UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA

UNIVERZITET U NOVOM SADU
FAKULTET TEHNIČKIH NAUKA
NOVI SAD

ISPITNI RAD

Kandidat: **Marko Sladojević**
Broj indeksa: **12345**

Predmet: **Objektno orijentisano programiranje 2**
Tema rada: **Robot u Knososu**

Mentor rada: **dr Dušan Kenjić**

Novi Sad, jul, 2025.

SADRŽAJ

SPISAK SLIKA

SPISAK TABELA

1. Uvod

1.1 Robot u Knososu

Lavarint u Knososu na ostrvu Krit bio je ozloglašena građevina minojske civilizacije stvorena od strane jednog od najvećih izumitelja iz antičkih mitova - Dedala. Ovaj lavirint je predstavljao nemoguću misiju za tadašnjeg običnog čoveka i u njemu je po predanju stradalo mnogo ljudi. Dodatni izazov predstavljalo je mitsko biće Minotaur koji je lutao lavirintom i predstavljao smrtnu opasnost svima koji su pokušali da pronađu izlaz.

U ovom projektu potrebno je iskoristiti programerske veštine da se isprogramira Tezejev robot koji bi uspeo da pronađe izlaz u ovakvom lavirintu. Robot mora da se kreće kroz lavirint izbegavajući zidove i Minotaura, koristeći specijalne predmete koji mu mogu pomoći u njegovoj misiji. Igra kombinuje elemente strategije, nasumičnosti i interaktivnog igranja kroz konzolu.

Slika 1: Prikaz lavirinta sa "*health barovima*" efekata

1.2 Zadatak

Cilj projekta je implementacija interaktivne igre "Robot u Knososu" u programskom jeziku C++. Program treba da omogući:

- **Dinamičko generiranje lavirinta** na osnovu dimenzija zadatih kroz argumente komandne linije, sa minimalnim dimenzijama 16x16
- **Implementaciju algoritma za kreiranje lavirinta** u zasebnoj biblioteci, sa merenjem vremena potrebnog za generisanje
- **Interaktivno upravljanje robotom** kroz konzolu sa komandama za kretanje u četiri smera
- **Simulaciju Minotaurovog ponašanja** koji se kreće nasumično ili lovi robota kada mu se približi
- **Sistem specijalnih predmeta** sa efektima koji traju tri poteza: "Magla rata" (ograničava vidljivost na podmatricu *3x3* oko robota), "Mač" (uništava Minotaura), "Štit" (odbrana od Minotaura) i "Čekić" (prolazak kroz zidove)
- **Korišćenje naprednih OO koncepata** kao što su nasleđivanje za implementaciju predmeta i njihovih efekata
- **Upravljanje datotekama** za čuvanje konačnog stanja igre
- **Testiranje funkcionalnosti** u slobodnoj formi sa dokumentovanim test slučajevima

Program mora da poštuje sva pravila igre, koristi adekvatne strukture podataka, pravilno upravlja dinamičkim memorijskim resursima i bude podeljen u smislene klase i module uz poštovanje standarda kodiranja.

2. Analiza problema

2.1 Izazovi problema

Glavni problem u implementaciji igre "Robot u Knososu" leži u projektovanju modularne arhitekture koja omogućava jasno razdvajanje funkcionalnosti i efikasno upravljanje resursima. S obzirom da se radi o interaktivnoj igri u realnom vremenu, potrebno je obezbediti sinhrono funkcionisanje različitih komponenti sistema.

Ključni izazovi uključuju **dinamičko generiranje lavirinta** sa algoritmom koji mora da garantuje postojanje puta od ulaza do izlaza, što zahteva implementaciju algoritma pretrage ili generativnog algoritma koji održava povezanost putanje. **Upravljanje stanjem igre** predstavlja složen problem jer je potrebno simultano pratiti pozicije robota, Minotaura i predmeta, kao i aktivne efekte koji imaju vremensko ograničenje od tri poteza.

Sistem polja matrice i predmeta predstavlja složen problem objektno-orijentisanog dizajna, gde je potrebno implementirati nasleđivanje i polimorfizam za različite tipove polja matrice („Bazno polje“, „Zid“, „Prolaz“, „Ulaz“, „Izlaz“) predmeta ("Magla rata", "Mač", "Štit", "Čekić") pri čemu svaki ima specifično ponašanje i uticaj na stanje igre.

Upravljanje memorijskim resursima je kritično zbog dinamičkog kreiranja lavirinta čija veličina može značajno varirati (minimum 15x15, ali može biti i mnogo veće). Potrebno je obezbediti pravilno oslobađanje memorije i izbegavanje memory leak-ova.

Korisničko iskustvo i interfejs zahtevaju responzivno upravljanje kroz konzolu sa jasnim prikazom trenutnog stanja lavirinta, posebno kada je aktivna "Magla rata" koja ograničava vidljivost na 3x3 oblast. Cilj je u optimizaciji korisničkog iskustva.

Testiranje i validacija predstavljaju dodatni izazov jer je potrebno testirati različite scenarije igre, uključujući granične slučajeve kao što su minimalne dimenzije lavirinta, različite

kombinacije predmeta i njihovih efekata, kao i sve moguće ishode igre (pobeda, poraz, prekidanje).

Konačno, **merenje performansi** algoritma za generisanje lavirinta u zavisnosti od dimenzija predstavlja važan aspekt za optimizaciju i analizu skalabilnosti rešenja.

2.2 Razmatranje pristupa rešavanju problema

Dinamičko generiranje lavirinta

Pri razmatranju algoritma za generisanje lavirinta, postojalo je nekoliko opcija. ***Depth-First Search*** **(DFS)** bi kreirao dugačke, zavijugane putanje ali sa malo razgranjavanja. ***Kruskalov algoritam*** bi dao uniformnu distribuciju prolaza, ali kompleksniju implementaciju. ***Randomizovani Primov algoritam*** je odabran jer balansira kompleksnost implementacije sa kvalitetom rezultata - kreira lavirinte sa dovoljno razgranjavanja da budu zanimljivi, a algoritam je dovoljno jednostavan za pouzdanu implementaciju. Dodatno, ovaj algoritam prirodno garantuje povezanost svih delova lavirinta.

Problem garantovanja povezanosti ulaza i izlaza zahtevao je posebnu pažnju. Umesto oslanjanja samo na generativni algoritam, odlučeno je da se implementira eksplicitna provera i korekcija putanje između robota i izlaza. Ovo osigurava da svaki generisani lavirint bude rešiv, što je fundamentalni zahtev za funkcionalnu igru.

Upravljanje stanjem igre

Stanje igre je inherentno kompleksno - mora da prati pozicije, efekte predmeta, broj poteza, i različite uslove za završetak. ***Event-driven pristup*** se nametnuo kao prirodan izbor jer igra reaguje na korisničke akcije. Alternativa bi bila ***turn-based sistem*** sa eksplicitnim fazama, ali to bi bilo rigidnije i manje responzivno.

Za upravljanje vremenskim efektima predmeta, razmatrane su opcije kao što su ***timer-based sistem*** ili ***turn-based counters***. Turn-based pristup je odabran jer je lakši za implementaciju - svaki potez robota decrementuje sve aktivne efekte za jedan.

Arhitektura objektno-orijentisanog dizajna

Hijerarhija klasa za polja matrice zahtevala je pažljivo planiranje. ***Kompozicijski pristup*** (gde bi Matrix sadržavao različite tipove podataka) bi bio jednostavniji, ali manje fleksibilan. ***Nasleđivanje*** omogućava polimorfno ponašanje - svako polje "zna" kako da se prikaže i da li je prolazno, što čini kod čitljivijim i lakšim za proširavanje.

Za predmete, ***Strategy pattern*** je razmatran gde bi svaki predmet imao svoju strategiju aktivacije. Međutim, **nasleđivanje sa *Template Method*** je odabrano jer su efekti predmeta dovoljno slični da dele zajedničku logiku, a razlike su uglavnom u tipu efekta koji aktiviraju.

Upravljanje memorijskim resursima

Dinamička alokacija matrice predstavlja klasičan trade-off između performansi i fleksibilnosti. ***Statička alokacija*** bi bila brža, ali ograničila bi maksimalne dimenzije lavirinta. ***std::vector*** bi bio sigurniji, ali možda sporiji za često pristupanje elementima. ***Ručna dinamička alokacija*** je odabrana za maksimalnu kontrolu i performanse, uz pažljivo implementiran *RAII* pattern kroz destruktor.

Korisničko iskustvo i optimizacija prikaza

Najveći izazov korisničkog iskustva je bilo efikasno ažuriranje prikaza. Ponovno ispisivanje cele matrice nakon svakog poteza bi bilo previse neefikasno i lose estetski. Zato biramo ažuriranje matrice na konzoli.

Potpuno osvežavanje ekrana na svaki potez bi bilo jednostavno ali neefikasno. ***ANSI escape sekvence*** omogućavaju precizno pozicioniranje kursora i ažuriranje samo onih karaktera koji su se promenili. Ovo dramatično poboljšava responzivnost, posebno za velike lavirinte.

Za vizuelne efekte kao što je "magla rata", razmatrane su opcije ***potpunog prekrivanja*** matrice ili ***selektivnog prikazivanja***. Selektivni pristup je odabran jer omogućava glatki prelazak između normalnog i ograničenog prikaza bez gubitka informacija o pozicijama.

Performanse i sklabilnost

Merenje performansi generisanja lavirinta je ključno za razumevanje kako algoritam skalira sa veličinom. ***Mikrosekunde*** pružaju dovoljnu preciznost za analizu, a poređenje sa ***milisekundama*** čini rezultate razumljivijim korisniku.

Upravljanje datotekama i perzistentnost

Timestamp-based nazivi datoteka izbegavaju konflikte i omogućavaju prirodno sortiranje rezultata.

3. Analiza problema

3.1 Sekvencijalni program – moduli i osnovne metode

3.1.1 Glavni program (main)

```
int main(int argc, char* argv[]);
```

Glavna funkcija programa. Parsira argumente komandne linije (dimenzije lavirinta i broj predmeta), inicijalizuje igru i pokreće glavni game loop.

3.1.2 Klasa Matrix

Modul za kreiranje i upravljanje lavirintom.

generateMatrix(unsigned int no_of_items)

```
microseconds generateMatrix(unsigned int no_of_items);
```

Generiše kompletan lavirint koristeći Randomizovani Primov algoritam, postavlja ulaz/izlaz, raspoređuje predmete i meri vreme generisanja.

generativePrim(unsigned int entrance_x)

```
void generativePrim(unsigned int entrance_x);
```

Implementira Primov algoritam za kreiranje lavirinta počevši od robot-ove pozicije.

assurePathConnectivity(unsigned int exit_x)

```
void assurePathConnectivity(unsigned int exit_x);
```

Garantuje postojanje putanje između robot-a i izlaza kreiranjem eksplicitne konekcije.

3.1.2 Klasa Gameplay

Modul za upravljanje tokom igre i interakcijom sa korisnikom.

initializeGame(unsigned int no_of_items)

```
void initializeGame(unsigned int no_of_items);
```

Inicijalizuje i generiše matricu, postavlja početne pozicije robota i Minotaura, prikazuje uvodne poruke.

startGameLoop()

```
void startGameLoop();
```

Glavni loop igre koji prima korisničke unose, ažurira pozicije, proverava uslove za završetak igre.

moveMinotaur(unsigned int prev_x, unsigned int prev_y)

```
void moveMinotaur(unsigned int prev_x, unsigned int prev_y);
```

Implementira logiku kretanja Minotaura - nasumično kretanje ili praćenje robota kada je u dometu.

3.1.3 Hijerarhija MatrixField klasa

Polimorfni sistem za različite tipove polja u lavirintu.

MatrixField (bazna klasa)

```
virtual FieldType getFieldType() const = 0;

virtual char getSymbol() const = 0;

virtual bool isWalkable() const = 0;
```

Izvedene klase: Passage, Wall, Entrance, Exit, Item (sa podklasama Sword, Shield, Hammer, FogOfWar).

3.1.4 Klasa FileHandler

Modul za upravljanje datotekama i čuvanje rezultata igre.

saveGameResult(...)

```
bool saveGameResult(const Matrix* matrix, unsigned int robot_x, unsigned int robot_y,
                    GameResult result, const microseconds& game_duration, unsigned int moves_made);
```

Kreira timestamp-based datoteku sa kompletnim stanjem igre i statistikama partije.

3.2 Pomoćni moduli

3.2.1 ConsoleHandler

Upravljanje ANSI escape sekvencama za optimizovano ažuriranje prikaza.

3.2.2 RNGEngine

Centralizovano upravljanje Mersenne Twister random number generator-om za konzistentne rezultate.

4. Testiranje

4.1 Testni skupovi

Testiranje programa izvršeno je kroz nekoliko testnih skupova koji pokrivaju različite aspekte funkcionalnosti.

4.1.1 Test 1 - Testiranje generisanja lavirinta

Cilj testa: Provera ispravnosti generisanja lavirinta i merenje performansi algoritma za različite dimenzije.

Testni slučajevi:

- Lavirint dimenzija 16x16 sa 5 predmeta
- Lavirint dimenzija 30x30 sa 12 predmeta
- Lavirint dimenzija 50x50 sa 20 predmeta
- Lavirint dimenzija 100x100 sa 35 predmeta

Rezultati:

- Svi lavirinti su uspešno generisani
- Svi lavirinti sadrže ispravan broj predmeta
- U svim lavirintima postoji put od ulaza do izlaza
- Randomizovani Primov algoritam garantuje povezanost

Veličina	15x15	30x30	50x50	100x100
Vreme	631µs	1552µs	5,125ms	17,755ms

Tabela 1: Vremena izvršenja generisanja lavirinta za različite dimenzije

4.1.2 Test 2 - Testiranje kretanja

Cilj testa: Provera ispravnosti implementacije kretanja robota i Minotaura, uključujući ponašanje i ograničenja kretanja.

Testni slučajevi:

- Kretanje robota u svim smerovima (WASD)
- Pokušaj kretanja robota kroz zid
- Kretanje Minotaura u normalnom režimu (nasumično)
- Kretanje Minotaura kada je robot u dometu (praćenje)
- *Collision detection* između robota i Minotaura

Rezultati:

- Robot se ispravno kreće u svim smerovima
- Robot ne može proći kroz zid (osim kada ima čekić)
- Minotaur se ispravno kreće nasumično kada robot nije blizu
- Minotaur precizno prati robota kada je u dometu od jednog polja
- Kolizija se ispravno detektuje i game over se aktivira

4.1.3 Test 3 - Testiranje sistema predmeta i efekata

Cilj testa: Provera ispravnosti implementacije predmeta i njihovih specijalnih efekata.

Testni slučajevi:

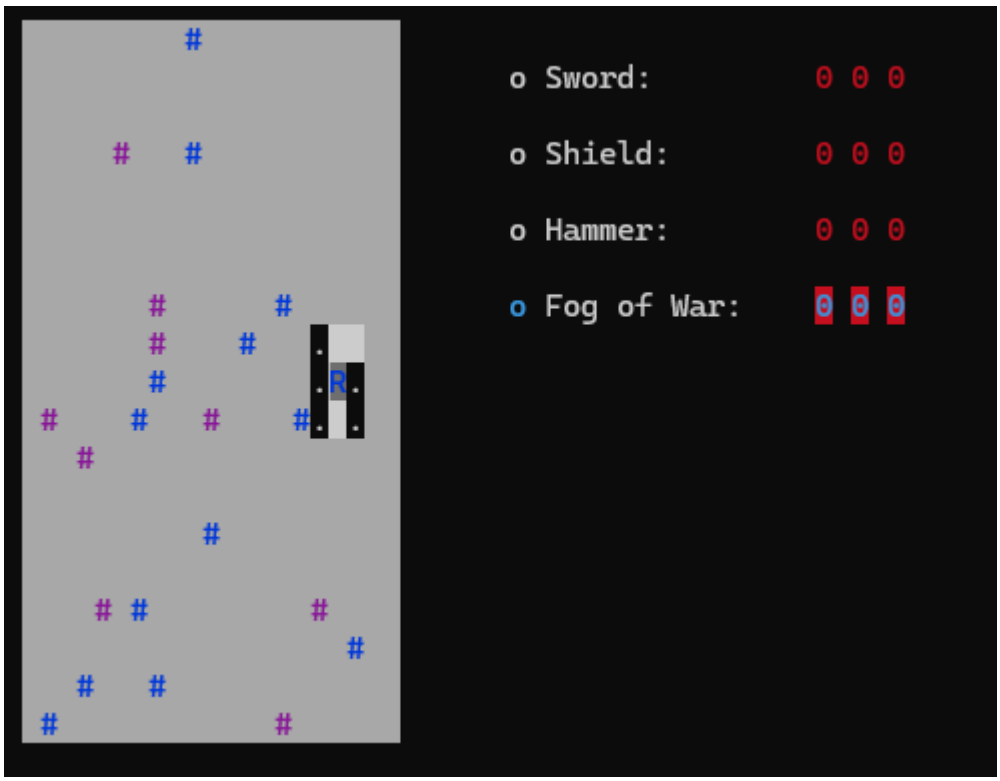
- Aktiviranje efekta "Magla rata" (ograničava vidljivost na 3x3)
- Aktiviranje efekta "Mač" (omogućava ubijanje Minotaura)
- Aktiviranje efekta "Štit" (odbija Minotaura na udaljenost od 2 polja)
- Aktiviranje efekta "Čekić" (omogućava prolazak kroz zidove)
- Provera trajanja efekata (4 poteza umesto 3)
- Simultana aktivacija više efekata

Rezultati:

- Svi efekti se ispravno aktiviraju nasumičnim odabirom
- Efekti traju tačno 3 poteza
- Magla rata ispravno ograničava vidljivost i generiše animiranu maglu
- Štit uspešno "*bounce-uje*" Minotaura na validne pozicije
- Čekić omogućava prolazak kroz unutrašnje zidove (ne i granične)

4. Odabrane slike

5. Odabrane slike



4. Odabrane slike

Slika 2: *Fog of War* efekat

4. Odabrane slike

6. Zaključak

Napravljen je jedan koncept za realizaciju igre "Robot u Knososu" koji kombinuje objektno-orijentisan dizajn sa efikasnim algoritmima i optimizovanim korisničkim interfejsom. Implementiran je i verifikovan kompletan sistem koji obuhvata sve aspekte problema opisane u zadatku.

Verifikacija programa urađena je kroz sistematsko testiranje funkcionalnosti pomoću različitih test skupova. Time je potvrđena ispravnost svakog pojedinačnog modula (Matrix, Gameplay, FileHandler, MatrixField hijerarhija), njihovih veza i programa u celini. Posebna pažnja posvećena je testiranju granične slučajeva kao što su minimalne dimenzije lavirinta, edge case-ovi za kretanje kroz zidove i validacija AI logike Minotaura.

Program je pokretan za različite testne scenarije na istom računaru u cilju merenja performansi generisanja lavirinta. Rezultati merenja (Tabela 1) pokazuju da Randomized Prim's algoritam ima prihvatljivu kompleksnost za praktične dimenzije lavirinta. Značajno ubrzanje se postiže korišćenjem ANSI escape sekvenci za ažuriranje prikaza umesto potpunog refresh-a ekrana, što dramatično poboljšava responzivnost igre.

4. Odabrane slike

Najveće performanse se gube na velikim lavirintima (100x100+) gde algoritam generisanja postaje sporiji, ali i dalje ostaje u prihvatljivim granicama za interaktivnu igru. Optimizacija bi mogla da se postigne prelaskom na iterativnu implementaciju umesto rekurzivne, ili korišćenjem naprednih struktura podataka za frontier management.

Implementacija ANSI-based interfejsa pokazala se kao ključni faktor korisničkog iskustva. Precizno cursor positioning i selective character updates omogućavaju glatko igranje čak i na lavirintima velikih dimenzija, dok bi tradicionalni approach potpunog refresh-a bio neupotrebljiv za praktičnu upotrebu.

Objektno-orijentisan pristup kroz hijerarhiju MatrixField klasa omogućio je laku proširivost sistema - dodavanje novih tipova predmeta ili polja zahteva minimalne izmene postojećeg koda. Polimorfizam kroz virtuelne funkcije čini kod čitljivijim i jednostavnijim za održavanje.

Sistem upravljanja datotekama sa timestamp-based imenima datoteka pokazao se kao robustan i praktičan za praćenje progresa igrača kroz vreme, omogućavajući prirodno sortiranje i izbegavanje konflikata naziva.