocketRef) {
```

```
17      let socket_clone = socket.clone();
18      socket.on_disconnect(move |reason: DisconnectReason| {
19          socket_clone.extensions.remove::<JoinedFlag>();
20      });

21      socket.on(JOIN_EVENT, join_handler::<T>);
22  }
```

Isječak koda 5: Postavljanje socketioxide poslužitelja kao axum rutu

4.2.4. Baza podataka

Baze podataka koriste se u aplikacijama kako bi osigurale pouzdan, trajan i strukturiran način pohrane informacija. Omogućuju organiziranje podataka u tablice i relacije, što olakšava upravljanje složenim skupovima podataka. Osim pohrane, baze podataka ključne su za očuvanje integriteta podataka putem transakcija, koje osiguravaju da podaci ostanu dosljedni čak i u slučaju kvarova sustava. Također omogućavaju aplikacijama pretraživanje i dohvaćanje specifičnih informacija putem upitnih jezika poput SQL-a [16].

U aplikaciji se koristi SQLite za pohranu podataka. SQLite je odabran zbog jednostavnosti direktnog ugrađivanja baze podataka u aplikaciju. Za razliku od tradicionalnih baza podataka, poput PostgreSQL-a, koji koriste klijent-poslužitelj arhitekturu i zahtjevaju zaseban poslužiteljski proces za upravljanje podacima i zahtjevima. SQLite integrira cijeli mehanizam baze podataka izravno u aplikaciju. SQLite cijelu bazu podataka pohranjuje u jednu datoteku [16].

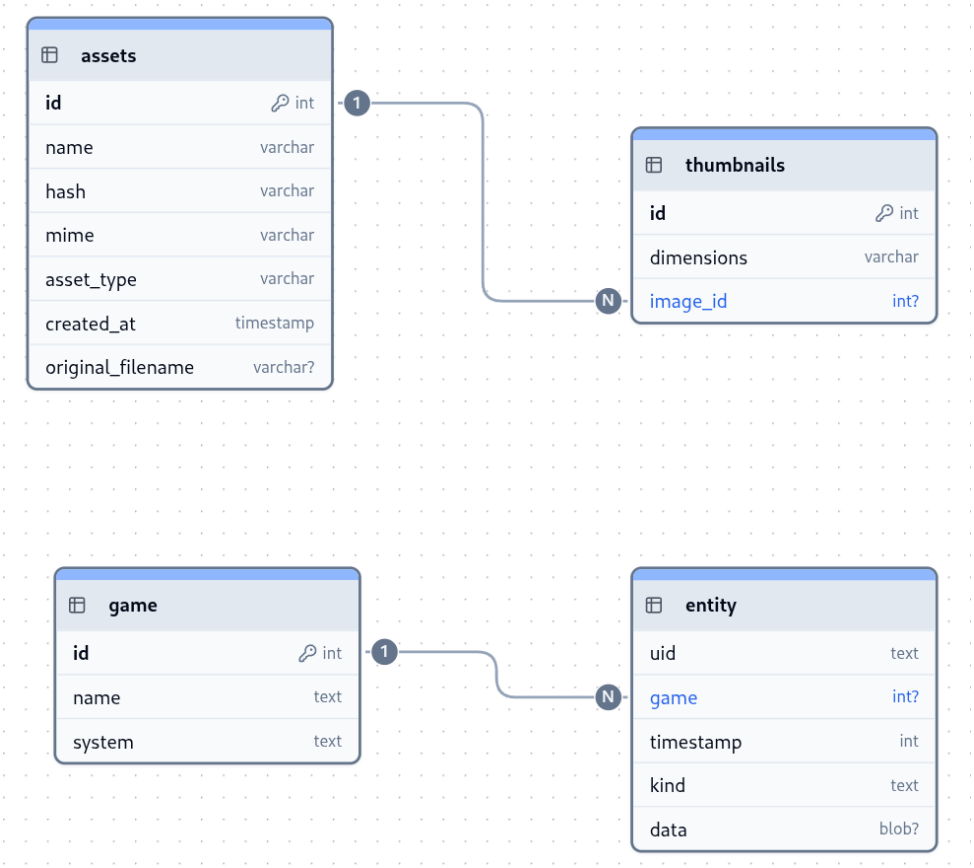
Bez posebne konfiguracije SQLite nije najbolje namijenjen za istovremeno pisanje i čitanje podataka. Zato su u slijedećem isječku koda prikazane korištene postavke za SQLite vezu, kako bi se poboljšale performanse kod istovremenog pisanja i čitanja za više klijenata. `journal_mode` je postavljena na `WAL` kako bi se omogućilo istovremeno čitanje i pisanje. Opcija `synchronous` je postavljena na `NORMAL` kako bi se izbjegao gubitak podataka u slučaju da se dogodi neka greška pri pisanju podataka iz write ahead log-a (WAL-a) u bazu. Opcija `busy_timeout` ima vrijednost od pet sekundi te omogućava bazi da pričeka određeno vrijeme prije vraćanja greške u slučaju da pokušava zapisivati u redove zaključane drugom transakcijom. Opcija `cache_size` povećava veličinu predmemorije i smanjuje broj zapisa koji će se raditi na disk. Opcija `foreign_keys` uključuje provjeru stranih ključeva na tablicama, bez ove opcije SQLite ne provjerava ako tablica ima relaciju na neku drugu tablicu, ni ako je ta relacija validna. Postavljanjem `auto_vacuum` na `INCREMENTAL` omogućuje postepeno oslobađanje prostora na disku kako se retci iz baze brišu, umjesto potpunog oslobađanja tijekom nekih od operacija nad bazom. Postavljanjem `temp_store` na `MEMORY` SQLite će privremene tablice pohraniti u memoriji umjesto spremati ih na disk i kasnije odbaciti [17].

```
1  let pool = db.get_sqlite_connection_pool().clone();

2  let connect_options = (*pool.connect_options())
```

```
    .clone()
    .journal_mode(sqlx::sqlite::SqliteJournalMode::Wal)
    .synchronous(sqlx::sqlite::SqliteSynchronous::Normal)
    .busy_timeout(std::time::Duration::from_secs(5))
    .pragma("cache_size", "-20000")
    .foreign_keys(true)
    .auto_vacuum(sqlx::sqlite::SqliteAutoVacuum::Incremental)
    .pragma("temp_store", "MEMORY");
pool.set_connect_options(connect_options);
```

Isječak koda 6: Postavljene SQLite opcije

Na slijedećoj slici je prikaz svih tablica i relacija u bazi podataka. Baza se sastoji od tablica `assets`, `thumbnails`, `game` i `entity`. Tablica `assets` definira datoteku učitanu na poslužitelja, ima nekoliko stupaca koji opisuju naziv, vrstu datoteke, originalni naziv datoteke. `Thumbnails` tablica ima relaciju na `assets` tablicu, jedan red u thumbnails tablici predstavlja sličicu koja je generirana za originalnu učitanu sliku. Tablica `game` samo zadrži naziv igre i sustav koji igra koristi, dok `entity` tablica sadrži sve entitete vezane za neku igru i sve njihove podatke. Podaci i vrsta entiteta definirana je od strane klijenta tj. implementacije igre, jednako tako klijent je odgovoran za spremanje id-a asset-a u neki od podataka u entitetu, poslužitelj nema način za pregled svih asset-a i provjeru kojoj igri pripadaju.



Slika 5: Shema baze podataka [autorski rad]

4.2.5. API krajnje točke

U tablici ispod su definirane sve krajnje točke kojima klijenti imaju pristup i na koje poslužitelj odgovara. Većina komunikacije između poslužitelja i klijenta odvija se kroz websocket. Može se primijetiti da u popisu fali krajnja točka za pridruživanje igri, fali iz razloga što se pridruživanje izvršava kroz websocket te ima nekoliko koraka koje klijent mora odraditi da bi se uspješno pridružio igri. Više o načinu pridruživanja u poglavlju 4.2.6.

Tablica 4: API krajnje točke [autorski rad]

Ruta	**HTTP Metoda**	**Opis**
/public/{path}	GET	Omogućava dohvaćanje datoteka koji su dostupni javnosti, na primjer ikonica ili css stilova
/socket.io/	GET	Služi za povezivanje socket.io klijenata
/assets/upload	POST	Omogućuje učitavanje datoteka na poslužitelj. Poslužitelj odgovara s idem datoteke linkovima i id-evima sličica ukoliko je učitana datoteka slika
/assets/thumbnails/{image_id}	GET	Dohvaća sličicu definiranu parametrom `image_id`
/assets/{filename}	GET	Dohvaća dokument definiran parametrom `filename`
/game/list	GET	U `JSON` formatu šalje popis svih dostupnih sesija igre
/game/create	POST	Kreira novu sesiju igre

4.2.6. Mrežna komunikacija

Mrežna komunikacija između klijenta i poslužitelja se odvija na dva načina. Prvi je kroz API krajnje točke, definirane u poglavlju 4.2.5, drugi je kroz websocket koristeći socket.io. U isječku koda 5 je vidljivo korištenje funkcije `.with(auth:middleware::<T>)`, da bi se klijent uspješno povezao na websocket mora poslati određeni sadržaj pri povezivanju na `/socket.io/` krajnju točku. Sadržaj koji klijent mora poslati je `id` igre u koju se želi pridružiti. Poslužitelj će tada obaviti autentifikaciju i korištenjem `socket.extension.insert(auth);` zapisati podatke na instancu websocketa kako bi se podaci mogli kasnije koristiti te da bi poslužitelj imao uvid u koju igru je klijent pridružen.

```
1 fn auth_middleware<T: AppStateTrait>(
2     socket: SocketRef,
3     State(app_state): State<T>,
4     Data(auth): Data<WebsocketAuthMessage>,
5 ) -> Result<(), websocket_auth::Error> {
```

```
    match auth.authenticate(&app_state) {
        Ok(()) => {
            socket.extensions.insert(auth);
            Ok(())
        }
        Err(e) => Err(e),
    }
}
```

Isječak koda 7: Autentifikacije websocket klijenta

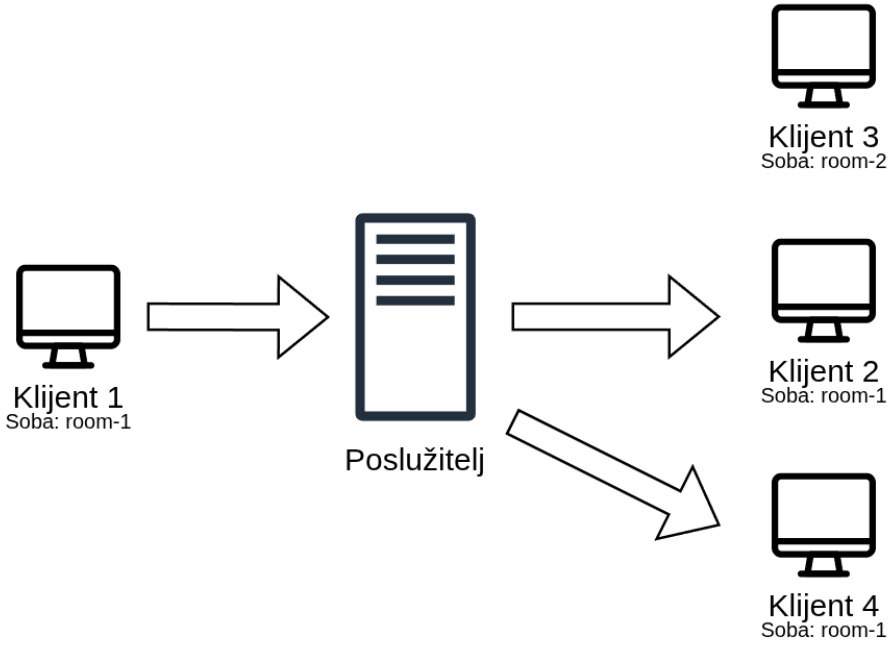
U poglavlju 4.2.3 je definirano da će poslužitelj, nakon uspješnog spajanja klijenta, samo reagirati na događaj odspajanja klijenta s veze i događaj `JOIN_EVENT`. Kad je klijent spreman može poslati `JOIN_EVENT` događaj na poslužitelj. Poslužitelj će u trenutku primitka ovog događaja početi proceduru za pridruživanje klijenta u igru. U isječku koda 8 je vidljiva cijela procedura pridruživanja klijenta u igru, započinje s čitanjem id-a igre kojoj se klijent pridružuje. Podatak je isčitan s websocket instance koja je u trenutnu autentifikacije imala postavljene podatke o igri. Nakon toga poslužitelj pridružuje websocket u sobu. Sobe se koriste za slanje podataka između klijenata koji su povezani u istu igru. Također odmah nakon pridruživanja sobe, poslužitelj registrira slušatelja za događaj `ACTION` na websocket. U isječku je vidljivo ponovno korištenje `AppStateTrait` globalnog stanja za dohvaćanje reda čekanja entiteta i pokretanje transakcije u bazi podataka. Red čekanja je korišten za periodično spremanje entiteta u bazu podataka te se kod svakog pridruživanja dohvaća i sprema u bazu kako bi kod dohvaćanja entiteta svi bili zapisani u bazi podataka. Nakon dohvaćanja svih entiteta iz baze, poslužitelj u manjim dijelovima šalje događaje `JOIN_EVENT` klijentu s podacima o entitetima i stanju slanja. Klijent je odgovoran za pretvaranje entiteta u objekte koje razumije i može koristiti tokom igre. Po završetku slanja entiteta poslužitelj šalje klijentu događaj `JOIN_FINISHED_EVENT` što označava da su svi entiteti uspješno poslani te klijent ima sve podatke za pridruživanje i sinkronizaciju igre.

```
#[tracing::instrument(skip(socket, app_state))]
async fn join_handler<T: AppStateTrait>(
    socket: SocketRef,
    State(app_state): State<T>,
    Extension(auth): Extension<WebsocketAuthMessage>,
) {
    let game_id = auth.game;
    let room = format!("room-{game_id}");

    socket.join(room);
    socket.on(ACTION, action_handler::<T>);

    let queue = app_state.get_entity_queue();
    {
        let mut lock = queue.lock().await;
        if let Some(handle) = lock.flush().await { handle.await.ok(); }
    }
```

```
16      tracing::debug!("Starting database entity fetch");
17      let db = app_state.get_db();
18      let transaction = db.begin().await?;

19      let entity_manager = EntityManager::new();

20      let entities = entity_manager.load_entities(&transaction, game_id).await?;

21      let total = entities.len();
22      let mut sent = 0;
23      tracing::debug!("Sending entities in {} chunks", total / CHUNK_SIZE);
24      entities.chunks(CHUNK_SIZE).for_each(|chunk| {
25          sent += chunk.len();
26          socket
27              .emit(JOIN_EVENT, &serde_json::json!({
28                      "progress": {
29                          "sent": sent,
30                          "total": total
31                      },
32                      "data": chunk
33                  }),
34              )
35              .ok();
36      });

37      socket.emit(JOIN_FINISHED_EVENT, &()).ok();
38  }
```

Isječak koda 8: Procedura za pridruživanje

Pridruženi klijenti komuniciraju kroz poslužitelja slanjem događaja `ACTION`. Svi validni `ACTION` događaji će biti proslijeđeni do klijenata. Na slici ispod je prikazan primjer razmjene poruka između klijenata i poslužitelja. Na poslužitelja su spojena četri klijenta, klijent 1, 2 i 4 su spojeni u sobu `room-1` dok je klijent 3 spojen u sobu `room-2`. Klijent 1 pošalje validni događaj prema poslužitelju, poslužitelj će taj događaj procesirati te proslijediti događaj ostalim klijentima. Poslužitelj odabire primatelje prema sobi kojoj pripada pošiljatelj događaja, pošto u ovom primjeru pošiljatelj (Klijent 1) pripada u sobu `room-1`, događaj je proslijeđen klijentima 2 i 4, dok klijent 3 nema uvid da se uopće dogodila bilo kakva razmjena informacija i ne dobiva nikakav događaj.



Slika 6: Komunikacija između klijenata [autorski rad]

Isječak koda ispod prikazuje strukturu poruke koja je očekivana. Klijent poruku mora poslati u JSON formatu te poruka mora imati definiranu strukturu. Mora sadržavati polja `action` i `data`. Polje `action` definira vrstu akcije, postoji nekoliko definiranih vrsta `Update`, `Create`, `Delete`, `Transitive` i `Other`.

Vrste `Update`, `Create`, `Delete` osim što klijentima označavaju da se neki entitet promijenio, poslužitelju daju do znanja da je za entitete u `data` polju potrebno izvršiti određenu radnju, bilo to kreirati ih i spremiti u bazu podataka, ažurirati postojeće vrijednosti ili izbrisati entitete. Zadnje dvije vrste `Transitive` i `Other` su vrste poruka koje su uvijek poslane samo do drugih klijenata, a poslužitelj ne radi dodatno procesiranje ovih poruka. `Transitive` je namijenjen porukama koje nemaju potrebe za poslužiteljom i ne ažuriraju podatke vezane uz igru, na primjer jedan klijent može poslati poruku da se drugom klijentu na ekranu otvori novi prozor. `Other` je vrlo slična vrsta `Transitive` s razlikom da klijent na ovu vrstu poruke ima mogućnost pridodati proizvoljne podatke te ti podaci ne podliježu nekoj određenoj strukturi, definirani su samo kod klijenta.

Funkcija polja `data` je slanje podataka o entitetima nad kojima se vrši neka radnja. Cijela struktura jednog entiteta definirana je od strane klijenta, no poslužitelj treba nekoliko podataka o entitetu kako bi ga mogao pohranjivati te sinkronizirati podatke između klijenata. Entiteti su definirani kroz nekoliko polja `uid`, `kind`, `timestamp` i `other_values`. Polje `other_values` služi kao rezervirano mjesto za bilo kakve dodatne podatke vezane uz entitet. Polje `uid` definira jedinstveni identifikator entiteta. Odgovornost je klijenta da sve identifikatore zadrži jedinstvenima. `Kind` služi kako bi klijent mogao razlikovati između različitih vrsta entiteta koje kreira. U poglavlju 4.3 je objašnjen način pretvaranja entiteta iz JSON formata u entitet objekt koji igra može koristiti. Zadnje definirano polje je `timestamp`, koje služi za sinkronizaciju podataka. Mrežna komunikacija je bazirana na mreži ravnopravnih čvorova te svaki klijent ima

odgovornost postaviti `timestamp` na vrijeme u milisekundama kad je zadnje napravljena neka promjena na entitetu. Poslužitelj koristi `timestamp` za identifikaciju promjena na entitetima i odlučivanja koje promjene su ažurne, a koje može odbaciti.

```rust
#[derive(Debug, Clone, Serialize, Deserialize, PartialEq, Eq)]
#[serde(rename_all = "kebab-case")]
pub enum Action {
    Update,
    Create,
    Delete,
    Transitive,
    Other(serde_json::Value),
}

#[derive(Debug, Clone, Serialize, Deserialize)]
pub struct ClientsideEntity {
    pub uid: UId,
    pub kind: EntityKind,
    pub timestamp: UtcTimestamp,
    #[serde(flatten)]
    pub other_values: serde_json::Value,
}

#[derive(Debug, Clone, Serialize, Deserialize)]
pub struct ActionMessage {
    action: Action,
    data: Vec<ClientsideEntity>,
}
```

Isječak koda 9: Struktura Action poruke

4.3. Klijent

Klijent je drugi glavni dio cijele aplikacije. Pruža korisnicima korisničko sučelje za praćenje bitke na više različitih scena te uvid u trenutna događanja u igri. Implementiran je korištenjem web tehnologija i pokreće se u web pregledniku kroz web stranicu ili kroz tauri okvir kao zasebna desktop aplikacija.

4.3.1. Pixi.js

Pixi.js je biblioteka za 2D renderiranje izgrađena na WebGL i WebGPU tehnologijama. Omogućava razvijanje grafičkih aplikacija na web-u te je pogodna za izradu igara, interaktivnih oglasa, edukacijskih materijala i vizualizaciju podataka [18]. U aplikaciji se pixi.js koristi za prikaz različitih scena s poljima, pozadinskim grafikama, likovima, vizualizaciju poteza te bilo kakvih drugih grafičkih elemenata.

Arhitektura pixi.js je podijeljena u nekoliko glavnih komponenata [19]:

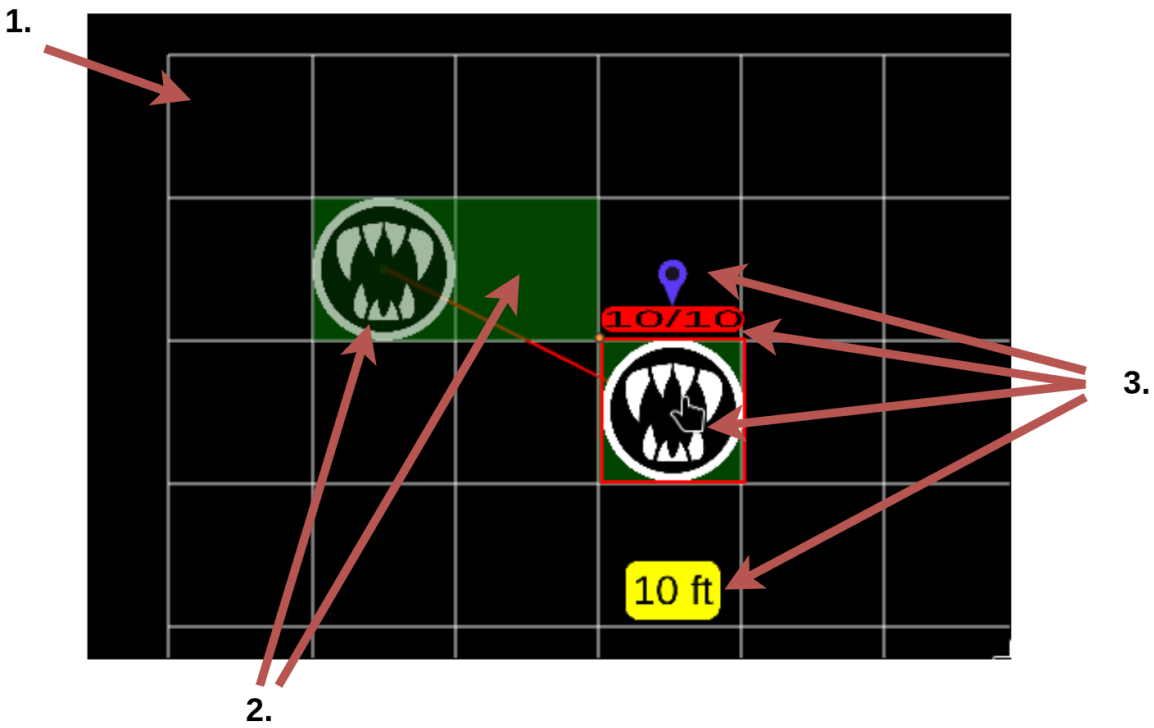
- Komponenta za renderiranje: prikazuje i crta scenski graf na ekran.
- Kontejneri: Glavni objekti na sceni kroz koje se kreira scenski graf tj. stablo kontejner objekata koji se mogu prikazati, objekti kao što su tekst, sprite-ovi i druga grafika.
- Slikovno/zvučni elementi: sustav za asinkrono učitavanje i dekodiranje resursa poput slikovnih i zvučnih datoteka.
- Ticker: periodični povratni pozivi temeljeni na vremenskom intervalu. Korisni su za ažuriranje podataka, grafike ovisno o vremenu ili nekoj promjeni podataka kroz vrijeme.
- Događanja: interaktivnost kontejner objekata kroz emitiranje i reagiranje na događaje, npr. klik na objekt ili prolaz pokazivača miša preko objekta.

Kombinacijom glavnih komponenata moguće je implementirati sve dijelove potrebne za aplikaciju. Na primjer, u aplikaciji se kontejneri koriste za prikaz scena i crtanje polja na ekran, slikovno/zvučni elementi za učitavanja slika i postavljanje slika u kontejnere, ticker za periodično crtanje obruba oko kontejnera te postepenu promjenu boje obruba, događanja su korištena i omogućavaju korisniku premještanje pozicije kontejnera i promjenu veličine određenih kontejnera.

Sva grafika u pixi.js-u osnovana je na kontejnerima koji sadrže podređene kontejnere, slike, tekst ili grafiku. Pixi.js uvijek ima barem jedan kontejner tzv. korijenski kontejner u koji se dodaju drugi kontejneri. Kontejneri su povezani u stablo, gdje svaki čvor stabla može imati povezane druge kontejnere i grafika je prikazana na ekran prema redoslijedu kontejnera u stablu. Za svaki kontejner koji sadrži druge kontejnere renderira se u dubinu tj. pixi.js će prolaziti kroz podređene čvorove sve dok ne dođe do čvora koji je list stabla. List stabla je čvor koji nema podređeni čvor.

U primjeru ispod prikazana je struktura stabla kontejnera za crtanje i prikaz tokena u trenutku kad ga korisnik klikom miša premješta na drugu poziciju. U korijenski kontejner dodana je pozadina s poljima, označena s 1., pošto je prva u redoslijedu crtanja biti će prikazana u pozadini. Ovaj kontejner sadrži dva podređena kontejnera 2. prijašnja lokacija tokena i 3. token kontejner. Slijedeći kontejner koji će pixi.js renderirati je 2. prijašnja lokacija tokena, ovaj kontejner sadrži sličicu tokena na prijašnjoj lokaciji te polja preko kojih je token prenešen, označena zelenom bojom. Na slici 7 vidljivo je da je sličica tokena iza polja označenih polja, baš iz razloga što je sličica renderirana prije označenih polja. Na isti način je renderiran 3. token kontejner. Taj kontejner sadrži više različitih elemenata, kao što su sličica, obrub, prikaz životnih bodova (koji može biti dalje rastavljen na dijelove), prikaz pomaknute udaljenosti te indikator trenutnog tokena na potezu. Sličica je iza obruba, prikaz životnih bodova, ne prekriva sličicu, ali bi bio prikazan iznad ako bi prekrivao, jednako tako vrijedi za prikaz udaljenosti i indikator.

```
Pixi.js Korijenski kontejner
|
|-- 1. Pozadina s poljima
     |
     |-- 2. Prijašnja lokacija tokena
     |    |
     |    |-- Sličica tokena
     |    |-- Polja preko kojih je token prenesen
     |
     |-- 3. Token kontejner
          |
          |-- Sličica
          |-- Obrub oko sličice
          |-- Prikaz životnih bodova
          |    |
          |    |-- Obojana pozadina
          |    |-- Tekst s vrijednostima
          |
          |-- Prikaz pomaknute udaljenosti
          |-- Indikator trenutnog tokena na potezu
```



Slika 7: Primjer redoslijeda renderiranja kontejnera [autorski rad]

4.3.2. React

React je javascript biblioteka namijenjena za izgradnju korisničkih sučelja u web aplikacijama [20]. Najčešće je korištena za aplikacije s jednom stranicom. Ovakve aplikacije uglavnom koriste različite implementacije usmjerivača kako bi jedna web stranica imala više različitih stranica. Arhitektura react-a je temeljena na komponentama, one su neovisni, ponovno upotrebljivi građevni blokovi korisničkog sučelja. Komponente upravljaju vlastitim internim podacima te promjenom podataka u komponenti react će ponovno prikazati komponentu s novim podacima, na ovakav način react omogućuje ažurno korisničko sučelje [20].

Pošto je react samo biblioteka za izradu korisničkih sučelja, ne grupira i ne dolazi s unaprijed kreiranim komponentama. Kako bi izbjegli potrebu za kreiranjem svake komponente, na primjer jednostavne tipke ili prozora za odabir više elemenata, aplikacija koristi biblioteku mantine. Mantine je biblioteka koja implementira veliki broj react komponenata i pruža unaprijed stilizirane i funkcionalne komponente koje se mogu dodatno prilagođavati.

Još jedna biblioteka koja se koristi u aplikaciji je jotai. Jotai je biblioteka za atomski pristup i upravljanje globalnim varijablama react stanja. Globalno stanje se izrađuje kroz kombinaciju atoma i promjene te potrebni prikazi promjena su optimizirani na temelju ovisnosti o atomu. Ovakav pristup rješava problem ponovnog i nepotrebnog renderiranje react konteksta, smanjuje ili eliminira potrebu za memoizacijom [21]. U aplikaciji je jotai uglavnom korišten za usklađivanje promjena između korisničkog sučelja i pixi.js konteksta, tj. objekata koje pixi.js renderira.

4.3.3. Socket.io

Glavnina socket.io funkcionalnosti je već spomenuta u poglavlju 4.2.3 no ovdje dodatno prolazimo kroz funkciju koju socket.io i websocketi imaju na strani klijenta. Glavnu funkciju koju websocket ima na strani klijenta je pretvaranje podataka o entitetima iz JSON formata u javascript objekte. Ta funkcionalnost je prikazana u isječku koda ispod.

Klasa `Websocket` sadrži funkcije `initJoin` i `initWebsocketListeneres` korištene za pokretanje procesa spajanja klijenta s poslužiteljem, prihvaćanja i slanja `action` događaja. Funkcija `initJoin` definira dvije povratne (engl. `callback`) funkcije koje se koriste za reagiranje klijenta na događaje `join-finished` i `join` poslane od strane poslužitelja. Pozivanjem funkcije `initJoin` započinje proces povezivanja s poslužiteljem i dohvaćanja svih podataka o igri. Klijent uzastopno prihvaća JSON podatke kroz događaj `join` te ih sprema u red čekanja. Nakon primitka događaja `join-finished` klijent pokušava pretvoriti sve primljene podatke u entitete. Više o entitetima u poglavlju 4.3.4. Događaj `join-finished` također zaustavlja klijenta od reagiranja na daljnje `join` događaje iz razloga što je klijent u tom trenutku u potpunosti pridružen igri i ima sve potrebne podatke za rekonstrukciju stanja igre.

Druga funkcija `initWebsocketListeners` je korištena za kontinuiranu sinkronizaciju podataka tokom igre. Klijent reagira na svaki `action` događaj i ako je događaj validne strukture ažurira, kreira ili briše entitete. Kad događaj definira akciju vrste `update` klijent će

pokušati ažurirati postojeće entitete s najnovijim podacima. Entiteti mogu definirati stanje kad se primljeni podaci smiju zamijeniti s novim primljenim podacima, no u većini slučajeva to je samo provjera ako je trenutni timestamp entiteta manji od timestamp vrijednosti dobivene iz događaja. Akcije `create` i `delete` dodaju entitete u red čekanja i odmah isprazne određeni red čekanja tj. `create` kreira sve novo primljene entitete dok, `delete` obriše sve entitete s klijenta.

```
export class Websocket {
    public initJoin() {
        const join = (joinData: JoinData) => {
        ↪  GBoard.entityRegistry.queue(joinData.data); };

        const joinFinished = () => {
            this.socket.off("join", join);
            GBoard.entityRegistry.convertQueuedEntities();
        };
        this.socket.once("join-finished", joinFinished);
        this.socket.on("join", join);
        this.socket.emit("join");
    }

    public initWebsocketListeners() {
        this.socket.on("action", (message) => {
            switch (message.action) {
                case "update": {
                    message.data.forEach((data) => {
                        const entity = GBoard.entityRegistry.entities.get(data.uid);
                        if (entity && entity.shouldApplyChanges(data)) {
                            entity.applyUpdateAction(data);
                        }
                    });
                    break;
                }
                case "create": {
                    GBoard.entityRegistry.queue(message.data);
                    GBoard.entityRegistry.convertQueuedEntities();
                    break;
                }
                case "delete": {
                    GBoard.entityRegistry.queue(message.data);
                    GBoard.entityRegistry.removeQueuedEntities();
                    break;
                }
            }
        });
    }
}
```

Isječak koda 10: Pridruživanje igri, primanje i slanje `action` događaja

4.3.4. Entiteti

Entiteti su različite vrste objekata kojima klijent upravlja, sadrže podatke o igri i sinkroniziraju se između svih povezanih klijenata (već definirano u poglavlju 4.2.6). Entiteti moraju sadržavati polja `uid`, `kind`, `timestamp` da bi se mogli slati drugim klijentima. Pošto je klijent odgovoran za generiranje unikatnih vrijednosti `uid` koristi se dinamično izgenerirani UUID kao vrijednost tog polja za svaki novokreirani entitet. Klijent podatke o entitetima prima u JSON formatu preko povezanog websocket-a, kako bi mogao pretvoriti dobivene podatke iz `join` ili `action` događaja u entitet. Da bi se ovo izvršilo potrebno je registrirati vrstu entiteta u globalnoj instanci klase `EntityRegistry`. Ova klasa je odgovorna za definiranje vrsta entiteta, način i redoslijed kojim se pretvaraju JSON podaci u entitete te za dohvaćanje i spremanje svih korištenih entiteta.

U isječku koda ispod definirano je nekoliko vrsta entiteta koje će sustav poznavati i imati mogućnost pretvoriti iz JSON formata u entitete. Kod registracije entiteta potrebno je definirati unikatni naziv entiteta i funkciju koja će biti odgovorna za pretvaranje JSON podataka u entitet. Redoslijed registracije je također bitan, pošto neki entiteti mogu ovisiti o drugim entitetima. Na primjer `RpgToken` entitet ovisi o `RpgTokenData` entitetu. Klijent može primiti podatke od poslužitelja u bilo kojem redoslijedu te se određivanjem redoslijeda definira koji podaci se mogu pretvoriti u entitet, dok drugi podaci moraju ostati u redu čekanja sve dok se ne kreira odgovarajući entitet o kojem ovise. Nije nužno da sve vrste entiteta budu registrirane kroz `EntityRegistry` klasu, no svi entiteti koji nisu registrirani nemaju mogućnost biti sinkronizirani preko svih klijenata.

```
1 const registerEntities = (registry: EntityRegistry) => {
2     registry.register(RpgScene.getKindStatic(), RpgSceneConverter.convert);
3     registry.register(RpgTokenData.getKindStatic(), RpgTokenDataConverter.convert);
4     registry.register(RpgToken.getKindStatic(), RpgTokenConverter.convert);
5     registry.register(TurnOrder.getKindStatic(), TurnOrderConverter.convert);
6     registry.register(DecorationTokenData.getKindStatic(),
      ↪  DecorationTokenDataConverter.convert);
7     registry.register(DecorationToken.getKindStatic(),
      ↪  DecorationTokenConverter.convert);
8 };
```

Isječak koda 11: Registracija različitih vrsta entiteta za implementaciju D&D sustava

Funkcije za pretvaranje entiteta iz JSON formata su korištene interno u `EntityRegistry` klasi. Ove funkcije se pozivaju kad klijent primi podatke o entitetima od poslužitelja. Na primjer, u isječku koda 10 na liniji 26, funkcija `convertQueuedEntities`, pretvara sve dobivene podatke u entitete prema definiranom redoslijedu i korištenjem registriranih funkcija za svaku vrstu entiteta.

Isječak koda ispod prikazuje jednu od klasa koje implementiraju konverziju iz JSON podataka u entitet. Funkcija prima JSON objekt s poljima koja su definirana kroz tip podataka

`TokenAttributes`. Prvo iz globalne `EntityRegistry` klase pokušava dohvatiti već postojeći `RpgToken` entitet te ako ga uspješno pronađe i podaci o entitetu se mogu mijenjati, na postojeći entitet se postavljaju najnoviji dobiveni podaci. Ako entitet ne postoji i nije dodan u `EntityRegistry`-u tada se pokušavaju dohvatiti entiteti `Scene` i `RpgTokenData`. O ovim entitetima ovisi entitet `RpgToken` i bez njih ga nije moguće kreirati. Nakon što sustav pronađe ovisne entitete kreira novi `RpgToken` entitet i dodjeljuje mu podatke dobivene od poslužitelja. Na ovakav način svaki entitet se može sinkronizirati s drugim klijentima i moguće je rekonstruirati cjelo stanje igre kod inicijalnog spajanja klijenta.

```
1  export class RpgTokenConverter {
2      public static convert(attributes: TypedJson<TokenAttributes>): RpgToken {
3          const existingEntity = GBoard.entityRegistry.entities.get(attributes.uid);
4          if (existingEntity instanceof RpgToken &&
           ↪  existingEntity.shouldApplyChanges(attributes)) {
5              existingEntity.applyUpdateAction(attributes);
6              return existingEntity;
7          }
8          const scene = GBoard.entityRegistry.entities.get(attributes.sceneUid);
9          const tokenData = GBoard.entityRegistry.entities.get(attributes.tokenData);
10         if (scene instanceof Scene && tokenData instanceof RpgTokenData) {
11             const token = new RpgToken(scene, tokenData);
12             token.applyUpdateAction(attributes);
13             scene.addToken(token, token.layer);
14             return token;
15         }
16         throw new Error("RpgToken conversion failed");
17     }
18 }
```

Isječak koda 12: Konverzija JSON podataka u RpgToken entitet

4.3.5. Vrste entiteta

Aplikacija implementira različite vrste entiteta i entiteti su podijeljeni u općenite entitete koji su namijenjeni kao osnovne klase koje se mogu proširiti i kreirati više specijalizirani entiteti. Osnovni entiteti su:

- **Grid**: predstavlja pixi.js kontejner korišten za prikaz polja. Sastoji podatke o poljima, njihovu veličinu, ukupne dimenzije i poziciju.
- **GameAssets**: entitet koji sprema sve slike i sličice učitane na poslužitelja od strane

klijenata i povezanih za instancu igre. Ne sadrži podatke o samim slikama već poveznice kako bi klijenti mogli učitati sliku sa poslužitelja u trenutku kad im zatreba.

- **Scene**: entitet označava scenu, scena je bilo kakva skupina pixi.js kontejnera koji će biti crtani i prikazani na ekranu.
- **SpriteExtension**: pixi.js kontejner koji je moguće dodati na scenu. Razlikuje se od običnog pixi.js kontejnera u tome što može sadržavati sliku i prikazivati je na sceni, korisnik ima mogućnost premještanja i postavljanja ovog entiteta po sceni, također podržava promjenu veličine.
- **EmptyTokenData**: entitet namijenjen za proširenje, ne sadrži nikakve podatke, osim potrebnih za slanje na poslužitelja.
- **Token**: spaja entitete `SpriteExtension` i `EmptyTokenData`, namijenjen proširenju i korišten za prikazivanje entiteta na sceni, te povezivanja vanjskih podataka na taj entitet.

Definirani osnovni entiteti su dovoljni za proširenje i implementaciju sustava za praćenje bitki te se koriste u djelomičnoj implementaciji D&D sustava, slijedeći su definirani prošireni entiteti:

- **RpgScene**: proširenje scene, sadrži dodatne funkcionalnosti za scene, na primjer, neke scene mogu biti skrivene od igrača, samo su prikazane DM-u. Omogućava dodatne kontrole za `Token` entitete (likovi) na sceni, provjeru i prikaz udaljenosti za koju se neki lik može pomaknuti, provjeru ako je na određenom polju već drugi lik.
- **RpgTokenData**: proširenje `EmptyTokenData` entiteta, sadrži sve podatke o D&D liku, na primjer trenutne životne bodove, ime lika, veličinu, trenutnu brzinu i ostale podatke koji opisuju lika.
- **RpgToken**: proširenje `Token` entiteta. Interno koristi `RpgTokenData` entitet za dohvaćanje podataka i mora biti povezan na neku scenu. Korišten za prikaz lika na sceni i kontroliranje lika. Ovakav pristup odvajanja podataka i entiteta koji se prikazuje na sceni omogućuje promjenu podataka na jednom mjestu, u `RpgTokenData` i svi podaci/promjene će biti vidljive i sinkronizirane na različitim scenama. Ne dozvoljava ručnu promjenu i postavljanje veličine tokena, već je veličina određena prema veličini definiranoj u `RpgTokenData` i poštuje veličinu tokena temeljenu na D&D pravilima.
- **TurnOrder**: entitet korišten za praćenje poteza i odgovoran za cijeli tok bitke. Sprema podatke o redoslijedu poteza, koji lik je trenutno na potezu i koji su uključeni u borbu, koliko akcija i brzine ima lik na potezu.
- **DecorationTokenData**: sadrži poveznicu na sličicu koja će biti prikazana na sceni.
- **DecorationToken**: entitet namjenjen za dodavanje dekoracija na scenu povezan s `DecorationTokenData`.

5. Prikaz slučajeva korištenja

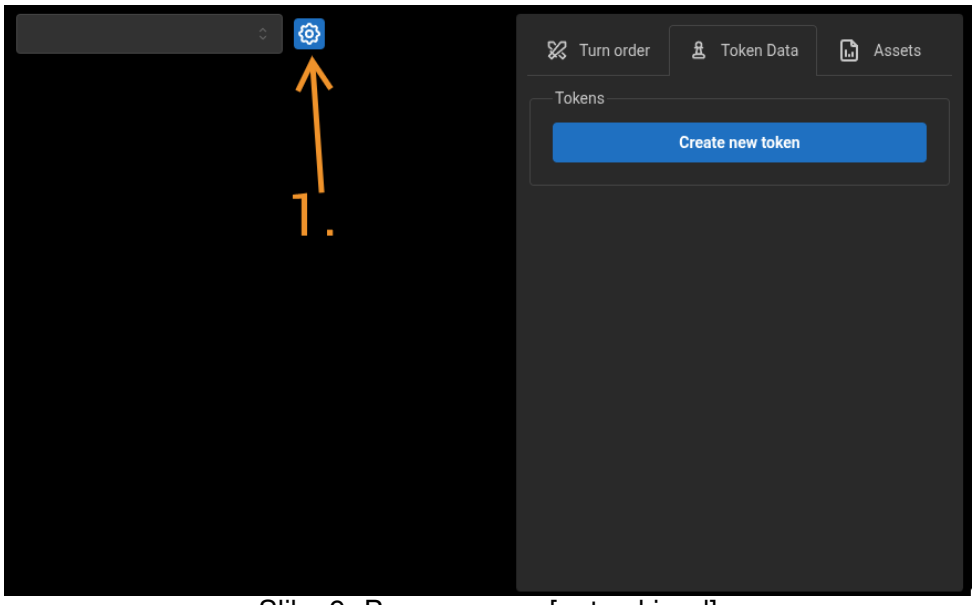
Aplikacija je namjenjena za istovremeno korištenje od strane više korisnika. Korisnici se mogu podijeliti u dvije grupe, to su DM i ostali igrači. Slika ispod prikazuje slučajeve korištenja aplikacije te pojedine razlike u funkcionalnostima koje različite vrste korisnika imaju priliku koristiti. Svi korisnici, DM i igrači imaju mogućnost pregleda scena i upravljanje tokenima. Tokeni mogu predstavljati likove igrača, NPC-ove, čudovišta ili kakve druge objekte, na primjer sličice dodane kao dekoracije ili sliku lokacije u pozadini. Upravljanje tokenima podrazumijeva kreiranje novih tokena, postavljanje i uređivanje podataka o tokenu, pozicioniranje tokena na sceni te, tokom bitke, korištenje akcija dodijeljenih tokenu.

Najveća razlika između DM-a i igrača je što DM može sakrivati neke dijelove igre od samih igrača. Na primjer, DM može pripremiti scenu za borbu u špilji, no igrači ne moraju unaprijed znati da će tokom sesije istraživati špilju te u tom slučaju DM može sakriti scenu i tek je prikazati igračima dok bude pravo vrijeme. Igrači imaju pristup pregledu svim nesakrivenim scenama, na njih mogu dodavati tokene, pozicionirati ih na bilo koji dio scene te uređivati neke dijelove scene.

Slika 8: Dijagram slučajeva korištenja aplikacije [autorski rad]

5.1. Sustav scena i korisničko sučelje

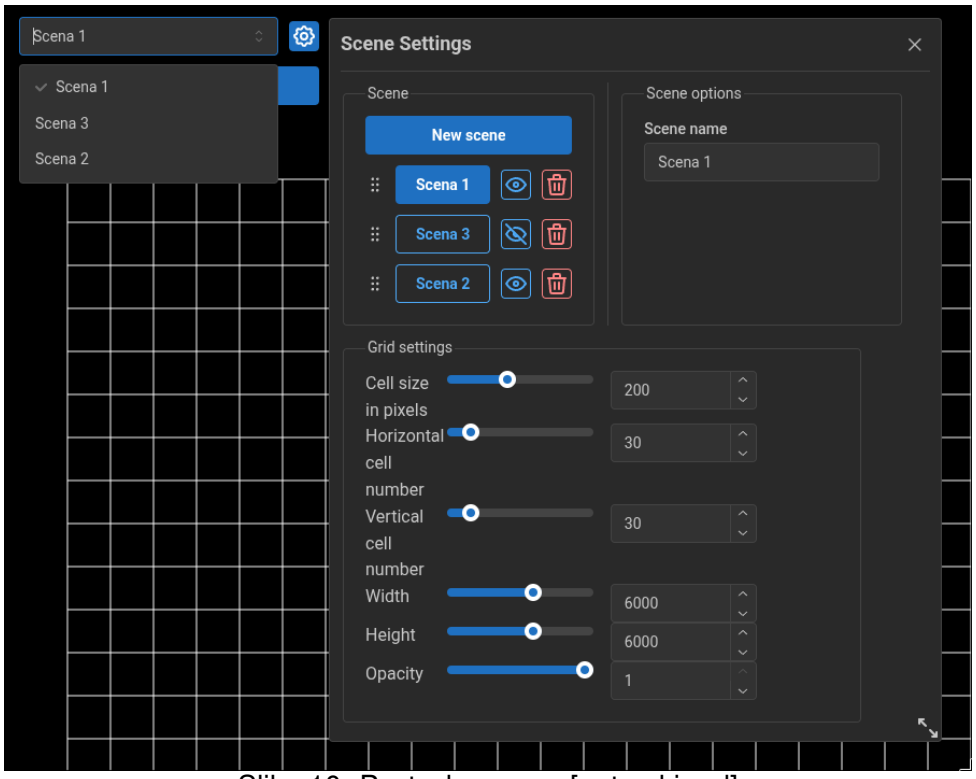
Scene su temelj aplikacije, svaka scena je unikatna, može sadržavati različite tokene te biti postavljena neovisno o drugim scenama. Kreiranjem nove igre, DM-u je prikazana prazna scena, tj. ekran bez ikakve scene, označeno s 1. je tipka za odabir scena, kreiranje novih scena i prilagođivanje scene. S desne strane se nalazi meni za upravljanje podacima o tokenima, upravljane bitkom i mogućnost za učitavanje slika, no više o ovim funkcionalnostima u kasnijem poglavlju.



Slika 9: Prazna scena [autorski rad]

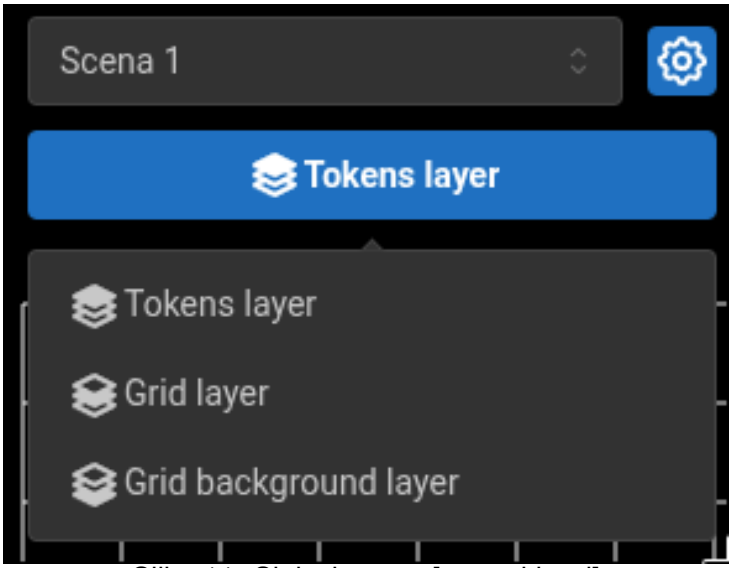
Klikom na tipku, otvara se prozor za postavljanje scena. U prozoru je moguće dodavanje više scena, određivanje njihovog poretka, sakrivanje scena od igrača, postavljanje naziva i postavljanje veličine polja i količine polja prikazanih na sceni. Postavljanje poretka scena će promjeniti poredak u kojem se scene prikazuju u izborniku za odabir scena. Scena koja je postavljena kao prva scena uvijek će biti prikazana korisnicima pridruživanja u igru.

Svaka scena se sastoji od rešetka ćelija/polja i moguće je postaviti broj ćelija u horizontalnom, vertikalnom smjeru te definirati veličinu ćelija u pikselima. Ove postavke mogu pomoći kod postavljanja pozadinske slike scene i svi tokeni (koji nisu dekoracijski) će biti postavljeni unutar ćelija te se mogu kretati samo po ćelijama.



Slika 10: Postavke scena [autorski rad]

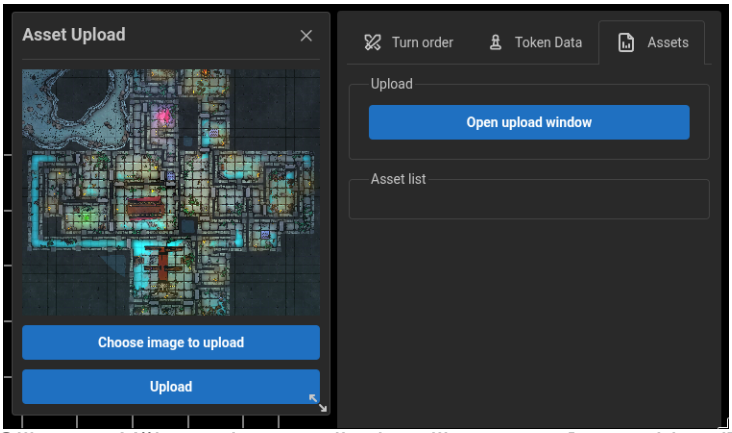
Osim rešetaka ćelija, svaka scena sadrži nekoliko slojeva. Ti slojevi su `Token layer`, `Grid layer`, `Grid background layer`. Samo `Token layer` je dostupan igračima, dok su ostali slojevi dostupni samo DM-u. Odabirom sloja omogućava se uređivanje tokena na tom sloju. Ovo omogućava izolaciju između tokena na različitim slojevima, tokene koji se nalaze na drugim slojevima nije moguće uređivati. Uređivanje tokena uključuje dodavanje tokena u sloj, postavljanje njihove pozicije, promjenu veličine i brisanje tokena.



Slika 11: Slojevi scene [autorski rad]

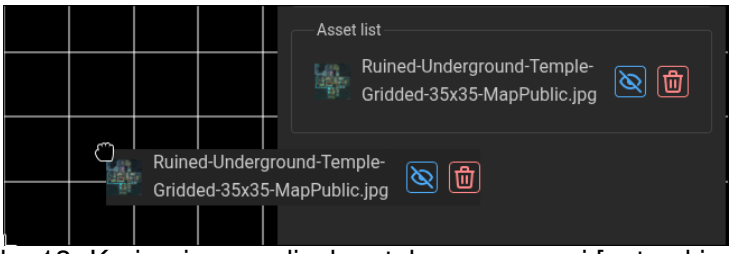
Na slici 9, s desne strane je prikazan meni s više kartica. Za učitavanje slika na poslužitelj koristi se `Assets` kartica. Odabirom kartice prikazuje se tipka za učitavanje slike, klikom

se otvara novi prozor za odabir slike, pregled i učitavanje slike. Nakon što korisnik učita sliku na poslužitelj, slika će biti vidljiva u `Assets` kartici pod `Assets list` kategorijom - vidljivo na slici 13.



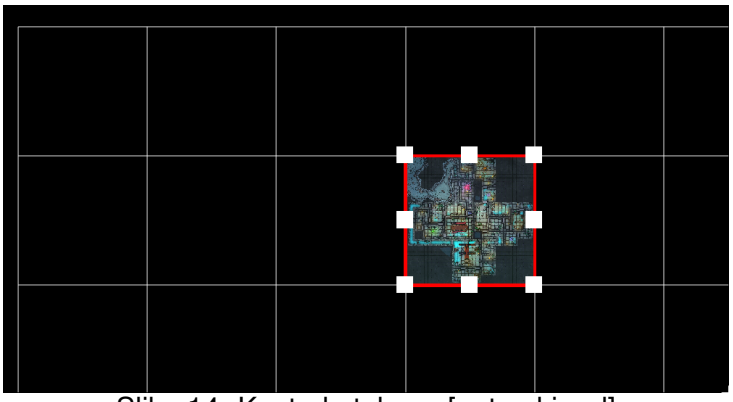
Slika 12: Učitavanje pozadinske slike scene [autorski rad]

Postavljanjem sloja na `Grid background layer`, korisnik može povući i spustiti sliku na bilo koji dio scene. To će automatski dodati dekoracijski token na taj sloj scene, no token će biti veličine jednog polja. Potrebno ga je namjestiti na točnu poziciju i proširiti kako bi se mogao koristiti kao pozadina za scenu.

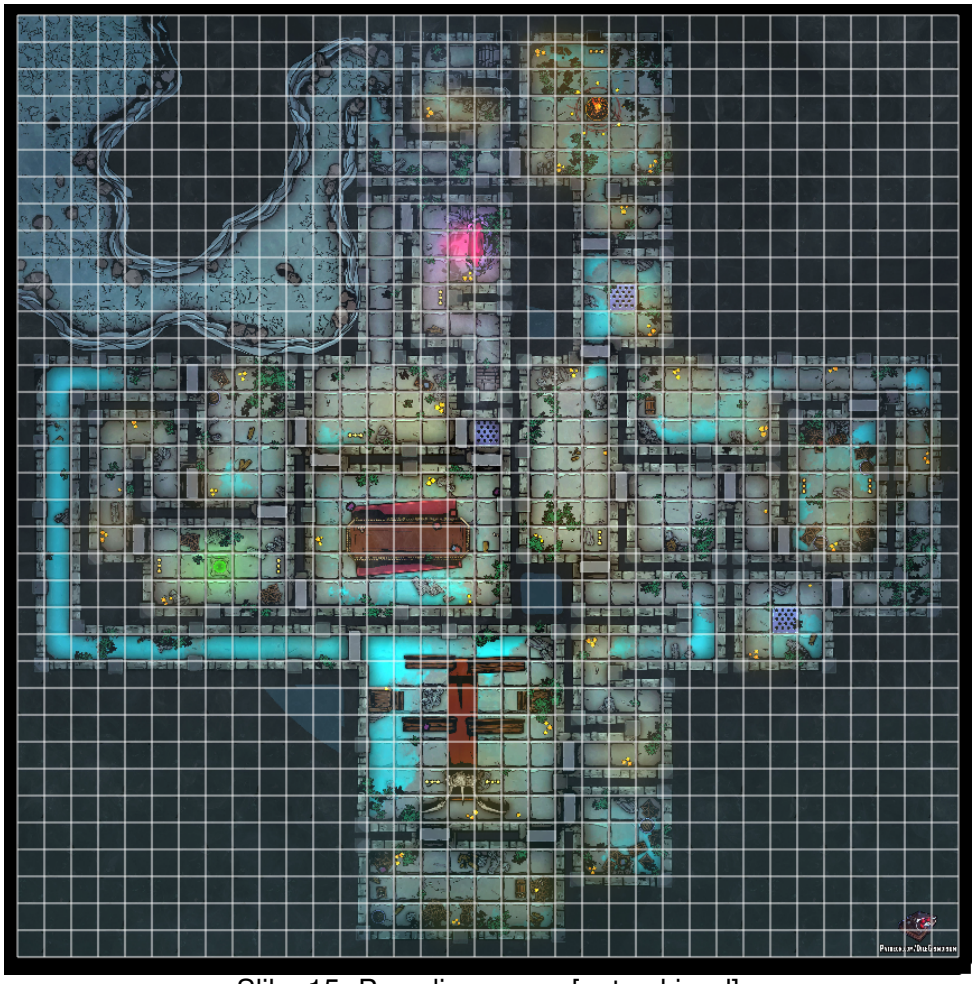


Slika 13: Kreiranje pozadinskog tokena na sceni [autorski rad]

Kao što je vidljivo na slici 14, token je trenutno premali da bi bio korišten za pozadinu, još se može uočiti da ima crveni obrub i bijele kvadratiće na svakom vrhu te polovici svake stranice kvadratića. Crveni obrub signalizira korisniku da je taj token trenutno odabran. Odabrani token može se premještati klikom i prevući na drugu poziciju, dok bijeli kvadratići označavaju točke kojima je moguće proširiti dimenzije označenog tokena. Postavljanjem pozicije i proširenjem tokena, pozadina scene je uspješno postavljena - vidljivo na slici 15. Sva polja pozadine jednaka su poljima na pozadinskoj slici.

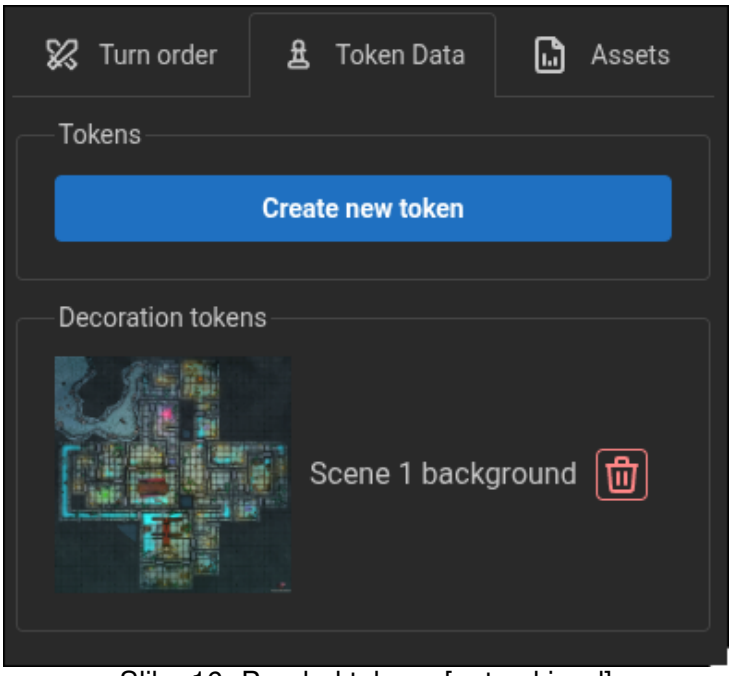


Slika 14: Kontrole tokena [autorski rad]



Slika 15: Pozadina scene [autorski rad]

Klikom na karticu `Token Data` otvara se prikaz svih kreiranih tokena. Tokeni su odvojeni u dvije grupe, `Tokens` i `Decoration tokens`. Trenutno je samo kreiran jedan dekoracijski token, korišten za pozadinu scene dok ne postoji ni jedan token za igru. Igrači imaju pregled tokena koje su sami kreirali te tokene koje im je dodijelio DM. Time je osiguran prikaz samo relevantnih tokena za igrače dok DM ima pregled svih tokena. Iz ove kartice moguće je dodavati tokene na scenu, na isti način kako je već bilo prikazano na slici 13.

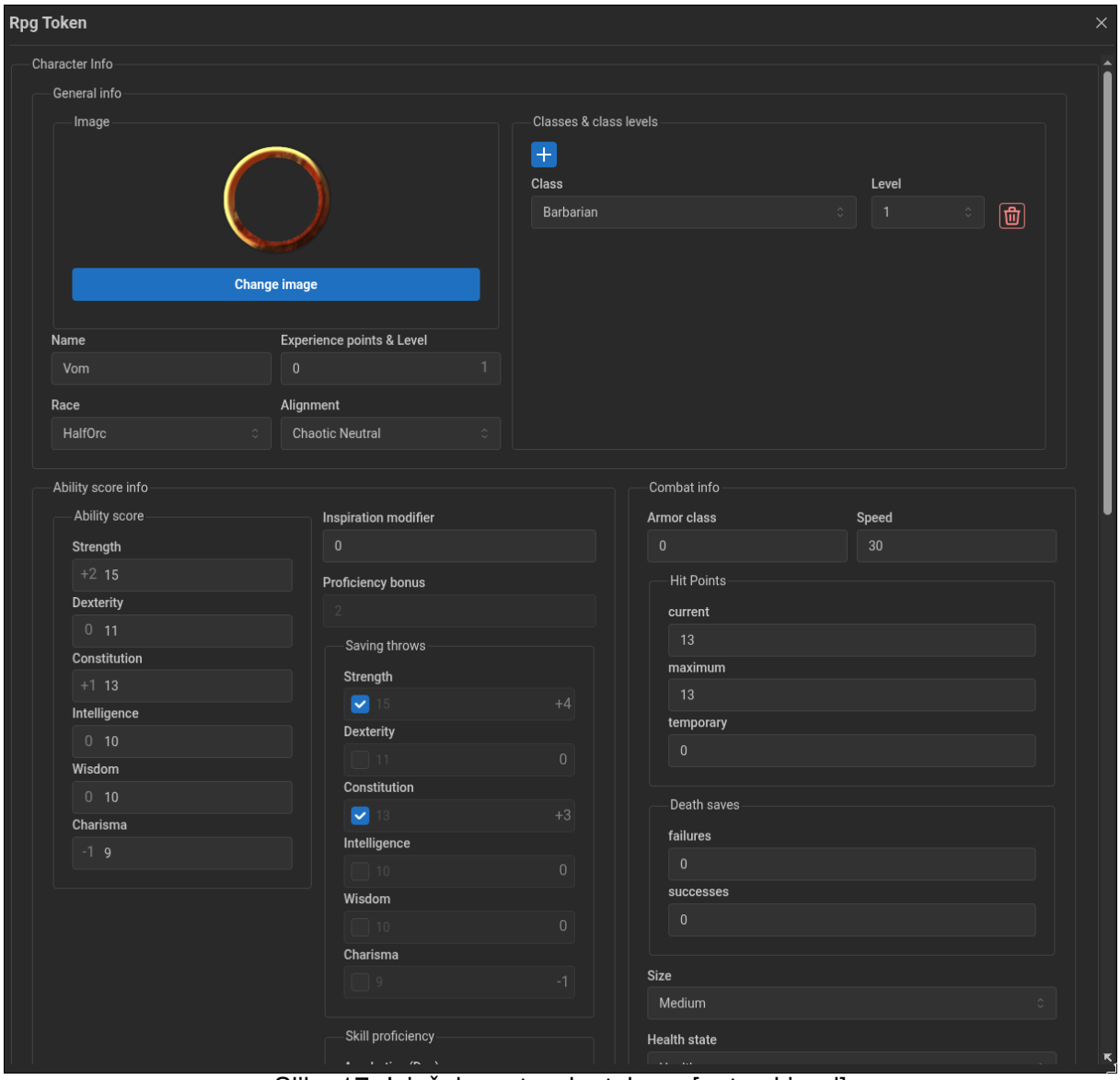


Slika 16: Pregled tokena [autorski rad]

Novi tokeni se kreiraju pritiskom na tipku `Create new token`, vidljivo na slici 16. Pritiskom tipke otvara se prozor za postavljanje tokena. Postavke imitiraju character sheet. Na slici 17 rekreiran je lik iz poglavlja 3.5. Razina tokena je automatski izračunata prema broju bodova iskustva. Upisane vrijednosti za sposobnosti automatski izračunavaju odgovarajuće modifikatore, na primjer, lik ima 15 snage što odgovara modifikatoru vrijednosti +2. Stručnosti u vještinama i bacanjima za spas također imaju automatski dodan modifikator odgovarajuće sposobnosti te dodatni modifikator vrijednosti za stručnost.

U ovom prozoru se također može postaviti veličina lika, koja će odgovarati veličini lika na sceni. Veći likovi će zauzimati više prostora na sceni, kao što je definirano tablicom kategorija veličina (tablica 3). Također moguće je postaviti opremu koju token koristi. Ovime se određuje koji napadi će biti dostupni tokenu tokom bitke. Važan dio postavka je `tags` grupa. Postavljanjem tagova DM može označiti koji token će biti dio:

- **družine igrača**: `party` tag,
- **neprijatelji**: `hostile-npc` tag,
- **prijateljski NPC-ovi**: `npc` tag,
- **posebni neprijatelji**: `special-npc-hostile`. Ova vrsta neprijatelja će padom na nula životnih bodova imati šansu za bacanje spasa od smrti, objašnjenog u poglavlju 3.6.5, jednako kao članovi družine, dok tokeni označeni s `hostile-npc` tagom umiru odmah nakon pada na nula životnih bodova.

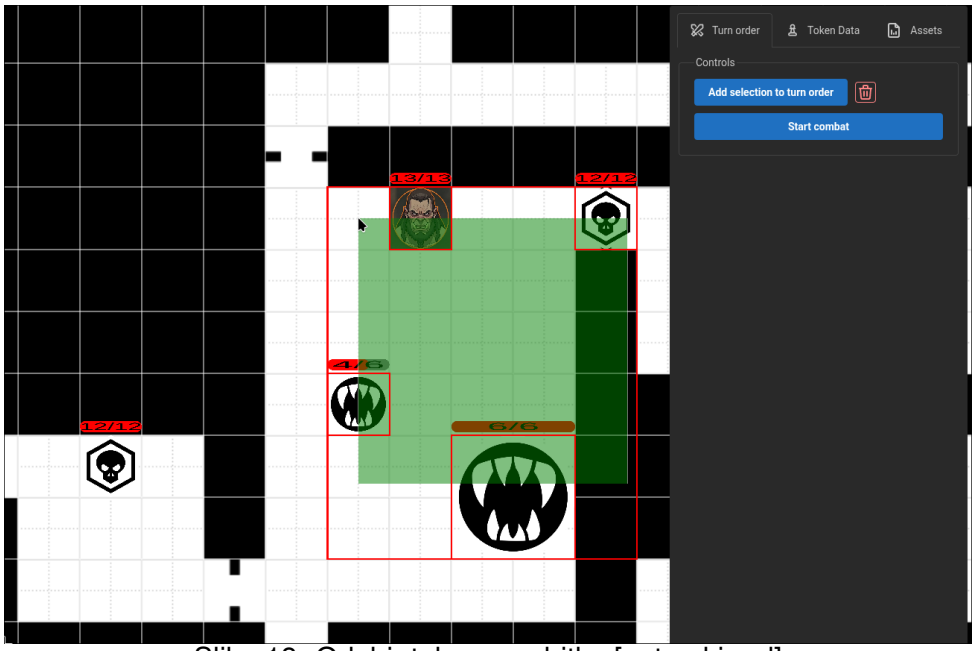


Slika 17: Isječak postavaka tokena [autorski rad]

5.2. Primjer bitke

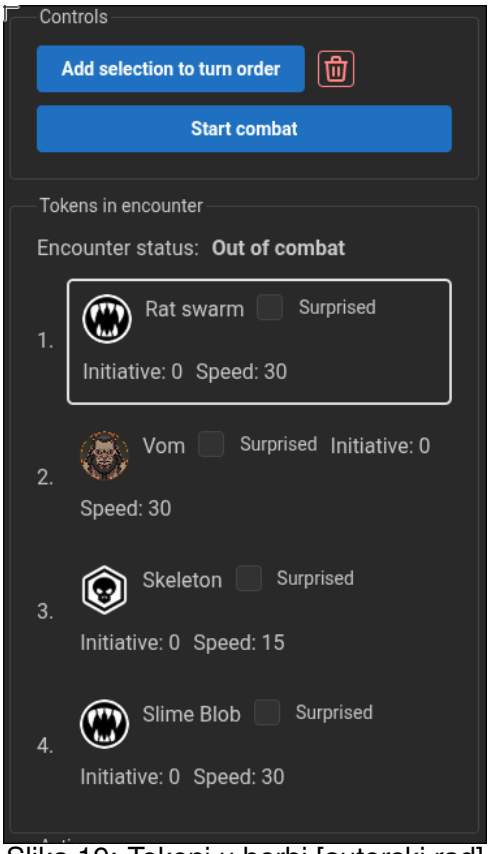
U aplikaciji bitke se odvijaju između odabranih tokena na nekoj sceni. Svaka scena može imati samo jednu aktivnu bitku, no u bitci može sudjelovati bilo koji broj tokena. Da bi tokeni bili dodani u bitku potrebno ih je označiti. Tokeni se mogu označiti na dva načina, klikom na određeni token ili crtanjem površine u kojoj će svi tokeni biti označeni. Token je označen kad ima crveni obrub oko sličice, ako se radi o skupini označenih tokena, svaki token će imati svoj obrub i cijela skupina će imati dodatni crveni obrub.

Na slici 18 je prikazano označavanje više tokena. Označeni tokeni mogu biti dodani u bitku klikom na gumb `Add selection to turn order` unutar `Turn order` kartice. Proces je jednak za dodavanje jednog ili više tokena u bitku.



Slika 18: Odabir tokena za bitku [autorski rad]

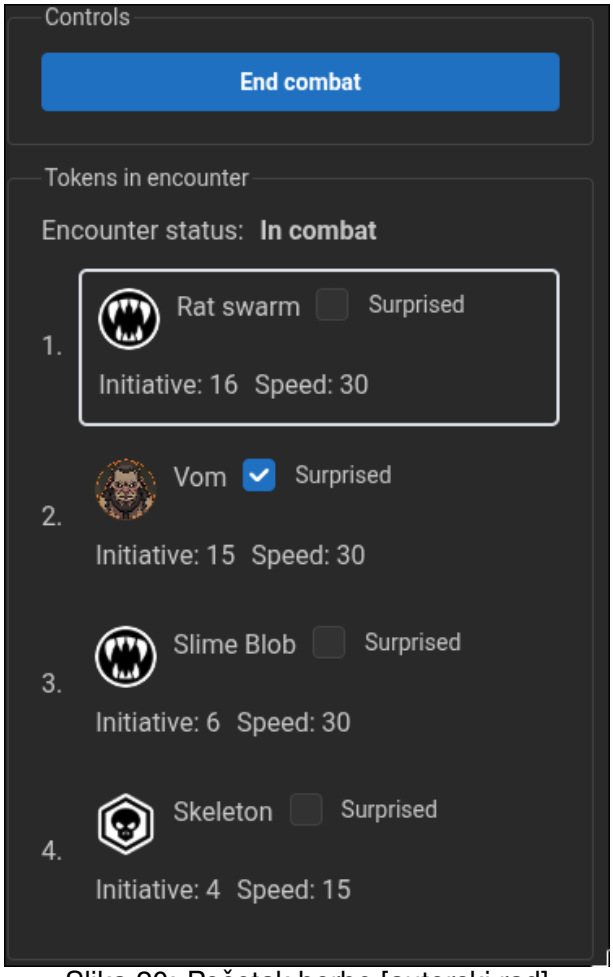
Nakon dodavanja tokena u borbu popunjava se grupa `Tokens in encounter`. Grupa prikazuje sve tokene koji su dodani u borbu, status borbe, ako je započela ili nije te brzinu i inicijativu svakog tokena. Prije početka borbe DM može postaviti koji tokeni će biti iznenađeni. Iznenađeni tokeni će automatski biti preskočeni kad prvi put dođu na red.



Slika 19: Tokeni u borbi [autorski rad]

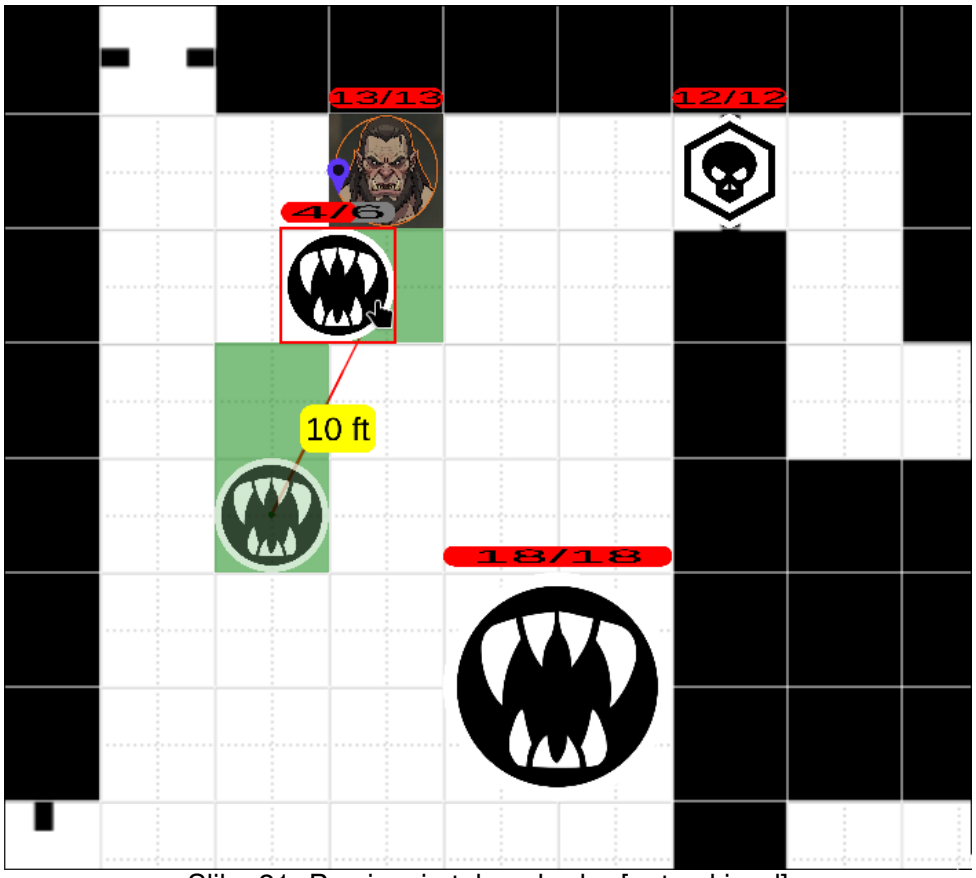
Bitka će započeti klikom na tipku `Start combat`. Pri pokretanju bitke svakom tokenu će se izračunati inicijativa i redoslijed poteza će biti postavljen i prikazan. Token koji je trenutno na potezu ima bijeli obrub u pregledu trenutnog poteza te je jedini token koji se može kretati po sceni. Pokušavanjem kretanja tokena koji trenutno nije na redu, korisnik dobiva poruku o greški i pozicija pomaknutog tokena nije promjenjena.

Prikaz za brzinu se dinamički mijenja kako token iskoristi briznu za trenutni potez. Nakon ponovnog dolaska na potez brzina će biti resetirana na inicijalnu vrijednost brzine koju taj token ima.



Slika 20: Početak borbe [autorski rad]

Tokom borbe tokeni imaju određenu količinu brzine koju mogu utrošiti za premještanje na drugu poziciju. Polja preko kojih će token preći su označena zelenom bojom. Token ne može završiti svoje kretanje na istom polju kao neki drugi token. U tom slučaju korisnik će dobiti grešku i token će biti resitiran na početnu poziciju, bez da izgubi ili iskoristi brzinu. Tokom odabira pozicije, ispod tokena je prikazan indikator udaljenosti koju će token preći. Jednak broj prikazan u indikatoru će na kraju pomaka biti oduzet od tokenove trenutne brzine.

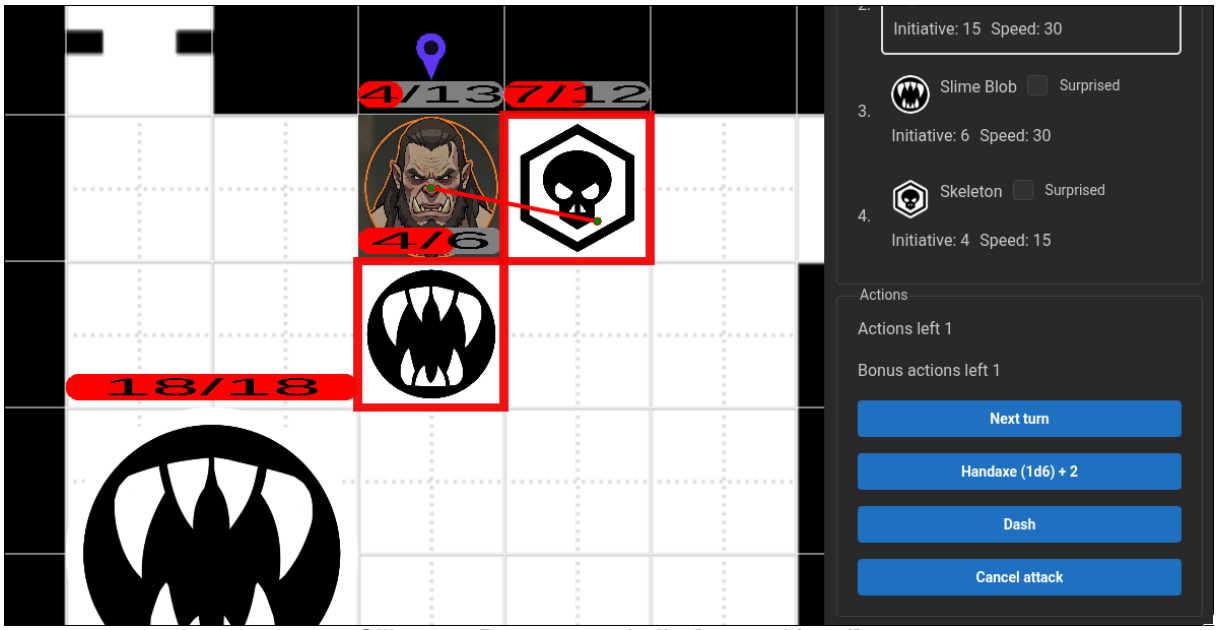


Slika 21: Pomicanje tokom borbe [autorski rad]

Token koji je na potezu može poduzeti akcije koje su pridodane tom tokenu. Na primjer, na slijedećoj slici igrač je na potezu (token na potezu ima indikator iznad sličice). Igrač može odabrati neku od akcija koje ima na izbor ili završiti potez. Na slici je igrač odabrao napasti neprijatelja koristeći napad `Handaxe`. Klikom na napad automatski su crvenom bojom obrubljeni tokeni koje može napasti. Napad ima domet od 5 stopa, što znači da može napasti samo tokene udaljene jedno polje. Ovo je vidljivo na slici 22. Tokeni pored su označeni dok token izvan dometa nema crveni obrub.

Da bi igrač odabrao metu napada mora kliknuti dva puta na metu. Prvim klikom biti će nacrtana crta između napadača i mete, dok će drugim klikom napad biti izvršen. Ovaj sustav duplog odabira je kreiran kako bi igrač mogao promjeniti metu u slučaju da se predomisli i želi napasti neku drugu metu. Druga prednost ovakvog pristupa je vezana uz napade koji imaju veći doseg. Aplikacija ne implementira zidove te će kao validne mete napada prikazati sve tokene u dosegu, to znači da je moguće napasti tokene iza zida. Zidovi su definirani pozadinom scene te je na igračima i DM-u da prate ako neki napad ne bi trebao biti moguć. Prikazom crte između napadača i mete je lakše identificirati ako je napad s većim dosegom uistinu izvodiv.

Osim odabira akcije, igrači imaju uvid o broju akcija i bonus akcija koje još mogu iskoristiti tokom poteza.

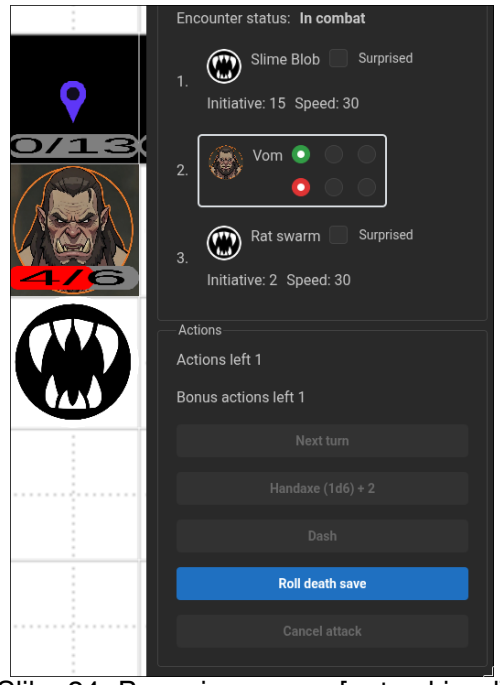


Slika 22: Dostupne akcije [autorski rad]

Nakon odabira mete igrač baca kockicu da vidi ako će napad uspješno pogoditi metu i ako će uspjeti nanijeti štetu meti. Klikom na tipku `Roll` kockica će biti bačena i nakon nekog vremena će prikazati dobiveni broj, ako je napad uspješan u prozorčiću će se prikazati nova kartica `Roll damage`. U novoj kartici prikazana je kockica ili kockice za određivanje štete. Bacanje tih kockica meta će primiti štetu baziranu na vrijednostima bacanja, dodatni modifikatori za štetu su automatski dodani. Na slici 22 je vidljivo da `Handaxe` napad radi 1d6 štete i ima modifikator +2 na dobivenu štetu.

Slika 23: Bacanje kockice za napad [autorski rad]

Kad token padne na nula životnih bodova, više neće biti prikazan kao sudionik borbe, sve dok borba traje, osim ako se radi o tokenima s tagom `party` ili `special-npc-hostile`. Oni ostaju u borbi jer moraju bacati kockice za spas od smrti. Kad dođu na potez imaju mogućnost odabira jedne akcije `Roll death save`. Klikom na ovu akciju otvara se prozor s kockicom, sličan prozoru na slici 23. Nakon bacanja kockice i dobivanja rezultata aplikacija će automatski zabilježiti uspješni pokušaj spašavanja ili neuspješni. Uspjeh ili neuspjeh zabilježeni su u podacima tokena i prikazani u potezu kao popunjeni zeleni ili crveni kružići.



Slika 24: Bacanje za spas [autorski rad]

6. Zaključak

Razvoj platforme za praćenje bitke u igrama igranja uloga pokazao je kako se kompleksna, analogna pravila sustava Dungeons & Dragons mogu uspješno digitalizirati i automatizirati korištenjem suvremenih web tehnologija. Kroz ovaj rad povezana je povijest strateških igara i ratnih simulacija s modernim potrebama zajednice igrača, čime je stvoren alat koji pomaže u administraciji vođenja igre.

Gledajući aspekte koji su uspješno realizirani, najveća prednost platforme je sinkronizacija podataka u stvarnom vremenu koja osigurava svim sudionicima pregled identičnog stanja igre, što je ključno za online igranje. Razdjela platforme na poslužitelj i klijent infrastrukturu omogućila je fleksibilnost kod implementacije D&D sustava na strani klijenta, pošto poslužitelj zahtijeva minimalno strukturirane podatke te je na sličan način moguće implementirati druge RPG sustave.

Postoje određeni segmenti koji su mogli biti bolje izvedeni. Trenutna implementacija sustava pravila oslanja se na fiksne parametre D&D sustava, što platformu čini manje fleksibilnom za specifična kućna pravila. Implementacija drugih RPG sustava je i dalje veliki i zahtjevni posao, uz implementirane entitete koje platforma pruža za proširenje. Osim toga, sustav za sinkroniziranje podataka između klijenata je trenutno naivno implementiran. Bolja implementacija bi smanjila količinu podataka koji se šalju između klijenata, npr. samo promijenjeni podaci o entitetu. Također, korisničko sučelje, iako funkcionalno, zahtijeva daljnju doradu u smislu intuitivnosti i preglednosti za korisnike.

U budućnosti, projekt se može proširiti u nekoliko smjerova. Puno bi pomogao razvoj sustava za uvoz statistika čudovišta i čarolija, što bi uvelike smanjilo potrebni ručni rad za implementaciju detalja. Također, uvođenje sustava za dinamičku rasvjetu (fog of war), dodavanje koncepta zidova i vertikalnih razina te podrške za zvučne efekte značajno bi doprinijelo atmosferi igre.

Rad predstavlja čvrst temelj za daljnji razvoj digitalnih alata u sferi igara igranja uloga, potvrđujući da tehnologija ne mora nužno zamijeniti maštu, već joj može pružiti okvir za lakše i zabavnije funkcioniranje.

Popis literature

[1] J. H. Mann, „Experimental evaluations of role playing.,” *Psychological Bulletin*, sv. 53, br. 3, str. 227, 1956.

[2] W. J. White, J. Arjoranta, M. Hitchens, J. Peterson, E. Torner i J. Walton, „Tabletop role-playing games,” *The Routledge Handbook of Role-Playing Game Studies*, Routledge, 2024.

[3] J. Peterson, *Playing at the World: A History of Simulating Wars, People and Fantastic Adventures, from Chess to Role-playing Games*. Unreason Press, 2012., ISBN: 9780615642048.

[4] P. Sidhu, M. Carter i J. P. Zagal, *Fifty Years of Dungeons & Dragons*. MIT Press, 2024.

[5] M. Mearls i J. Crawford, *Player's Handbook*. Wizards of the Coast, 2014.

[6] M. Mearls i J. Crawford, *Dungeon Master's Guide*, 5th. Renton, WA: Wizards of the Coast, 2014., ISBN: 978-0786965625.

[7] W. of the Coast, *Dungeons and Dragons Character Sheet*, `https://media.wizards.com/2016/dnd/downloads/5E_CharacterSheet_Fillable.pdf`, Pristupano 26.11.2025, 2014.

[8] *What is Tauri?* `https://tauri.app/start`, Pristupano 12.12.2025.

[9] L. Shklar i R. Rosen, *Web application architecture*. John Wiley & Sons, 2009., sv. 36.

[10] S. Casteleyn, F. Daniel, P. Dolog, M. Matera i dr., *Engineering web applications*. Springer, 2009., sv. 30.

[11] *Rust Programming Language*, `https://rust-lang.org/`, Pristupano 12.12.2025.

[12] V. Pimentel i B. G. Nickerson, „Communicating and displaying real-time data with web-socket,” *IEEE Internet Computing*, sv. 16, br. 4, str. 45–53, 2012.

[13] B. Gupta i M. Vani, „An overview of web sockets: The future of real-time communication,” *Int. Res. J. Eng. Technol. IRJET*, sv. 5, br. 12, 2018.

[14] *Axum*, `https://docs.rs/axum/latest/axum/`, Pristupano 13.12.2025.

[15] *Socketioxide*, `https://github.com/Totodore/socketioxide`, Pristupano 13.12.2025.

[16] J. Kreibich, *Using SQLite*. " O'Reilly Media, Inc.", 2010.

[17] *Sensible SQLite defaults*, `https://briandouglas.ie/sqlite-defaults/`, Pristupano 15.12.2025.

[18] *Pixi.js Guides*, `https://pixijs.com/8.x/guides/getting-started/intro`, Pristupano 17.12.2025.

[19] *Pixi.js Architecture*, https://pixijs.com/8.x/guides/concepts/architecture, Pristupano 17.12.2025.

[20] *React*, https://react.dev/, Pristupano 17.12.2025.

[21] *Jotai*, https://jotai.org/, Pristupano 17.12.2025.

Popis slika

Popis tablica