

**SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
VARAŽDIN**

Luka Kukec

TOP-DOWN SURVIVAL SHOOTER SA ZODB I PYGAME

PROJEKT

TEORIJA BAZA PODATAKA

Varaždin, 2026.

SVEUČILIŠTE U ZAGREBU
FAKULTET ORGANIZACIJE I INFORMATIKE
V A R A Ź D I N

Luka Kukec

Matični broj: 0016158557

Studij: Informacijski i poslovni sustavi

TOP-DOWN SURVIVAL SHOOTER SA ZODB I PYGAME

PROJEKT

Mentor:

prof. dr. sc. Bogdan Okreša Đurić

Varaždin, siječanj 2026.

Luka Kukec

Izjava o izvornosti

Izjavljujem da je ovaj projekt 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 prihvatanjem odredbi u sustavu FOI Radovi

Sažetak

Ovaj projekt prikazuje razvoj računalne igre žanra Top-Down Survival Shooter korištenjem objektno-orijentirane baze podataka ZODB i PyGame frameworka za grafičko sučelje. Glavni cilj projekta je demonstrirati prednosti korištenja objektno baze podataka za perzistenciju velikog broja dinamičkih objekata (poput metaka i neprijatelja), izbjegavajući pritom problem nepodudarnosti impedancije (impedance mismatch) karakterističan za relacijske baze. Aplikacija implementira sustav za praćenje igrača, aktivnih projektila, neprijatelja i rezultata (High Scores). Poseban naglasak stavljen je na implementaciju ACID transakcija, korištenje B-stabala (BTree) za indeksiranje rezultata, te implementaciju poslovne logike kroz metode modela i "property" settere koji djeluju kao okidači (triggers). Rezultat je robusna aplikacija koja transparentno sprema stanje stotina objekata u `.fs` datoteku.

Ključne riječi: ZODB; objektno baze podataka; PyGame; Survival Shooter; Python; perzistencija; transakcije; okidači; BTree

Sadržaj

1. Uvod	1
1.1. Opis Aplikacijske Domene	1
1.2. Motivacija i Cilj	1
2. Teorijski okvir	2
2.1. Objektno-Orijentirane Baze Podataka (OOBP)	2
2.2. Usporedba s Relacijskim Pristupom	2
2.3. ACID Svojstva i Transakcije	2
3. Model Baze Podataka	3
3.1. Struktura Korijenskog Objekta	3
3.2. Klase Podataka	4
3.2.1. Klasa Player	4
3.2.2. Klasa Item	4
4. Implementacija	5
4.1. Arhitektura Sustava	5
4.2. Upravljanje Bazom (GameDB klasa)	5
4.3. Implementacija Okidača (Triggers)	5
4.4. Game Loop i Transakcije	6
5. Primjeri Korištenja	7
5.1. Instalacija i Pokretanje	7
5.2. Scenarij Igranja	7
5.3. Resetiranje Baze	7
6. Zaključak	8
Popis literature	8
Popis slika	9
Popis tablica	10

1. Uvod

1.1. Opis Aplikacijske Domene

Računalne igre su kompleksni softverski sustavi koji zahtijevaju upravljanje velikim brojem stanja u stvarnom vremenu. U žanru Top-Down Survival Shooter (TDSS) igara, stanje uključuje veliki broj aktivnih entiteta koji se brzo mijenjaju: pozicije stotina metaka, koordinate neprijatelja koji nadiru u valovima, te stanje igrača (zdravlje, *power-up* statusi). Tradicionalni pristup korištenjem relacijskih baza podataka zahtijeva mapiranje svakog metka ili neprijatelja u redak tablice, što je iznimno neefikasno za ovakav tip "živog" sustava.

Ovaj projekt implementira jednostavnu 2D TDSS igru gdje igrač upravlja likom, bori se protiv hordi neprijatelja i skuplja *power-up* predmete. Svi podaci o igri – uključujući svaki ispaljeni metak i svakog neprijatelja na ekranu – moraju biti trajno pohranjeni kako bi igrač mogao nastaviti igru točno tamo gdje je stao.

1.2. Motivacija i Cilj

Glavna motivacija za odabir Zope Object Database (ZODB) tehnologije je njezina sposobnost transparentne perzistencije Python objekata. Cilj je pokazati kako se objektna baza podataka može integrirati u petlju igre (Game Loop) te kako olakšava razvoj eliminirajući potrebu za SQL upitima i ORM (Object-Relational Mapping) slojevima.

2. Teorijski okvir

2.1. Objektno-Orijentirane Baze Podataka (OOBP)

Objektno-orijentirana baza podataka (Objektno-orijentirana baza podataka (OOBP)) pohranjuje podatke u obliku objekata, baš onako kako su predstavljeni u programskom jeziku. Ovo rješava problem *impedance mismatch*-a, odnosno nesrazmjera između objektnog modela aplikacije i relacijskog modela baze podataka **OOBP2015**.

ZODB je nativna objektna baza za Python koja pruža sljedeća svojstva:

- **Transparentnost:** Objekti se automatski spremaju kada su promijenjeni.
- **ACID Transakcije:** Osiguravaju integritet podataka.
- **Povijest i Undo:** Mogućnost vraćanja na prethodna stanja.

2.2. Usporedba s Relacijskim Pristupom

Usporedba ZODB-a i klasičnih relacijskih baza (poput PostgreSQL-a) u kontekstu razvoja igara prikazana je u tablici 1.

Tablica 1: Usporedba ZODB i Relacijskih baza

Svojstvo	ZODB (Objektna)	PostgreSQL (Relacijska)
Model podataka	Python klase i objekti	Tablice i redci
Upiti	Navigacija po grafu, BTree	SQL (SELECT, JOIN)
Odnosi	Direktne reference	Strani ključevi (Foreign Keys)
Perzistencija	<code>transaction.commit()</code>	SQL INSERT/UPDATE
Fleksibilnost	Visoka (promjena sheme)	Niska (ALTER TABLE)

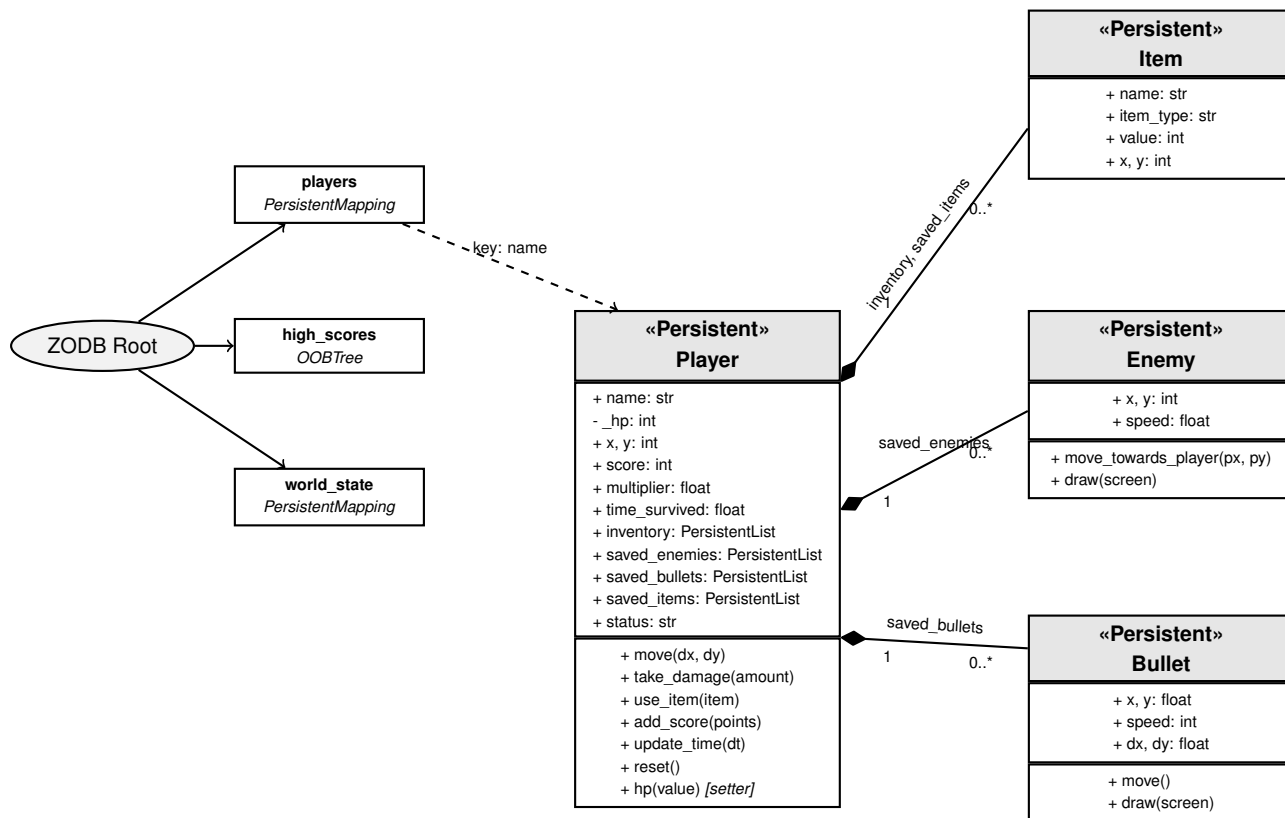
2.3. ACID Svojstva i Transakcije

Sigurnost podataka u igri osigurana je ACID svojstvima **ACID2020**:

- **Atomarnost:** Cijela promjena stanja (npr. igrač pokupi predmet i dobije bodove) se sprema odjednom.
- **Konzistentnost:** Baza prelazi iz jednog validnog stanja u drugo.
- **Izolacija:** Ako bi igra podržavala više igrača, njihove transakcije se ne bi miješale.
- **Trajnost:** Jednom kada se napravi `commit`, podaci su sigurni na disku.

3. Model Baze Podataka

Za razliku od ER dijagrama kod relacijskih baza, model ZODB-a najbolje se opisuje grafom objekata koji kreću od korijenskog (root) objekta. Slika 1 prikazuje UML dijagram klasa implementiranih u sustavu.



Slika 1: UML Model podataka s prikazom ZODB hijerarhije

3.1. Struktura Korijenskog Objekta

Korijenski objekt (`db.root`) sadrži tri glavne kolekcije:

1. `players` (`PersistentMapping`): Mapa koja čuva objekte igrača, gdje je ključ ime igrača.
2. `high_scores` (`OOBTree`): B-stablo za efikasno čuvanje i dohvaćanje najboljih rezultata. `OOBTree` je optimiziran za velike količine podataka i pretraživanje raspona.
3. `world_state` (`PersistentMapping`): Spremište za globalno stanje svijeta (npr. ime zadnjeg igrača).

3.2. Klase Podataka

Glavni entiteti u sustavu su `Player` i `Item`. Obje klase nasljeđuju `Persistent` kako bi ih ZODB mogao pratiti.

3.2.1. Klasa Player

Predstavlja igrača u igri. Sadrži logiku kretanja, zdravlja i inventara.

Tablica 2: Atributi klase Player

Atribut	Tip	Opis
<code>name</code>	<code>string</code>	Jedinstveno ime igrača (ID).
<code>_hp</code>	<code>int</code>	Trenutno zdravlje (0-100).
<code>x, y</code>	<code>int</code>	Koordinate na ekranu.
<code>score</code>	<code>int</code>	Trenutni rezultat.
<code>multiplier</code>	<code>float</code>	Množitelj bodova (povećava se prikupljanjem).
<code>time_survived</code>	<code>float</code>	Vrijeme preživljavanja u sekundama.
<code>inventory</code>	<code>PersistentList</code>	Lista prikupljenih predmeta.
<code>saved_enemies</code>	<code>PersistentList</code>	Lista aktivnih neprijatelja (za save/load).
<code>saved_bullets</code>	<code>PersistentList</code>	Lista aktivnih metaka (za save/load).
<code>saved_items</code>	<code>PersistentList</code>	Lista ne-prikupljenih predmeta na mapi.
<code>status</code>	<code>string</code>	Stanje ("Active", "Defeated").

3.2.2. Klasa Item

Predstavlja predmete koje igrač može pokupiti.

- `name`: Naziv predmeta (npr. "Drop_heal").
- `item_type`: Tip predmeta ("heal" ili "score").
- `value`: Numerička vrijednost (količina liječenja ili bodova).
- `x, y`: Pozicija na mapi prije nego je pokupljen.

4. Implementacija

Aplikacija je implementirana u programskom jeziku Python koristeći PyGame biblioteku za grafiku i ulazne uređaje, te ZODB za sloj podataka.

4.1. Arhitektura Sustava

Projekt je podijeljen u module radi bolje organizacije:

- `src/database.py`: Upravlja konekcijom prema `game.fs` datoteci i inicijalizira strukture.
- `src/models.py`: Definira klase podataka i poslovnu logiku.
- `src/main.py`: Sadrži *Game Loop*, upravljanje događajima i iscrtavanje.

4.2. Upravljanje Bazom (GameDB klasa)

Klasa `GameDB` enkapsulira sve operacije nad bazom. Prilikom inicijalizacije provjerava postojanje `data/` direktorija i kreira potrebne strukture unutar `db.root` ako ne postoje.

```
1 def get_top_scores(self, limit=5):
2     """ Vraca top rezultate koristeći BTree efikasnost """
3     items = list(self.root['high_scores'].items())
4
5     # Sortiranje po bodovima (prvi element u tuple-u vrijednosti)
6     items.sort(key=lambda x: x[1][0], reverse=True)
7
8     # Formatiranje povrata: (name, score, time)
9     return [(name, val[0], val[1]) for name, val in items[:limit]]
```

Isječak koda 1: Metoda za dohvat Top Score-ova koristeći BTree

4.3. Implementacija Okidača (Triggers)

ZODB nema klasične SQL okidače, ali se oni elegantno implementiraju koristeći Python *property* mehanizam. U klasi `Player`, postavljanje HP-a automatski provjerava uvjet poraza.

```
1 @hp.setter
2 def hp(self, value):
3     self._hp = min(100, max(0, value))
4     # OKIDAC: Ako je HP 0, promijeni status u "Defeated"
5     if self._hp == 0:
6         self.status = "Defeated" # Automatska promjena
7     # Javljam ZODB-u da spremi promjenu
8     self._p_changed = True
```

Isječak koda 2: Implementacija okidača za promjenu statusa

Ovaj pristup osigurava da je poslovna logika ("igrač je poražen ako je HP=0") centralizirana i ne može se zaobići, bez obzira na to koji dio koda mijenja HP.

4.4. Game Loop i Transakcije

Igra se odvija u beskonačnoj petlji koja obrađuje ulaze, ažurira stanje i iscrtava sliku. Spremanje u bazu (commit) se ne radi u svakom frame-u zbog performansi, već u ključnim trenucima:

1. Prilikom izlaza iz igre.
2. Prilikom ažuriranja High Score tablice (kraj igre).
3. Na zahtjev korisnika (tipka ESC za povratak u meni).

5. Primjeri Korištenja

5.1. Instalacija i Pokretanje

Aplikacija dolazi s instalacijskom skriptom `setup.py` koja instalira potrebne biblioteke definirane u `requirements.txt`. Pokretanje igre vrši se naredbom: `python src/main.py`

5.2. Scenarij Igranja

1. **Početak:** Igrač pokreće igru. Ako je prvi put, kreira se novi lik "Igrac1" na poziciji (400, 300).
2. **Akcija:** Igrač se kreće tipkama W, A, S, D i puca lijevim klikom miša.
3. **Prikupljanje:** Ubijanjem neprijatelja (crveni krugovi) ispadaju predmeti (žuti kvadrati). Igrač ih skuplja prelaskom preko njih, čime se oni dodaju u `PersistentList` inventar i odmah primjenjuju (povećanje HP-a ili bodova).
4. **Spremanje:** Igrač mora prekinuti igru. Pritiskom na 'ESC', igra se pauzira, vraća u meni i poziva se `transaction.commit()`.
5. **Nastavak:** Sljedeći dan igrač ponovno pokreće igru. Lik se nalazi na točno istoj poziciji, s istim brojem bodova i istim inventarom kao prije gašenja.
6. **Game Over:** Ako HP padne na 0, igra završava. Rezultat se upisuje u `high_scores` B-stablo u bazi. Prikazuje se lista najboljih rezultata dohvaćena iz baze.

5.3. Resetiranje Baze

Za potrebe testiranja ili novog početka, priložena je skripta `reset_db.py` koja briše `.fs` datoteku i omogućuje čisti start.

6. Zaključak

ZODB se pokazao iznimno efikasnim za razvoj kompleksne logike igre. Ključne prednosti koje su uočene tijekom razvoja su:

- **Brzina razvoja:** Nema potrebe za pisanjem SQL shema i migracija. Dodavanje novog atributa u klasu `Player` automatski je podržano.
- **Prirodna reprezentacija:** Graf objekata u memoriji se preslikava 1:1 na disk.
- **Pouzdanost:** ACID transakcije osiguravaju da se inventar ne izgubi čak ni u slučaju rušenja aplikacije (pod uvjetom da je napravljen commit).

Ovaj projekt uspješno demonstrira da su objektne baze podataka validna i često superiorna alternativa relacijskim bazama za domene koje su inherentno objektne, kao što su računalne igre i simulacije.

Popis slika

1.	UML Model podataka s prikazom ZODB hijerarhije	3
----	--	---

Popis tablica

1.	Usporedba ZODB i Relacijskih baza	2
2.	Atributi klase Player	4